UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELÉTRICA E INFORMÁTICA INDUSTRIAL
ESPECIALIZAÇÃO EM TELEINFORMÁTICA E REDES DE COMPUTADORES

OESLEI TABORDA RIBAS

SAFEFW: CONTROLE DE CONEXÕES REMOTAS

MONOGRAFIA DE ESPECIALIZAÇÃO

CURITIBA
2011

OESLEI TABORDA RIBAS

SAFEFW: CONTROLE DE CONEXÕES REMOTAS

Trabalho de Conclusão de Curso de pós-graduação, apresentado à disciplina Trabalho de Diplomação, da especialização em Teleinformática e Redes de Computadores do Departamento Acadêmico de Elétrica e Informática Industrial – DAELT – da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de especialista.

Orientador: Prof. Msc. Christian Carlos Souza Mendes

CURITIBA

2011

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS CURITIBA
DEPARTAMENTO ACADÊMICO DE ELÉTRICA E INFORMÁTICA INDUSTRIAL
ESPECIALIZAÇÃO EM TELEINFORMÁTICA E REDES DE COMPUTADORES
DISCIPLINA DE TRABALHO DE DIPLOMAÇÃO

TERMO DE APROVAÇÃO

SAFEFW: CONTROLE DE CONEXÕES REMOTAS

Oeslei Taborda Ribas

Este Trabalho de Diplomação foi considerado adequado como cumprimento das exigências legais do currículo da Especialização em Teleinformática e Redes de Computadores e aprovado em sua forma final pelo Departamento Acadêmico de Elétrica e Informática Industrial da Universidade Tecnológica Federal do Paraná – Campus de Curitiba.

NOTA: 9,0 (NOVE INTEIROS)

Prof. Msc. Christian Carlos Souza Mendes
Orientador

Prof. Dr. Walter Godoy Júnior
Coordenador do Curso de especialização
em Teleinformática e redes de computadores

Banca Examinadora:

Msc. Christian Carlos Souza Mendes

Dr. Walter Godoy Júnior

Dedico este trabalho à minha família,
pelos momentos de ausência.

AGRADECIMENTOS

Certamente estes parágrafos não irão atender a todas as pessoas que contribuíram para a realização deste trabalho. Portanto, desde já peço desculpas àquelas que não estão presentes entre essas palavras, mas elas podem estar certas que fazem parte do meu pensamento e de minha gratidão.

Agradeço ao meu orientador Prof. Msc. Christian Carlos Souza Mendes, pelos conselhos que me guiaram nesta trajetória. Ao coordenador do curso, Prof. Dr. Walter Godoy Júnior que sempre foi prestativo e atencioso. Aos demais professores do curso, pelo tempo dispensado para nos ensinar

Ao meu colega Hermano Pereira, o qual me ajudou a sanar algumas dúvidas de redes e de protocolos, contribuindo significativamente com os resultados alcançados. Ao colega José Roberto Andrade Júnior, que ajudou com a criação das figuras, fundamentais para o entendimento da arquitetura que está sendo proposta.

Sou grato à CELEPAR - Companhia de Informática do Paraná, que financiou parcialmente está pos-gradução. Instituição que estou vinculado atualmente a qual conta com toda minha dedicação e orgulho de ser um de seus funcionários.

Gostaria de deixar registrado também, o meu reconhecimento à minha família, pois acredito que sem o apoio deles seria muito difícil chegar até aqui, em especial a minha mãe e a minha irmã Priscila Taborda Ribas. E por último, e nem por isso menos importante, agradeço a minha linda filha Laura Bellincanta Taborda Ribas pelo carinho, amor e compreensão pela minha ausência.

Enfim, a todos os que por algum motivo contribuíram para a realização deste trabalho.

O fator humano é o elo mais fraco da segurança. (MITNICK, Kevin, 2003).

RESUMO EM LÍNGUA PORTUGUESA

RIBAS, Oeslei T. **SAFEFW:** controle de conexões remotas. 2011. 83 f. Dissertação (Especialização em Teleinformática e Redes de Computadores) – Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná. Curitiba, 2011.

Os protocolos de acesso remoto são utilizados para possibilitar que um usuário acesse remotamente um equipamento. Entre os protocolos mais comuns estão o SSH, RDP, RFB e Telnet. Esses protocolos proporcionam uma grande flexibilidade para os usuários, permitindo acesso rápido aos equipamentos sem a necessidade de proximidade física. Porém, a sua utilização gera alguns problemas relacionados à segurança. Entre os problemas gerados, pode-se citar a ocorrência de ataques de força bruta, dificuldade de gerenciamento das sessões e mecanismos que provenham informações detalhadas de todos os acessos realizados. Para sanar esses problemas é proposta uma nova arquitetura, baseada em um servidor central responsável por gerenciar todas as conexões e por prover mecanismos de segurança. A arquitetura proposta foi implantada em uma rede de grande porte. Os resultados obtidos demonstram que ela resolveu os problemas relacionados à segurança, além de ser estável e robusta.

Palavras-chave: Segurança. Acesso. Remoto. Controle. Conexões.

RESUMO EM LÍNGUA ESTRANGEIRA

RIBAS, Oeslei T. **SAFEFW:** controle de conexões remotas. 2011. 83 f. Dissertação (Especialização em Teleinformática e Redes de Computadores) – Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná. Curitiba, 2011.

The remote access protocols are used to enable a user to access a machine remotely. The most common protocols are SSH, RDP, RFB and Telnet. These protocols provide greater flexibility for users, allowing quick access to equipment without the need for physical proximity. However, their use cause some security issues. Among the issues raised there are: the occurrence of brute force attacks, difficulty in management connections and mechanisms that provide detailed information of all access made. To solve these problems, we propose a new architecture based on a central server responsible for managing all connections and for providing security mechanisms. The proposed architecture was implemented in a large network. The results show that it has solved the safety problems, and it is stable and robust.

Keywords: Security. Remote. Access. Management. Connection.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

ANSI	American National Standards Institute
API	Application Programming Interface
APT	Advanced Packaging Tool
ASP	Active Server Pages
CERT.br	Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil
CGI	Common Gateway Interface
CLI	Command Line Interface
CPAN	Comprehensive Perl Archive Network
CVS	Concurrent Versions System
DNS	Domain Name System
DSA	Digital Signature Algorithm
DTLS	Datagram Transport Layer Security
GCC	GNU Compiler Collection
GNU	GNU is Not Unix
GPL	GNU General Public License
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPSec	Internet Protocol Security
ISO	International Organization for Standardization
ITU	International Telecommunication Union
JCP	Java Community Proces
JDK	Java Development Kit
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
MAC	Message Authentication Code
MPPE	Microsoft Point-to-Point Encryption
MVCC	Multiversion Concurrency Control
ORL	Olivetti Research Laboratory
PAM	Pluggable Authentication Modules
PHP	PHP: Hypertext Preprocessor
RDP	Remote Desktop Protocol
RFB	Remote Framebuffer
RFC	Request for Comments
SQL	Structured Query Language
SSH	Secure Shell
SSL/TLS	Transport Layer Security/Secure Sockets Layer

SSTP	Secure Socket Tunneling Protocol
TCP	Transmission Control Protocol
UML	Unified Modeling Language
VNC	Virtual Network Computing
VPN	Virtual Private Network
XML	Extensible Markup Language

SUMÁRIO

INTRODUÇÃO

Com o surgimento das redes de computadores, foi possível o desenvolvimento de inúmeras tecnologias. O conceito inicial de redes de computadores era bem diferente do cenário que temos atualmente. Os dados eram centralizados em *mainframes* e já era possível o compartilhamento de arquivos. Os *links* de dados – que tinham um preço elevado inicialmente – permitiam o compartilhamento de dados entre as unidades das empresas, quase que instantaneamente (GOVERNANÇA, 2011).

A evolução da tecnologia e das aplicações de redes de computadores possibilitou um grande salto e abriram um gigantesco leque de possibilidades. Os arquivos não necessitam mais estar centralizados em *mainframes*, podem estar disponíveis em qualquer estação de trabalho. Os usuários não necessitam mais estar próximos geograficamente dos equipamentos. É possível estar a centenas de quilômetros de um computador e mesmo assim ter acesso aos seus dados, desde que ele esteja conectado na rede.

Uma das consequências dessa evolução tecnológica foi na maneira com que é realizada a administração dos equipamentos de rede, tais como: servidores, roteadores, *appliances*, etc. Vários protocolos de comunicação surgiram para permitir que remotamente fosse possível gerenciar os equipamentos. Para alterar os parâmetros de configuração de um servidor ou mesmo resolver um problema de rede ocasionado por um *switch*, passou a não ser mais necessário estar próximo ao equipamento. Para que isso aconteça basta que o equipamento tenha suporte a um dos protocolos de acesso remoto e que exista conectividade entre o equipamento e a estação que irá realizar o gerenciamento.

É possível observar várias vantagens com a utilização desses protocolos. Primeiramente a questão da mobilidade, você pode estar em qualquer lugar e ainda assim pode administrar seus equipamentos, resolver problemas e obter informações em tempo real. Outro ponto é a questão da flexibilidade, a maioria desses protocolos possui clientes que funcionam até mesmo em sistemas operacionais diferentes do qual está sendo administrado, por exemplo: é possível em uma estação de trabalho rodando o sistema operacional Windows, da Microsoft, acessar um servidor Linux.

Por último pode-se citar a questão da agilidade, como é possível ter acesso rapidamente aos equipamentos, o tempo de resposta para resolução de um problema é menor.

Um fator que contribui para a popularização dos protocolos de acesso remoto foi a utilização de *datacenter*. *Datacenters*, ou centro de dados, são estruturas especialmente construídas para abrigar servidores. Em geral essas estruturas apresentam elementos para garantir que os servidores estejam a maior parte do tempo disponível. Elementos como: *links* de internet redundantes, *no-breaks* de grande porte, geradores, refrigeração adequada, proteção contra incêndios, dispositivos de segurança, entre outros. Quando uma empresa passa a hospedar seus servidores em um *datacenter* ela não tem mais acesso físico a ele. Todo o acesso precisa ser feito a distância através de algum protocolo de acesso remoto.

Outra utilidade para esses protocolos é para suporte remoto. Isto é utilizado principalmente no ambiente empresarial para auxiliar os usuários da rede a resolver determinados problemas, que estejam ocorrendo em seus computadores. Nesse cenário temos os usuários comuns de microinformática e os funcionários responsáveis pelo suporte, quando um usuário reclama que está enfrentando algum problema com um computador, um funcionário da equipe de suporte pode acessar remotamente o computador do usuário, ver quais mensagens estão aparecendo na tela e tentar resolver o problema. Nessa situação pode-se perceber bem as vantagens oferecidas pelos protocolos.

Apesar de todas as vantagens e facilidades oferecidas por essas aplicações, elas também introduzem alguns problemas. Principalmente problemas relacionados a segurança da informação. Antes de exemplificar algum desses problemas devesse ter claros os atributos básicos da segurança da informação, que são: confidencialidade, integridade e disponibilidade. Confidencialidade quer dizer que a informação deve estar disponível apenas para as pessoas autorizadas, integridade significa que a informação deve manter as características originais e disponibilidade implica na informação estar disponível no momento em que ela for necessária. Desse modo, qualquer evento que afete um desses três atributos afeta a segurança da informação.

No caso de acesso remoto pode ocorrer eventos que permitam a modificação da informação por pessoas não autorizadas, afetando a integridade. Acesso a informação por pessoas não autorizadas, afetando a confidencialidade. Por fim, podemos pensar que através de um acesso remoto podemos deixar um serviço indisponível, por conseqüência afetando a disponibilidade de algumas informações.

1.1 JUSTIFICATIVA

Os protocolos de acesso remoto são extremamente úteis nas operações diárias. Algumas vezes são fundamentais para a realização de certas atividades, como o acesso a servidores em *datacenter*. Sem eles muitas atividades não poderiam ser realizadas, ou talvez não pudesse ser realizadas com a mesma agilidade. Dessa forma, eles se consolidaram dentro das organizações, já fazem parte dos processos realizados e são uma ferramenta de trabalho indispensáveis em muitas situações.

Todavia, com o surgimento desses protocolos apareceram uma gama de problemas relacionados à segurança. Da mesma forma que o acesso aos equipamentos ficou mais fácil, o roubo das informações também foi facilitado. Antes de esses protocolos surgirem, para que fosse possível o roubo de informação era necessário ter acesso a sala na qual estavam os equipamentos. Agora não existe mais essa barreira física de proteção. Mesmo sem acesso físico a sala é possível ter acesso as informações. Com isso houve uma perda significativa em relação ao nível de segurança dos dados.

Além disso, temos o problema do tráfego dos dados na rede. Alguns protocolos mais simples não implementam nenhum tipo de criptografia dos dados. Com isso todos os dados que trafegam na rede passam de forma legível. Isso significa que todos os dados que o servidor envia para o cliente podem ser capturados e interpretados por um usuário mal intencionado. Por exemplo: caso um administrador de rede esteja acessando um roteador que possua um protocolo sem nenhuma criptografia, um atacante que esteja capturando o tráfego de rede pode

capturar o usuário e a senha informados no momento do acesso. No cenário anterior o atacante de posse dessas senhas passa a poder acessar o roteador.

Mesmo as pessoas que são previamente autorizadas a realizar os acessos são motivo de preocupação. Quando um grupo de pessoas tem acesso a um servidor, é mais difícil determinar quem realizou uma alteração indevida de dados ou quem desligou o equipamento em um momento inoportuno. Em situações como estas fica complicado manter a rastreabilidade das mudanças. Essa falta de controle sobre os acessos acaba contribuindo para a insegurança das conexões. Sem poder saber quem acessou um equipamento ou quem está conectado no momento não é possível realizar auditorias, não é possível garantir que todos os acessos sejam lícitos e devidos.

Toda essa preocupação com a segurança não é infundada. Segundo dados do CERT.br (Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil), os incidentes de segurança nos últimos anos vêm aumentando significativamente, do ano de 2008 foram reportados mais de 222 mil incidentes, já em 2009 foram 358 mil incidentes, um aumento de 60% de um ano para o outro (CERT, 2011). Outro número que aumentou, além do número de incidentes reportados, foi o de prejuízo causado: um estudo do Ponemon Institute (2010) com 45 organizações estadunidenses de médio e grande porte mostrou que o cibercrime custou 3,8 milhões de dólares por ano para cada uma, com uma média de um ataque bem sucedido por semana. Apenas com fraude, que é um dos tipos de incidentes de segurança, uma pesquisa realizada pela consultoria KPMG (2010) com mil empresas brasileiras revelou que 267 bilhões de dólares são gastos por ano. 90% das organizações reconhecem que a fraude é um problema no ambiente corporativo e 60% acreditam que ela tende a aumentar nos próximos dois anos.

Outro ponto preocupante são os dados de ataques contra as portas utilizadas pelos serviços de acesso remoto. Segundo dados produzidos pelo Brazilian Honeypots Alliance (2011) elas figuram entre as portas mais atacadas. O relatório de ataques do mês de janeiro de 2011 mostra que a porta 22 recebeu em média cinco mil ataques pode dia, a porta 23 recebeu dois mil ataques por dia, já porta 5900 recebeu 500 ataques por dia. Conforme os dados mostrados, pode-se perceber que essas portas estão entre as mais visadas pelos atacantes.

A quantidade de ataques contra as portas utilizadas para acesso remoto pode ser explicada em parte pelo número de ferramentas disponíveis para realizar esses ataques. São as chamadas ferramentas de ataques de força bruta, essas ferramentas testam em um curto intervalo de tempo um grande número de usuários e senhas. Dessa forma, é possível conectar em servidores que estejam utilizando senhas conhecidas ou senhas não muito bem elaboradas, como senhas curtas ou utilizando número sequenciais (RECKS; LOVE; TERPSTRA, 2004).

Através dos dados acima se pode perceber que está ocorrendo um aumento dos incidentes de segurança, além do prejuízo causado por eles. Pode-se perceber também que entre os principais alvos estão os protocolos de acesso remoto. Dessa forma, a necessidade de proteção e controle desses acessos se torna clara. Verificamos que existe a necessidade de uma ferramenta que permita auditar os acessos e que, além disso, aumente a segurança dessas conexões. Uma ferramenta que possibilite uma comunicação segura entre os clientes e o servidor, criptografando os dados. Uma ferramenta que impossibilite ou minimize as chances de ocorrência de ataques de força bruta.

1.2 OBJETIVOS

O principal objetivo deste trabalho é desenvolver uma solução que permita a centralização do acesso remoto permitindo uma gerência melhor dessas conexões. Além disso, permitir que toda e qualquer comunicação remota se realize mediante a utilização de criptografia com algoritmo publicamente reconhecido como fortes, difíceis de serem quebrados.

A solução desenvolvida deve possuir ainda as seguintes características:

Permitir que todo acesso possa ser auditado, para isso é necessário que seja salvo informações como IP (*Internet Protocol*) da origem da conexão, porta de origem, IP de destino, porta de destino, usuário utilizado para a conexão, data e horário do inicio da conexão e data e horário da desconexão.

Possuir uma camada adicional de segurança. Essa camada deve implementar um mecanismo de identificação, autenticação e autorização dos

acessos. Preferencialmente essa camada deve dar suporte à autenticação em uma base LDAP (*Lightweight Directory Access Protocol*).

Possibilitar que sejam realizadas consultas dos acessos, consultas como: últimos acessos realizados, quais usuários estavam conectados em um equipamento em uma determinada data, quais usuários estão conectados atualmente, data da última sessão de um usuário, data da última sessão em um equipamento e usuários conectados na solução de acesso remoto. Todas as consultas devem trazer quando possível a informação de qual era o IP de origem da conexão e porta de origem.

Oferecer suporte aos principais protocolos de acesso remoto. Para que se tenha uma boa aceitação é necessário que a ferramenta permita que sejam utilizados os protocolos mais usuais de acesso remoto. Os principais protocolos de acesso remoto são: SSH (*Secure Shell*), Telnet, RFB (*Remote Framebuffer*), RDP (*Remote Desktop Protocol*).

Fornecer um cliente que funcione em plataformas heterogêneas (sistema operacional combinado com hardware). Isso é necessário para que os usuários possam utilizar a ferramenta independentemente da plataforma que eles possuam, independente do hardware ou do sistema operacional. É desejável que usuários que utilizem Linux, Windows ou Mac OS possam utilizar a ferramenta sem maiores transtornos.

Fornecer um cliente que funcione por linha de comando, sem necessidade de estar em ambiente gráfico. Existe a necessidade de existir um cliente que funcione por linha de comando para atender situações como: Usuários que não utilizam o ambiente gráfico, utilização através de equipamento que possua poucos recursos de hardware, quando se faz necessário a partir de um servidor sem interface gráfica acessar outra maquina.

Fornecer recursos que permitam a troca de arquivos nos protocolos que forneçam essa funcionalidade. Alguns protocolos disponibilizam a funcionalidade de troca de arquivos entre os clientes e o servidor, assim sendo, é necessário que a solução proposta tenha suporte a essa funcionalidade.

Impossibilitar acesso aos equipamentos que não sejam realizados pela solução proposta. Para garantir um controle efetivo dos acessos todo o acesso

remoto que seja realizado deve utilizar a solução proposta, outras formas de acesso não podem ser passíveis de uso.

Garantir a confidencialidade da comunicação entre os clientes e o servidor. Os dados trocados pelo cliente e o servidor devem ser mantidos em sigilo, para tal é necessário que a solução implemente mecanismos de criptografia, publicamente conhecidos, que garantam que essa comunicação esteja segura e não possa ser entendida por pessoas não autorizadas.

Impossibilitar ou tornar mais difícil a realização de ataques de força bruta. A solução deve funcionar como uma camada adicional de segurança dos equipamentos, impossibilitando ou tornando mais difícil a realização de ataques de força bruta, é desejável que as tentativas de ataques não cheguem até o equipamento de destino, sendo bloqueadas antes disso.

1.3 FERRAMENTAS RELACIONADAS

Embora existam diversas ferramentas de controle de acesso é difícil encontrar uma que atenda a todos os requisitos, que funcione nos diferentes cenários existentes. Boa parte das ferramentas foca em uma atuação apenas no *host* e não na rede como um todo. O problema dessa abordagem no *host* é que o *host* em geral acaba ficando suscetível a ataques de força bruta. Outro problema da abordagem em *host* é a falta de centralização e gerenciamento das conexões. Para se ter um gerenciamento melhor, o ideal é que existe um ponto central que coordene os demais pontos.

Pela descrição das características necessárias pode-se pensar que um *firewall* poderia atender ao que se pede. Embora seja verdade que uma solução com *firewall* possa resolver muitos problemas, como ataques de força bruta e auditoria da origem dos acessos, ela ainda deixe de atender alguns aspectos. Aspectos como a questão de criptografia dos dados, não é possível utilizando apenas um *firewall* prover esse tipo de serviço. Outro aspecto é a questão da granularidade dos acessos, não é possível no *firewall* criar políticas voltadas para usuários e equipamentos, o máximo que se pode fazer é criar políticas baseadas em origem e

destino dos acessos, sem poder criar regras especificas para determinados usuários (TANENBAUM, 2003).

A solução mais próxima de atender aos requisitos é a solução de VPN (*Virtual Private Network*). A VPN, ou Rede Privada Virtual, é um túnel de criptografia entre pontos autorizados, criados através de uma rede, que pode ser pública ou privada, para transferência de informações, de modo seguro, entre redes corporativas ou usuários remotos. É a solução mais próxima de atender a todos os requisitos necessários.

Até o final dos anos 90 redes de computadores eram conectadas através de linhas alugadas ou de conexões discada. O valor desses links eram altos. Nesse cenário surgiram as VPNs, ou redes privadas virtuais, que permitiram que as empresas reduzissem custos de conexão, além de prover a troca de dados confidenciais de maneira segura (REDE NACIONAL DE PESQUISA, 2011).

O funcionamento de uma VPN é fácil de ser entendida, basicamente ela encapsula os dados que estão sendo enviados, em um túnel criptográfico. Com isso, os dados podem estar trafegando, mesmo em uma rede pública, como a internet, que ainda assim estarão seguros. Apenas a origem e o destino possuem as chaves criptográficas necessárias para decodificar o tráfego que foi enviado.

Para acessar o túnel VPN é necessário estar autenticado. A autenticação pode ser através de vários mecanismos como: senhas, biometria ou mesmo usando *tokens*. Com isso é possível realizar a identificação, autenticação e autorização dos acessos. Somente as pessoas devidamente autorizadas podem acessar a rede VPN e os equipamentos que lá estão.

É possível utilizar várias soluções de criptografias em túneis VPN. Entre os protocolos suportados então: IPSec (*Internet Protocol Security*) , SSL/TLS (*Transport Layer Security/Secure Sockets Layer*), DTLS (*Datagram Transport Layer Security*),MPPE (*Microsoft Point-to-Point Encryption*) e SSTP (*Secure Socket Tunneling Protocol*) (SILVA, 2005).

Aproveitando todas essas características que a VPN possui, poderíamos construir uma arquitetura de rede na qual seria possível utilizar essas características para prover um pouco mais de segurança nos acessos remotos. Para isso, seria necessário colocar todos os equipamentos que serão acessados remotamente, em

uma rede. O acesso a essa rede seria possível apenas utilizando um túnel VPN. Todos os clientes que desejassem acessar um servidor teriam necessariamente que autenticar na VPN para então ter acesso a rede, após isso poderia acessar o equipamento desejado. Nesse cenário teríamos criptografia dos dados, autenticação dos usuários e também evitaríamos ataques de força bruta, pois as portas de acesso remoto não estariam abertas para toda a rede, estariam seguras atrás da VPN.

Contudo, alguns requisitos não podem ser atendidos pela solução de VPN. Um deles é a questão da auditoria dos acessos aos equipamentos, com a VPN apenas seria possível saber que determinado usuário esta conectado na rede, mas não seria possível saber qual equipamento ele esta acessando em um exato momento. Outro ponto negativo é que utilizando VPN, seria necessário, antes de realizar um acesso remoto, estabelecer o túnel VPN, isso faz com que o processo não seja tão ágil.

DESENVOLVIMENTO

Diante dos problemas de segurança inerente aos protocolos de acesso remoto, e da falta de ferramentas apropriadas que solucionem os problemas por completo, verificamos a necessidade do desenvolvimento de uma nova solução que enderecem todos esses problemas. Essa solução tem por objetivos resolver aos problemas ocasionados pela atividade de acesso remoto, permitindo que essa atividade possa continuar existindo, porém com um grau de segurança e gerenciabilidade melhor.

1.1 FERRAMENTAS UTILIZADAS

Para atender a todos os requisitos de segurança, foi necessário selecionar ferramentas heterogêneas. Algumas ferramentas selecionadas, embora tenham funcionalidades semelhantes são complementares e não concorrentes. Foram retiradas as melhores características de cada uma delas para que fosse possível atingir a todos os objetivos do projeto.

1.1.1 Debian

O Debian é um sistema operacional composto por diversos softwares livres ou de código aberto. É baseado no *kernel* do Linux e utilizam diversas ferramentas do projeto GNU (GNU *is Not Unix*), como o compilador GCC (GNU *Compiler Collection*). Seu nome oficial é Debian GNU/Linux. Foi criado em 1993 por Ian Murdock, em seus tempos de universitário. O nome Debian é a junção do nome da namorada de Ian, Debra Lynn, com o seu primeiro nome. Existem iniciativas de fazer o Debian funcionar com outros Kernels, além do Linux, mas essas iniciativas ainda são incipientes (DEBIAN, 2011).

Entre as principais características do projeto, pode-se destacar a aderência a filosofia do software livre e o foco em segurança e estabilidade. Devido a esse foco em estabilidade e segurança, ele se tornou base para outras distribuições Linux,

além de ser largamente utilizado em servidores de e-mail, banco de dados, web, DNS (*Domain Name System*), entre outros. Outra característica que contribuiu para o sucesso do Debian foi o processo colaborativo de desenvolvimento e teste, através desse processo foi possível manter a qualidade do sistema operacional e atender a demanda para a inclusão de novos softwares.

Outro ponto marcante do Debian é o seu sistema de instalação e remoção de novos softwares, o APT (*Advanced Packaging Tool*). Através do APT é possível instalar e remover software apenas utilizando algumas linhas de comando, é possível atualizar todo o sistema operacional ou até mesmo migrar para uma nova versão. Além da interface de linha de comando é possível adicionar novos softwares utilizando a interface gráfica. Atualmente existem mais de 29 mil softwares disponíveis para instalação, nos repositórios utilizados pelo APT. Caso um software não esteja no repositório o usuário pode solicitar que seja adicionado ou o próprio usuário pode criar um pacote no formato aceito pelo APT, processo conhecido como empacotamento de software (SOUZA, 2010), (FILHO, 2006).

Segurança também é uma das preocupações do Debian. Como todo o sistema é baseado em software livre, qualquer pessoa pode verificar como funciona o código e validar todos os métodos de segurança utilizados. Esse modelo é o oposto do utilizado por softwares fechados, no qual se utiliza o conceito de segurança por obscuridade, não sendo possível validar se os métodos de segurança estão aplicados de maneira correta. O Debian também possui um time de auditoria e segurança, que revisa todos os códigos procurando *bugs* de segurança. Assim que essas vulnerabilidades são encontradas os responsáveis pelos softwares são informados para que providenciem a correção (MORIMOTO, 2006).

1.1.2 Java

Em 1991 a Sun Microsystem financiou uma pesquisa interna de codinome *Green*. O projeto resultou no desenvolvimento de uma linguagem baseada no C e C++. O primeiro nome da linguagem era *Oak*, significa carvalho em inglês. Mais tarde descobriu-se que já existia uma linguagem com esse nome. Os criadores da linguagem mudaram o nome então para Java. O nome é devido a cidade de origem

de um café, que era servido na cafeteria local. O projeto *Green* tinha como objetivo inicial desenvolver uma tecnologia para TV interativa, permitindo que o usuário pudesse interagir com a programação e não apenas receber as informações de maneira passiva. Entretanto, isso se mostrou muito avançado para indústria televisa da época. Java então acabou preenchendo outro nicho de mercado, o de aplicações para a internet, que estava nascendo naquele período e estava carente de tecnologia (DEITEL; DEITEL; HARVEY, 2002).

Um dos grandes diferenciais do Java em relação às outras linguagens é a portabilidade. A Sun chamou essa característica de "*Write Once, Run Anywhere*", ou em uma tradução livre: "escreva uma vez e rode em qualquer lugar". Isso permite que um programa que foi escrito em Java possa rodar em diferentes sistemas operacionais, sem que seja necessário reescrever nenhuma linha de código ou fazer qualquer alteração. Isso permitiu as empresas de desenvolvimento de software diminuir os custos de migração entre plataforma. Além de permitir que detalhes específicos de cada sistema operacional fossem abstraídos para os desenvolvedores, que não precisavam mais saber informações minuciosas sobre a plataforma na qual o programa irá rodar. Tudo isso possível graças a JVM (*Java Virtual Machine*), ou maquina virtual Java, ela atua como uma camada entre o programa desenvolvido e o sistema operacional (JAVA, 2011).

A JVM além de ser responsável pela abstração do sistema operacional é responsável por outras características da linguagem. Um dos itens de responsabilidade da JVM é a limpeza da memória. Programas escritos em algumas linguagens como C tem um grande problema relacionado com o gerenciamento de memória, sendo necessária que o programador faça à alocação de memória necessária e desaloque a memória quando essa não for mais necessária. Em Java o programador não tem essa preocupação, a própria maquina virtual realiza essa tarefa de limpeza da memória, através do *garbage collector*, removendo as instancias que não estão mais em uso. Essa característica garante que os programas em Java não sofram com problemas de alocação de memória. Isso proporciona aos programadores Java uma maior produtividade, uma vez que não é necessário ficar escrevendo código para fazer o gerenciamento da memória utilizada (DEITEL; DEITEL; HARVEY, 2002).

Uma característica importante do Java é o seu licenciamento, a Sun licenciou a maior parte do código como software livre, sobre os termos da GNU GPL (GNU *General Public License*). Devido a isto foi possível o desenvolvimento de diversas ferramentas e bibliotecas para a linguagem. Várias empresas patrocinaram o desenvolvimento de ferramentas para tornar a linguagem mais produtiva e eficiente. Inúmeras comunidades surgiram para apoiar os programadores iniciantes e produzir documentação sobre a linguagem e suas APIs. Com a união dessas comunidades com as empresas foi possível o desenvolvimento do JCP (*Java Community Proces*). O JCP é o processo formal que permite as pessoas e empresas influenciarem nas versões futuras do Java, definindo quais recursos devem ser adicionados ou removidos da plataforma (JAVA, 2011).

1.1.3 Perl

Perl é uma linguagem de programação criada no ano de 1987, por Larry Wall, enquanto ele trabalhava como programador na Unisys. O objetivo de Larry era desenvolver uma linguagem de script para UNIX que se torna fácil a geração de relatórios. A linguagem foi implementada utilizando a linguagem C. Possui implementação para diferentes sistemas operacionais. Hoje Perl é uma linguagem largamente conhecida pelos desenvolvedores, principalmente para aqueles que desenvolvem para Unix e Linux. Atualmente possui diversos recursos que não existiam nas versões iniciais, a última versão estável é a 5.12.3, mas já existe uma versão 6.0 em desenvolvimento. A linguagem é livremente distribuída sobre os termos da licença GNU GPL (PERL, 2011).

A sintaxe do Perl é parecida com a utilizada pela linguagem C. A sua sintaxe também lembra um pouco as linguagens de Shell Script devido ao uso de cifrão para variáveis escalares. Outras características herdados de outras linguagens é o uso de listas e de expressões regulares. As listas foram herdadas da linguagem Lisp. Já as expressões regulares foram herdadas de SED. Outra característica importante da linguagem é o gerenciamento automático de memória e a tipagem dinâmica. Em um programa em Perl não é necessário fazer alocação de memória, o próprio interpretador se encarrega disso, alocando e desalocando a memória quando

necessário. Nos programas em Perl também não é necessário informar o tipo da variável, isso é determinado em tempo de execução pelo interpretador.

Devido a essa versatilidade, Perl é bem utilizada pelos administradores de rede. Com apenas algumas linhas de código é possível fazer *scripts* em Perl para monitorar o comportamento de uma rede ou *scripts* para analisar os arquivos de log de um servidor web. Porém o uso de Perl não se restringe a administração de sistemas, diversos sistemas web são escritos em Perl, utilizando CGI (*Common Gateway Interface*). Aplicações para bioinformática também fazem uso de Perl, principalmente quando é necessário tratar grandes cadeias de caracteres. Em geral para problemas nos quais é necessário trabalhar com expressões regulares, Perl é altamente recomendado. A linguagem oferece flexibilidade para essas operações.

Um recurso interessante do Perl é o CPAN (*Comprehensive Perl Archive Network*). O CPAN é um arquivo que possui mais de 20 mil módulos de software escritos em Perl, bem como a documentação de cada módulo. Esses módulos são bibliotecas externas que podem ser utilizados pelo programador. O Perl também possui uma interface por linha de comando que atua como um gerenciador de pacotes. Com essa interface é possível adicionar e remover esses módulos. O CPAN é uma importante ferramenta para o programador Perl, tornando o desenvolvimento de programas em Perl mais produtivos. Com a utilização dos módulos CPAN o desenvolvimento se torna mais rápido, basta utilizar os módulos prontos para a função que se deseja. Caso não exista um determinado módulo o programador pode desenvolvê-lo e caso deseje torná-lo publico basta enviar para o repositório, assim demais programadores poderão utilizá-lo e melhorá-lo (MCPHIE; DEITEL; NIETO, 2002).

1.1.4 JSP

JSP ou *Java Server Pages* é uma tecnologia Java que ajuda os desenvolvedores a criar páginas web de conteúdo dinâmico. O JSP foi lançado em 1999 pela Sun para oferecer um suporte melhor para Web. Possui similaridade com outras linguagens de programação para web como ASP (*Active Server Pages*) e PHP (PHP: *Hypertext Preprocessor*). Em alguns casos chega a concorrer com essas

linguagens por possuir funcionalidades semelhantes. Por fazer parte da tecnologia Java possui a vantagem de ser multiplataforma, uma página escrita em linguagem JSP pode funcionar em diferentes sistemas operacionais (JSP, 2011).

Utiliza tags parecidas com XML (*Extensible Markup Language*). Essas tags encapsulam a lógica necessária para gerar o conteúdo das páginas. O servidor de aplicações Java lê essas *tags* e gera o conteúdo HTML, que por sua vez é enviado ao cliente que acessou a página JSP. Existem diversos servidores de aplicações compatíveis com JSP. Um dos servidores mais populares é o Tomcat, desenvolvido pelo grupo apache.

Dentro das páginas JSP é possível escrever código Java puro. A escrita desses códigos dentro de uma página JSP é conhecida como *scriptlet*. Com esses blocos de código é possível fazer qualquer coisa que seria possível em um programa Java convencional, como acessar banco de dados, ler arquivos, entre outras coisas. Embora exista essa opção de uso de *scriptlet*, ela não é considerada uma boa prática, o ideal seria a utilização de Taglibs. Taglibs, ou biblioteca de *tags*, em uma tradução livre, são extensões que são criadas para geração automática de código. O uso de taglibs é recomendado por deixar o código na página JSP mais "limpo", sendo mais fácil de ser entendido (SIERRA; BATES; BRIAN, 2005).

Diversas aplicações web utilizam JSP. Entre essas aplicações pode-se citar as de *internet bank*. Isso pode ser explicado em partes pela questão da segurança. Segurança é um dos pontos fortes da plataforma Java, dessa forma aplicações que necessitam de um grau maior de segurança acabam optando pelo uso de JSP e de outros recursos da tecnologia Java.

1.1.5 C

C é uma linguagem de programação procedural, estruturada e compilada. Foi desenvolvida entre 1969 e 1973 por Dennis Ritchie nos laboratórios da AT&T. O objetivo de seus criados era criar uma linguagem para poder desenvolver o sistema operacional Unix, até então o comum era que os sistemas operacionais fossem escritos na linguagem assembly. Atualmente é uma das linguagens de programação mais populares de todos os tempos. Possui compiladores para a maioria das

arquiteturas. Além disso, influenciou muitas outras linguagens de programação, que passaram a utilizar os seus conceitos e sintaxe (C PROGRAMMING, 2011).

Na linguagem C todo código executável está contido dentro de função. A passagem de valores para as funções é realizada via parâmetros. É possível também a passagem de valores por referência, nesse caso é necessário utilizar ponteiros. Ponteiros são estruturas que armazenam um endereço de memória, permitindo um acesso de baixo nível a memória do computador. A linguagem também permite a criação de estruturas com tipos heterogêneos, as chamadas "struct", possibilitando a manipulação desses tipos como uma única unidade. Funções avançadas como operações de *input* e *output* e manipulação de *string* são delegadas para bibliotecas do sistema.

Nos seus primeiros anos de vida a linguagem C passou a ser implementada por uma grande variedade de mainframes e microcomputadores, com isso sua popularidade aumentou significativamente. Como consequência no ano de 1983 a ANSI (*American National Standards Institute*) formou um comitê para estabelecer padrões para o C. Esses padrões ficaram prontos no ano de 1989. Essa versão padronizada ficou conhecida como ANSI C, algumas vezes também é referenciada como C89. Mais tarde passou a ser adotado também pela ISO (*International Organization for Standardization*) como ISO/IEC 9899:1990, ou apenas C90. Após isso no ano de 1999 foi lançada uma versão revisada do padrão C, conhecido como C99. Embora essa nova versão tenha sido aprovada, alguns compiladores não suportam todas as características descritas no padrão. Como exemplos de compiladores que não seguem o padrão estão os compiladores da Microsoft e da Borland (SCHILDT, 1996).

Pode-se utilizar a linguagem C em diversos cenários. Um dos cenários possíveis é para programação de sistemas operacionais e sistemas embarcados. Essa preferência pela linguagem C nesse cenário pode ser explicada por características como: portabilidade do código, eficiência, habilidade de acessar endereços específicos do hardware e baixa demanda por recursos do sistema. A linguagem também pode ser utilizada para programação para web através de CGI. Outra possibilidade de uso é para implementação de compiladores, bibliotecas e

interpretadores de outras linguagens. Com todos esses diferentes cenários é possível ver a flexibilidade que a linguagem oferece para o programador.

1.1.6 OpenSSH

O OpenSSH, ou OpenBSD Secure Shell, é um conjunto de programas que proveem comunicação criptografada em uma rede de computadores. Foi criado pelo projeto OpenBSD para ser uma alternativa livre para o SSH original. É distribuído livremente sobre os termos da licença BSD, possibilitando, por exemplo, que seja utilizado por software que possuem o código fechado (OPENSSH, 2011).

Possui uma suíte de ferramentas, entre essas ferramentas pode-se citar: ssh, scp, sshd, ssh-keygen, ssh-keyscan. O SSH possibilita acesso remoto as maquinas. O scp possibilita a troca segura de arquivos entre os computadores. Sshd é o servidor de SSH propriamente dito. Ssh-keygen é uma ferramenta que inspeciona e gera as chaves RSA e DSA (*Digital Signature Algorithm*) utilizadas na autenticação. Já o ssh-keyscan realiza um *scan* em uma lista de *hosts* para coletar suas chaves públicas (BARRETT; SILVERMAN; BYRNES, 2005).

O OpenSSH possui diversas funcionalidades além do acesso remoto. Umas das funcionalidades oferecidas é o *forward* remoto de portas TCP, possibilitando que sejam encaminhados dados através de um canal seguro. Outra funcionalidade também interessante é a opção de VPN, é possível criar um túnel VPN para conectar a uma rede de maneira segura. É possível também criar um servidor *proxy* de maneira *ad hoc* utilizando o conceito de *SOCKS*.

Devido a sua licença de uso não ser restritiva ele é utilizado por diversas aplicações. A lista de usuários é extensa, indo desde sistemas operacionais para microcomputadores até fabricante de roteadores. Entre os usuários mais notáveis estão fabricantes como cisco, que utilizam o OpenSSH para fornecer acesso remoto a seus equipamentos. Na linha de sistemas operacionais então o Mac OS e a família BSD: FreeBSD, OpenBSD e NetBSD.

1.1.7 Postgres

Postgres, ou PostgreSQL, é um gerenciador de banco de dados objeto relacional. Mantido por uma comunidade de usuários ao redor do mundo. É licenciado como software livre e como tal possui seu código aberto. O Postgres surgiu como uma continuação do projeto Ingres, mantido pela universidade de Berkeley. O projeto Ingres visava criar um banco de dados relacional. Para que pudesse ser utilizado além do ambiente acadêmico o Postgres teve que adicionar diversos recursos que não existiam no Ingres. Foram adicionados conceitos como MVCC (*Multiversion Concurrency Control*), melhorias no suporte ao SQL (*Structured Query Language*) e adição de novos tipos de dados nativos (POSTGRESQL, 2011), (GONZAGA, 2007).

Entre as funcionalidades oferecidas pelo banco de dados está o suporte a linguagens procedurais. Essas linguagens permitem que blocos de códigos sejam armazenados e executados no banco de dados. É possível criar esses blocos de códigos utilizando Tcl, Perl, Python, entre outras linguagens.

O controle de concorrência do Postgres é controlado por um sistema chamado MVCC. Esse sistema funciona fornecendo para cada usuário do banco um "snapshot" do banco, uma espécie de retrato fiel do banco naquele exato momento, permitindo assim que as mudanças realizadas pelo usuário não afetem os demais usuários até que a transação seja finalizada. Essa funcionalidade garante que o banco de dados possa manter as propriedades ACID - atomicidade, consistência, isolamento e durabilidade (MANZANO, 2008).

Entre os usuários mais notáveis do banco de dados Postgres pode-se citar o Yahoo e o Skype. O Yahoo mantém uma base de análise de comportamento dos usuários web, armazenando mais de dois petabytes de dados. Já o Skype utiliza o banco para gravar os dados de sua central de negócios.

1.1.8 Netbeans

O NetBeans é uma IDE (*Integrated Development Environment*) para desenvolvimento de programas nas linguagens Java, Java Script, PHP, Python, Ruby, Groovy, C, C++, Scala, Clojure entre outras. Foi escrito em Java e pode funcionar em qualquer ambiente que tenha uma JVM (Java *Virtual Machine*) instalada. No caso de utilização para desenvolvimento de aplicações Java é necessário também ter uma JDK (Java *Development Kit*) instalada. Oferece suporte para desenvolvimento tanto de aplicações para *desktop* quanto aplicações para web e dispositivos móveis. O seu desenvolvimento teve inicio em 1996 por dois estudantes universitários, logo após foi adquirido pela Sun Microsystems (NETBEANS, 2011), (GONÇALVES, 2006).

Possui um grande diferencial em relação a outras IDEs quando a questão é desenvolvimento de aplicativos para *Desktop*. Permite que apenas "arrastando" os componentes visuais, o programador possa construir uma tela com formulários e botões. Todo o código de construção das telas é gerado automaticamente pela IDE. Isso torna o desenvolvimento das aplicações mais rápido, economizando tempo e aumentando significativamente a produtividade. Outro beneficio é a manutenção do código. Caso seja necessário fazer qualquer alteração na parte visual, basta realizar as modificações que a própria IDE se ocupa em fazer as modificações no código.

Além de facilitar o desenvolvimento de aplicações para *Desktop*, também auxilia no desenvolvimento das demais aplicações. Por ser um ambiente integrado de desenvolvimento, oferece diversas funcionalidades para o desenvolvedor, como opções para compilar e *debugar* código. Possui ainda uma gama imensa de *plugins* que podem ser instalados na IDE. Alguns desses *plugins* oferecem funcionalidades como geração de diagramas UML (*Unified Modeling Language*), integração com CVS (*Concurrent Versions System*), entre outras facilidades.

Para desenvolvimento web oferece alguns recursos como integração com o servidor de aplicações. Através dessa integração é possível de dentro da IDE instalar uma nova aplicação no servidor e fazer operações como iniciar e parar. Além desse recurso, oferece editores de código Java Script e CSS. Possui também

"*wizards*", que auxiliam na criação de uma nova aplicação, criando toda a estrutura de diretórios e arquivos necessários.

1.2 PROTOCOLOS SUPORTADOS

Para atender o maior número possível de clientes serão suportados os quatro principais protocolos de acesso remoto, são eles: SSH (*Secure Shell*), Telnet, RFB (R*emote Framebuffer*), RDP (*Remote Desktop Protocol*). O SSH é o principal protocolo remoto utilizado pelo Linux, pelos sistemas operacionais derivados do Unix e pelos novos equipamentos de rede. O Telnet é utilizado por sistemas legados e por sistemas que possuem escassez de recursos. Já o RFB é utilizado para acesso a interfaces gráficas tanto do Windows quanto em outros sistemas. Por fim o RDP, que é utilizado por padrão pelo conjunto de sistemas operacionais Windows, superiores ao Windows XP.

1.2.1 Telnet

É um protocolo de rede utilizado para prover uma comunicação bidirecional. Toda a comunicação é realizada utilizando por padrão a porta 23 do protocolo TCP (*Transmission Control Protocol*). Para que essa comunicação funcione é necessária a existência de um servidor Telnet e de um cliente. Em geral, após realizada a conexão é fornecido ao cliente o acesso a uma CLI (*Command Line Interface*). Com o acesso a essa CLI o cliente pode realizar diversas operações, até mesmo configurar remotamente o *host* que está sendo acessado (RFC 854, 2011).

O Telnet é um dos protocolos de acesso remoto mais antigo que ainda se encontra em uso. Devido a sua simplicidade, ele foi largamente implementado por fabricantes de equipamentos de rede e de sistemas operacionais. Os fabricantes disponibilizavam acesso remoto através do Telnet para que os usuários pudessem reconfigurar os equipamentos e realizar outras operações. Ainda hoje é possível encontrar alguns equipamentos de rede que continuam fazendo uso desse

protocolo. Além dos equipamentos novos existe todo um parque de equipamento legados que ainda utilizam o protocolo.

Na época em que o Telnet foi desenvolvido, a maioria dos usuários de computadores estava em instituições de ensino ou em departamentos de pesquisa do governo. Nesses ambientes, segurança não era uma preocupação como é nos dias de hoje. O número de usuários era menor e o ambiente era mais controlado, assim as aplicações não eram desenvolvidas tendo segurança como uma prioridade. Devido a isso o Telnet não possui recursos de segurança como criptografia dos dados e autenticação dos *hosts*.

Embora tenha suas franquezas em relação a segurança, o protocolo continua sendo uma boa opção para alguns cenários de uso. Como exemplo de cenários em que pode ser recomendado o uso, pode-se citar os ambientes que tenham um acesso controlado, nesse caso as deficiências do Telnet não seriam problemas. Outra situação em que o protocolo pode ser recomendado é para equipamentos que possuam poucos recursos de hardware, como o protocolo é simples não necessita que os equipamentos disponham de grande processamento.

1.2.2 SSH

Protocolo de rede que permite a troca segura de dados, utilizando para isso um canal seguro entre dois dispositivos de redes. Faz uso do protocolo TCP e por padrão utiliza a porta 22. Utilizado primariamente no Linux e em sistemas baseados no Unix para permitir acesso ao *Shell* dos servidores. Foi criado para ser um substituto do Telnet e de outros protocolos inseguros de acesso remoto. Para isso passou a utilizar criptografia dos dados, provendo dessa forma confidencialidade e integridade dos dados mesmo que eles estejam trafegando em uma rede insegura como a internet (RFC 4251, 2011).

Além da criptografia dos dados o SSH também possibilita a autenticação dos *hosts*. Para que a autenticação dos *hosts* funcione é utilizado o conceito de chave publica, cada *host* possui uma chave publica que pode ser consultada por qualquer outro *host* que deseje iniciar uma comunicação. Quando o cliente recebe a chave

publica do *host* ele pode confirmar se aquela chave realmente é a chave utilizada por aquele *host* ou se existe uma outra maquina tentando se passar pelo *host*.

O protocolo possui duas versões, conhecidas como SSH1 e SSH2. A versão SSH2 é uma versão melhorada da versão anterior, possui novas funcionalidades que garantem uma segurança e integridade melhor dos dados. A segurança foi fortalecida através da troca de chaves Diffie–Hellman e também pela checagem de integridade utilizando algoritmos de MAC (*Message Authentication Code*) (STALLINGS, 2008). Além dessas novas funcionalidades, foi adicionado o suporte a múltiplas sessões *Shells*, através de uma única conexão SSH. Devido a essas novas características, as duas versões são incompatíveis.

As possibilidades de uso do SSH são variadas, indo desde sistemas operacionais até roteadores. Na linha de sistemas operacionais pode-se citar o MAC OS X, a família BSD e diversas distribuições Linux, todos esses utilizam o SSH como protocolo padrão para acesso remoto. Mesmo no ambiente Windows é possível utilizar o SSH, abrindo uma sessão para o CLI (*Command Line Interface*) do Windows. Já na linha de equipamentos é possível citar os equipamentos produzidos pela CISCO, além de outros *appliance* produzidos por fabricantes de equipamentos de rede.

1.2.3 RFB

O *Remote Framebuffer*, abreviado como RFB, é um protocolo que fornece acesso remoto a interface gráfica do usuário. Devido ao seu modo de funcionamento ele pode ser utilizado em vários sistemas de gerenciamento de interfaces, incluindo X11, Windows e Mac OS. Como exemplo de aplicação que utiliza o protocolo RFB pode-se citar o VNC (*Virtual Network Computing*) e seus derivados. O RFB é chamado de protocolo simples por requerer poucos recursos de hardware do cliente. Ao realizar a conexão o cliente consegue visualizar exatamente o que está sendo mostrado na tela do usuário. As modificações na tela do usuário também são refletidas para o usuário, assim ambos vêem exatamente a mesma imagem. Essa característica é extremamente útil para operações como suporte remoto a usuários (REAL VNC, 2011).

O protocolo foi criado pela Olivetti *Research Laboratory* (ORL). O objetivo inicial era criar uma tecnologia de acesso remoto capaz de ser executada em equipamento simples, chamados de *thin clients*. Os *thin clients* são equipamento que não possuem grandes recursos de hardware e que dependem de outro computador, geralmente um servidor, para executar todas as suas tarefas.

As marcas VNC e RFB são propriedades registradas da empresa RealVNC. Entretanto, o código original do VNC está disponível para consultas e alterações segundo os termos da GNU GPL. Isso possibilitou o surgimento de variadas aplicações, tanto clientes, quanto servidores, que implementam o protocolo RFB.

Em relação a segurança, não é um protocolo que implemente o melhor modelo possível. As senhas não são enviadas em texto puro, como acontece com o Telnet, mas também não implementam os melhores algoritmos de criptografia possíveis. Um atacante que tenha acesso aos dados que estejam trafegando na rede, pode através de técnicas de tentativa e erro descobrir qual senha está sendo utilizada. Para garantir uma segurança maior sugere-se a utilização de tunelamento através do uso de túneis SSH ou VPN. Esses túneis funcionam como uma camada adicional de segurança, tornando mais difíceis a realização de ataques.

1.2.4 RDP

O RDP (*Remote Desktop Protocol*) é um protocolo proprietário desenvolvido pela Microsoft que prove aos usuários acesso remoto a interface gráfica de outro computador. Por padrão trabalha na porta 3389 TCP. O protocolo é uma extensão de outros protocolos definidos pela ITU (*International Telecommunication Union*). Embora seja um protocolo de propriedade da Microsoft, existem clientes para outros sistemas operacionais, além do Windows. Atualmente está na versão 7.0, a cada nova versão são adicionadas novas funcionalidades. Para manter compatibilidade entre diferentes versões, os clientes e servidores negociam no inicio de uma nova sessão qual versão irão utilizar (MICROSOFT, 2011).

Utiliza o conceito de multicanais, permitindo que canais virtuais separados transportem diferentes tipos de dados. Ao todo existem mais de 64 mil canais. É possível segmentar informações provenientes do teclado, mouse, impressora, etc.,

em cada um dos canais. No entanto esse recurso não foi utilizado na versão inicial do RDP, sendo que todas essas informações eram trafegadas em um único canal.

Entre as principais características do protocolo estão: criptografia, redirecionamento de áudio, redirecionamento de arquivos e área de transferência compartilhada. A criptografia fornecida é de 128 bits, utilizando um algoritmo RC4. O redirecionamento de áudio permite que usuários executem um programa de áudio no servidor remoto e que o som seja redirecionado para o computador cliente que está realizando o acesso. O redirecionamento de arquivos possibilita que arquivos locais sejam utilizados na máquina remota. Já o compartilhamento da área de transferência possibilita que dois computadores, o cliente e o servidor, utilizem a mesma área de transferência.

Atualmente é a ferramenta padrão para realizar acesso remoto a servidores Windows. Possui apenas uma limitação em relação ao número de acessos simultâneos. Caso o administrador deseje aumentar esse limite de conexão é necessário adquirir novas licenças de uso. Algumas instituições utilizam esse recurso como forma de compartilhamento de programas pagos. Nesse caso é instalada em um servidor a ferramenta que se deseja compartilhar, todos os usuários que desejem usar a ferramenta podem acessar remotamente o servidor e fazer uso do software. Com isso a empresa economiza dinheiro, pois não é necessário adquirir varias licenças do software, apenas a licença para instalação no servidor é suficiente.

1.3 ARQUITETURA.

Nesta sessão iremos mostrar a arquitetura de rede utilizada para acesso remoto, enfatizando os problemas ocasionados por ela. Iremos também mostrar uma proposta de arquitetura para resolver esses problemas. Essa arquitetura que está sendo proposta é a que será utilizada no desenvolvimento do SafeFW. Mostraremos os benefícios inerentes ao uso dessa arquitetura. Por fim iremos mostrar uma arquitetura para alta disponibilidade.

1.3.1 Arquitetura atual.

A arquitetura de acesso remoto utilizada na maioria dos ambientes é parecida com a Figura 01. Nessa arquitetura pode-se observar que não existe um ponto central de acesso. Os usuários têm acessos a diferentes servidores em diferentes redes. Pode-se observar também a existência de caminhos diversos para ser realizado o acesso.

Esse modelo causa vários problemas. Primeiro deles é que devido a não ter um ponto central por onde os acessos devem passar, não é possível, ou pelo menos difícil, de implementar mecanismos que permitam logar todos os acessos que estão sendo realizados. Outra questão é que as portas de acesso remoto ficam abertas para toda a rede, logo os equipamentos ficam suscetíveis a ataques, como os de força bruta.

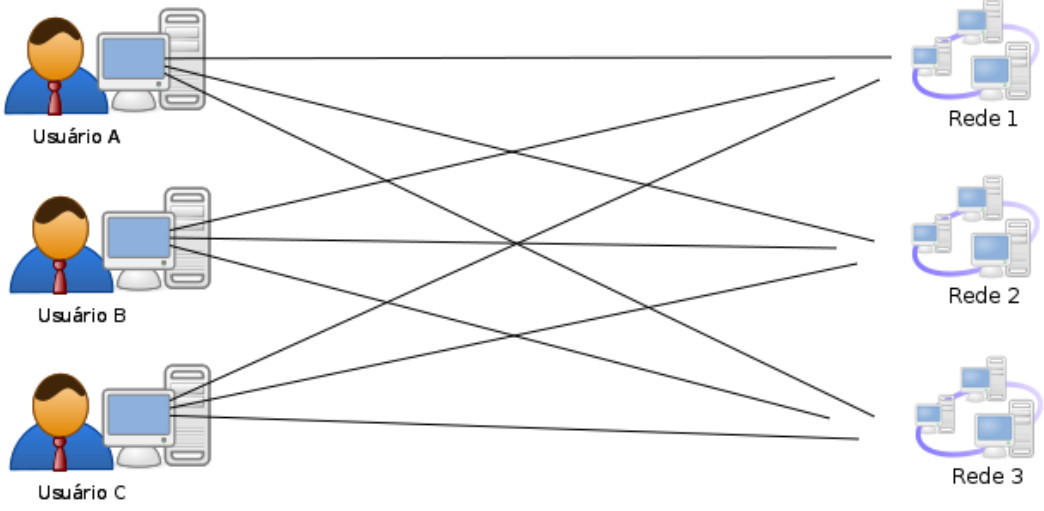


Figura 1: Arquitetura de rede atual.

1.3.2 Arquitetura proposta.

Levando em consideração os problemas ocasionados pela arquitetura utilizada até o momento, verificamos a necessidade de uma nova arquitetura. Essa nova arquitetura deve ser parecida com a Figura 02. Nessa arquitetura todos os acessos têm que passar por um ponto único. Os acessos devem ser identificados e autenticados por um outro elemento, como por exemplo um servidor LDAP. O ponto central de acesso se encarrega de salvar todas as informações relevantes em uma base de dados. Por fim os acessos são direcionados até o equipamento de destino.

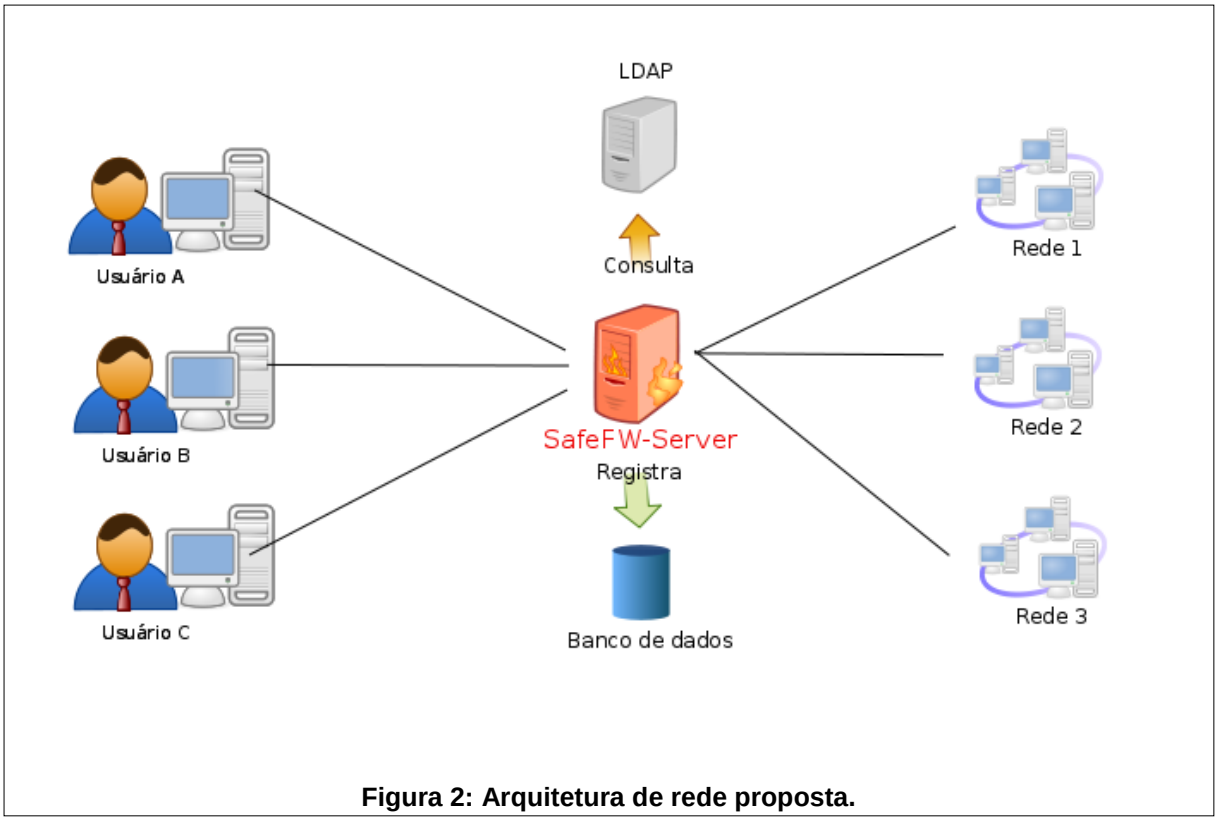


Figura 2: Arquitetura de rede proposta.

Nessa arquitetura não é possível realizar um acesso remoto sem passar pelo ponto central de acesso. Os benefícios desse modelo são:

- Evitar ataques de força bruta. Como todos os acessos somente são realizados pelo servidor central, pode-se fechar todas as portas utilizadas para acesso remoto, liberando acesso apenas para a maquina central. Uma vez que a porta esteja fechada é mais difícil a realização desse tipo de ataques.

- Prover auditoria dos acessos. Com o servidor central é possível gravar todos os acessos que são realizados, inclusive com informações detalhadas como IP de origem e nome do usuário que originou o acesso. Com isso é possível realizar consultas dos acessos realizados.

- Funcionar como uma camada adicional de segurança. Com a consulta a base de dados externa é possível prover: identificação, autenticação e autorização dos acessos. Isso prove uma segurança ainda maior da conexão, pois agora além da senha de acesso ao equipamento o usuário também deve se identificar no ponto central de acesso.

- Independência de plataforma. A comunicação entre os usuários e o ponto central funciona no modelo cliente servidor. Nesse modelo é possível a existência de diferentes clientes, funcionando em diferentes plataformas. Para isso basta que o cliente suporte o mesmo protocolo de comunicação utilizado pelo servidor. Ou seja, o software utilizado pelo cliente passa a ser irrelevante basta que ele tenha suporte ao protocolo utilizado pelo servidor.

- Uso de um canal criptografado. Para a troca de dados entre o ponto único de acesso e os seus clientes pode-se utilizar um canal criptografado. Para isso basta que o protocolo utilizado tenha suporte a criptografia. Isso aumenta significativamente a segurança nas conexões remotas. Pois é possível mesmo utilizado um protocolo inseguro como o Telnet, ter uma comunicação confiável, garantindo requisitos como confidencialidade dos dados. Até mesmo protocolos que já possuem criptografia como SSH e RDP podem ser beneficiados, nesses casos os dados já criptografados trafegariam em outro túnel criptografado.

Existem duas desvantagens em relação a essa arquitetura. A primeira delas é que a complexidade da rede aumenta, devido a inclusão de novos elementos e novos caminhos. A segunda desvantagem é em relação ao custo do servidor central, custos como aquisição de hardware e sua manutenção. Porém essas desvantagens não são maiores que os benefícios proporcionados. Assim sendo, mesmo com essas desvantagens a arquitetura ainda é viável e passível de uso.

1.3.3 Arquitetura para alta disponibilidade.

O problema de arquiteturas centralizadas é a questão da disponibilidade. Caso o elemento central fique indisponível toda a estrutura que depende desse elemento ficará indisponível. Isso significa que no caso de um problema no servidor central, todo o serviço de acesso remoto da rede ficaria inoperante. Ou seja, esse tipo de arquitetura introduz um elemento de falha. Um elemento que representa um risco para o funcionamento normal da rede. Caso todo o serviço de acesso remoto fique

indisponível as consequências seriam incalculáveis para o departamento de informática da empresa.

Para resolver o problema da disponibilidade é proposta a utilização de um ambiente com redundância, conforme mostra a Figura 03. Nesse ambiente existem dois servidores que respondem pelas requisições, chamados de node01 e node02. Antes dos servidores existe um *switch* de balanceamento, esse *switch* recebe todas as solicitações de acesso e realiza o balanceamento para um dos servidores disponíveis. O *switch* além de realizar o balanceamento também realiza o monitoramento de disponibilidade dos servidores. Caso o *switch* detecte que um dos nodes parou de responder, ele não encaminha mais requisições para essa maquina. Ou seja, em caso de problemas em um dos nodes a estrutura continua funcionando pois o outro node pode responder pelas requisições.

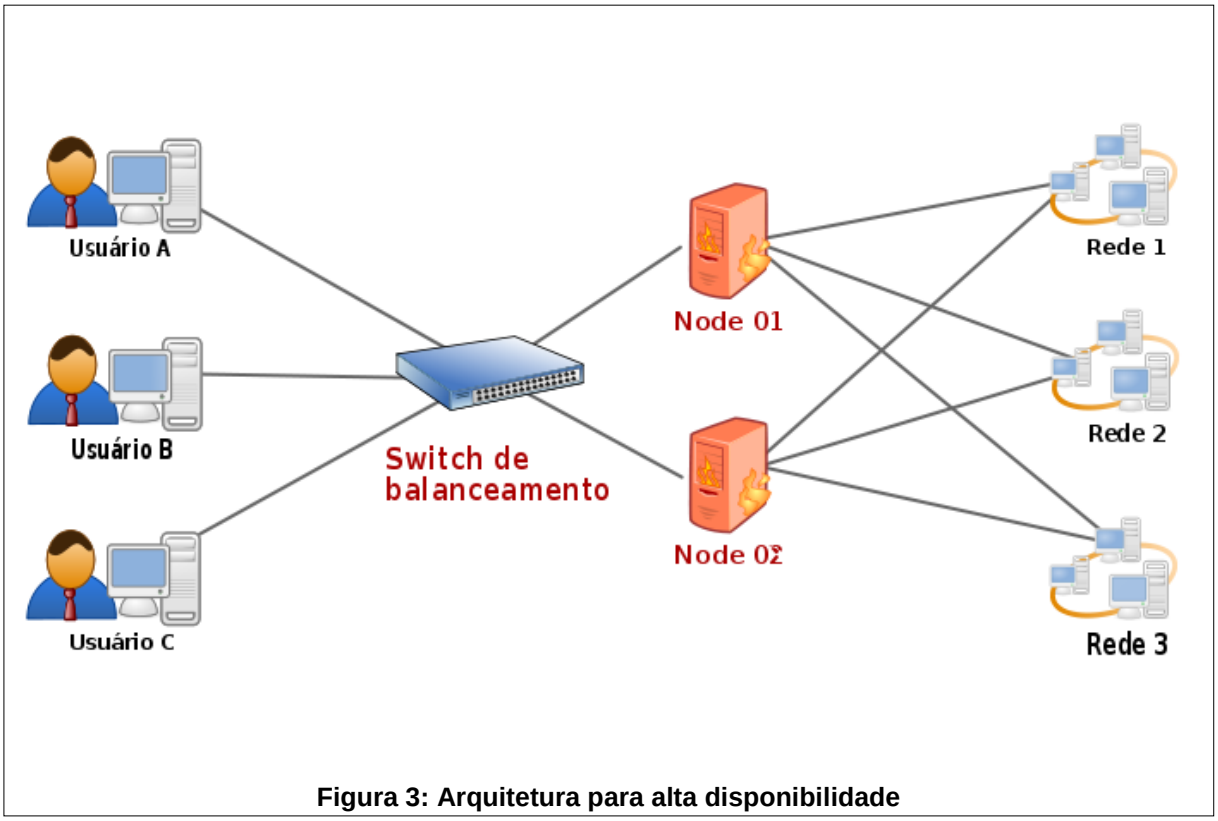


Figura 3: Arquitetura para alta disponibilidade

Toda essa arquitetura é transparente para o usuário, o qual não precisa saber detalhes internos da estrutura. O usuário também não é afetado em caso de problemas de disponibilidade de um *node*. O número de nodes também pode ser alterado de acordo com a necessidade. Caso a demanda de acessos aumente basta adicionar um novo node. Para que tudo isso funcione corretamente as aplicações devem estar configuradas para enviar as solicitações para o *switch* e não para os *nodes* em específico. Recomenda-se a criação de um domínio, e esse domínio deve ser resolvido para um IP do *switch* de balanceamento

1.4 SERVIDOR

Seguindo o modelo proposto na arquitetura, necessitamos de uma aplicação servidor que recebe todas as requisições de acesso. Essa aplicação após receber as requisições de acesso deve redirecionar as solicitações para o equipamento que o usuário deseja acessar remotamente. Além dessa funcionalidade a aplicação deve realizar uma consulta a uma base de dados LDAP para verificar se o usuário que está realizando o acesso remoto possui permissão para tal, deve também verificar se o nome de usuário e senha estão corretos. A aplicação deve também gravar em uma base de dados informações sobre o acesso que está sendo realizado, para que posteriormente seja possível realizar auditorias desses acessos.

Uma característica importante da aplicação servidor é que ela deve oferecer suporte a um protocolo que permita a conexão de clientes que utilizem diferentes plataformas. A comunicação entre a aplicação e o cliente deve ser realizada por um canal criptografado, garantindo a confidencialidades dos dados que estão sendo transportados.

1.4.1 Aplicação Servidor

Para prover todas as funcionalidades que a aplicação servidora deve possuir, foi escolhido realizar o desenvolvimento de um novo programa baseado no servidor OpenSSH.

A escolha do servidor OpenSSH foi baseada em diversos fatores. Primeiramente a questão de estabilidade do OpenSSH, o projeto possui mais de 10 anos de existência, sendo largamente utilizado por diversos sistemas operacionais, inclusive por fabricantes de hardware. Segundo a questão do protocolo de comunicação utilizado, o OpenSSH utiliza para comunicação entre as aplicações clientes e servidor o protocolo SSH, que é reconhecidamente um protocolo seguro com bons algoritmos de criptografia. O último ponto que contribui para a sua escolha, foi o fato de já possuir funcionalidades que permitem o redirecionamento de conexões recebidas.

Foi necessário fazer algumas alterações no código fonte do servidor OpenSSH para que ele pudesse atender a todos os requisitos.

As principais modificações realizadas foram:

- Inclusão de uma estrutura de dados para guardar informações sobre os acessos. Foi incluída uma estrutura de dados para salvar os seguintes dados: nome do usuário, IP de origem, porta de Origem, IP do servidor de destino, porta de destino, total de *bytes* enviado e total de bytes recebidos.

- Bloquear acesso do usuário ao *Shell* do servidor. Por medidas de segurança foi bloqueado acesso ao *Shell* do servidor na qual está sendo executada a aplicação servidor. Desse modo o usuário não pode realizar qualquer modificação no equipamento que está rodando a aplicação servidor.

- Permitir aos usuários apenas o redirecionamento de conexões. Após autenticado no servidor o usuário somente pode realizar conexões para outros equipamentos, qualquer outra operação que o usuário tente será negada.

- Gravar informações de acesso em log apropriada. A aplicação grava todos os registros de acessos em um arquivo de log destinado apenas para esta finalidade.

1.4.2 Autenticação

Um dos requisitos para o servidor é que ele permita a autenticação em base de dados LDAP. Para possibilitar a realização dessa tarefa foi utilizado o PAM (*Pluggable Authentication Modules*). O PAM é um mecanismo para integrar múltiplas

autenticações de baixo nível em uma API (*Application Programming Interface*) de alto nível. Isso permite que programas que dependem de autenticação sejam desenvolvidos sem estarem presos a um esquema de autenticação especifico.

O LDAP (*Lightweight Directory Access Protocol*) é um protocolo para acessar informações de diretórios utilizando uma rede IP. Um diretório é um conjunto organizado de registros, por exemplo: o nome do usuário, login e telefone podem ser um registro. Dessa forma, é possível utilizar o LDAP para guardar informações como usuário e senha e prover serviços de identificação e autenticação. É possível ainda realizar a autorização através da criação de grupos de usuários na base LDAP, concedendo a cada grupo o nível de acesso desejado (OPENLDAP, 2011), (TRIGO, 2007).

Foi escolhida a utilização do PAM para realizar a autenticação na base LDAP por dois motivos. O primeiro motivo é devido a ser uma aplicação estável, possui mais de 10 anos de desenvolvimento, suporte a vários tipos de protocolos, além de ser nativo em diversas distribuições Linux. O segundo motivo é o fato do servidor OpenSSH já possuir nativamente em seu código integração com o PAM.

Para integrar o PAM, LDAP e OpenSSH basta editar alguns arquivos de configuração. A sequência de configuração é a seguinte:

- Colocar no arquivo de configuração do ssh Server, chamado sshd_config, a flag "UsePAM yes". Isso faz com que o servidor de SSH realize a autenticação no PAM

- Dentro dos arquivos de configuração do PAM é necessário editar o arquivo responsável pelo serviço SSH. Nesse arquivo deve ser adicionada a configuração para que o serviço SSH seja autenticado no LDAP.

- Editar o arquivo de integração do PAM com o LDAP. Nesse arquivo deve ser informado o endereço do servidor LDAP, a versão do protocolo, o endereço do diretório LDAP e também habilitado o parâmetro para utilizar criptografia na comunicação entre eles.

- Editar o arquivo de configuração do LDAP, chamado do "ldap.conf", colocando informações como o endereço do servidor, endereço do diretório LDAP e também habilitar o parâmetro para utilização de TLS.

1.4.3 Geração de logs

Um dos principais objetivos do servidor é permitir a auditoria dos acessos. Para que isso possa ser realizado é necessário que os dados sejam salvos. Seria possível salvar os dados diretamente em um banco de dados ou então em um arquivo texto. Foi optado pela segunda opção, gravar os dados em uma log de acesso. Essa decisão foi tomada para tornar a aplicação independente de versão de banco de dados. Além de tornar a aplicação mais robusta, pois o servidor funciona sem depender de sistemas externos como um SGBD. Outra vantagem dessa opção é a questão da flexibilidade. Com os dados salvos em um arquivo texto, pode-se utilizar diversos aplicativos para: analisar, normalizar e até mesmo exportar esses dados para outros lugares.

Para gerar as logs foi utilizado o utilitário Syslog. O Syslog é um programa para gravação de mensagens em log, amplamente utilizado em sistemas operacionais derivados do UNIX. Ele permite a separação de programas que geram mensagens, dos sistemas que armazenam e dos softwares que realizam analise. Além da aplicação em si, existe o protocolo utilizado. Esse protocolo possui o mesmo nome e foi padronizado pela IETF (*Internet Engineering Task Force*) pela RFC (*Request For Comments*) 3164 . O protocolo permite que seja implantado um servidor centralizado de logs, possibilitando que os demais equipamentos enviem as suas logs para esse servidor. Tais mensagens podem ser enviadas tanto por UDP quanto TCP, utilizando a porta 514 (RFC 3164, 2011).

A escolha do Syslog foi devido a três fatores. Primeiramente a estabilidade da aplicação, a qual possui longos anos de desenvolvimento. Segundo fator foi a questão de ser um protocolo aberto, padronizado pela IETF e implementado por diversos fabricantes. E terceiro fator, foi o fato de já ser utilizado pelo servidor OpenSSH como padrão para mensagens de *logging*.

Para que fosse possível a exportação de mensagem para auditoria foi necessário realizar as seguintes atividades:

- Criar uma lógica de exportação dos dados para a log. Essa exportação é realizada a cada vez que o usuário abre uma conexão com o servidor ou quando

fecha a conexão, essa exportação também é realizada quando o usuário abre uma sessão de acesso remoto ou quando ele finaliza a sessão.

- Editar o arquivo de configuração utilizado pelo Syslog. Foi editado o arquivo de configuração para que o Syslog cria-se o arquivo "safefw.log" apenas para log do servidor, e cria-se o arquivo "safefw-access.log" apenas para logar os acessos.

1.4.4 Exportação dos dados

Para permitir a consulta dos acessos é necessário que os dados estejam salvos em um formato que agilize esse processo. Embora seja possível a consulta diretamente no arquivo de log, esse tipo de processo possui restrições. Consultar diretamente os arquivos pode resultar em consultas mais lentas, pois os dados não estão normalizados. Além disso, para realizar consultas complexas é necessário um esforço maior do que seria necessário, caso estivesse sendo utilizado um banco de dados.

Devido aos argumentos expostos acima, foi desenvolvido dois *script* na linguagem Perl para exportação dos dados.

O primeiro *script* é o "read.pl". Esse script monitora o arquivo de log gerado pelo servidor. Caso ele verifique que foi adicionada uma informação nova, essa informação é salva no banco de dados. Esses dados não são normalizados, apenas salvos diretamente no banco em uma tabela chamada buffer.

O segundo *script* desenvolvido é chamado "monitor.pl". Esse script monitora a tabela “buffer” do banco de dados, tabela alimentada pelo *script* "read.pl". Quando o script encontra um novo registro na tabela buffer, ele lê o registro e realiza o processamento, de acordo com o tipo de informação. Caso seja a informação de uma nova conexão ele adiciona a tabela apropriada. Caso seja a informação de fim de uma conexão, ele localiza no banco o registro de inicio da conexão e marca um parâmetro informando que aquela conexão foi finalizada. Além dessas funções o *script* também realiza o monitoramento das conexões ativas, visando localizar conexões que não foram finalizadas da maneira correta. Uma vez encontrada esse

tipo de informação, ele atualiza o banco de dados informando que aquela conexão já não está ativa.

Após desenvolvido os *scrits* em Perl para exportação e normalização, foi desenvolvido dois *scripts* para inicializá-los quando o servidor é ligado. Foram criados então dois arquivos: um arquivo em "/etc/init.d/read" e outro em "/etc/init.d/monitor". O primeiro arquivo é responsável por iniciar o *script* "read.pl", criado para realizar o parser do arquivo de log. O segundo arquivo é responsável por iniciar o script "monitor.pl", criado para normalizar os dados.

1.4.5 Instalação da aplicação

O processo de instalação do servidor é simples e não requer muitos passos. Antes de tudo é necessário criar um diretório, no qual serão armazenados todos os arquivos utilizados para executar a aplicação. Dentro do diretório é recomendado criar duas pastas, a pasta com o nome de "bin" e outra com o nome de "etc". Na primeira pasta deve-se colocar o arquivo binário com o executável. Na segunda pasta deve-se colocar os arquivos de configuração.

Dentro da pasta "etc" deve-se colocar os seguintes arquivos: um arquivo com a chave privada do SSH, um arquivo com a chave publica do SSH e um arquivo com toda a configuração do servidor. O arquivo com toda a configuração do servidor chama-se "sshd_config". Dentro desse arquivo basta informar o endereço IP que será utilizado para receber as conexões, recomenda-se também setar o parâmetro UseDNS para "no" para obter uma performance melhor.

Por último, basta criar um *script* para iniciar a aplicação, toda a vez que o servidor for reiniciado. O *script* deve ser colocado dentro da pasta "/etc/init.d". Assim a aplicação será iniciada automaticamente.

1.5 CLIENTES

Para utilizar a solução de acesso remoto é necessário que o usuário possua instalado, em sua estação, um programa cliente. O programa cliente é quem conecta com o servidor central. Ele também é responsável por enviar as credenciais de acesso do usuário para que o servidor central realize a autenticação. Outra tarefa desempenhada pelo programa cliente é criptografar os dados que saem da estação do usuário para o servidor central. Todos os dados que trafegam devem ser criptografados desde o inicio da conexão até o seu final. Para realizar todas essas atividades foram desenvolvidos dois tipos de programa, um deles funciona por linha de comando e outro em ambiente gráfico.

O cliente gráfico foi desenvolvido na linguagem Java e pode funcionar em qualquer sistema operacional. Devido a característica de portabilidade da linguagem a mesma versão funciona em diferentes plataformas não sendo necessário alterações ou recompilações. Esse cliente é especialmente útil para aqueles usuários que não gostam de utilizar a linha de comando. O programa suporta todos os protocolos de acesso remoto oferecidos pelo SafeFW: SSH, Telnet, RDP e RFB. Além de realizar o acesso o programa também possuí outras funcionalidades como salvar configuração de sessão.

Os clientes por linha de comando foram desenvolvidos na linguagem C. Esses clientes foram desenvolvidos principalmente para serem utilizados em sistemas que não possuem interface gráfica habilitada, para aqueles usuários que preferem utilizar a linha de comando e para *scripts* que necessitam acessar remotamente servidores para executar comandos. O Linux é o sistema que está mais próximo dessas características: versões com ausência de interface gráfica, utilização de *scripts* e usuários que preferem utilizar linha de comando. Por esse motivo é que foram criados clientes de linha de comando apenas para esse sistema operacional. Os usuários que não utilizam Linux podem fazer uso do cliente gráfico.

Para facilitar a instalação, todos os clientes desenvolvidos foram empacotados em pacotes “.deb”. Pacotes “.deb” são pacotes utilizados pela distribuição Debian, e seus derivados, para instalação de novos softwares. Uma vez que exista o pacote pronto basta utilizar algum sistema de gerenciamento de pacote,

que a instalação é realizada automaticamente. O próprio sistema de gerenciamento de pacotes se encarrega de colocar os arquivos nos diretórios corretos e realizar os ajustes necessário. A opção por utilizar pacotes ".deb" é devido ao Debian ser uma distribuição livre, largamente utilizada e base para diversas distribuições.

Um recurso interessante que foi adicionado aos clientes em C foi o reaproveitamento de sessão. O recurso permite que caso o usuário realize uma segunda conexão para um servidor, seja reaproveitado a sessão que já existe. Assim não é necessário abrir uma nova sessão, o que garante uma economia de recursos computacionais. Todos os dados enviados seguem pelo túnel criptografado que já está aberto.

1.5.1 Cliente em C para Telnet

O Telnet foi um dos primeiros protocolos de acesso remoto a ser desenvolvido. Embora seja um protocolo antigo, muitas aplicações ainda o utilizam. O protocolo conta também com um legado de vários equipamentos antigos que vinham com ele instalado por padrão. Um dos principais problemas do Telnet que levou o surgimento de novos protocolos como o SSH é a falta de criptografia dos dados. Todas as informações trafegadas entre o cliente e o servidor ocorrem em texto plano, assim é possível identificar até mesmo a senha digitada para se ter acesso aos equipamentos.

Com o cliente desenvolvido, chamado de safefw-telnet, esse problema é resolvido. A comunicação entre o cliente Telnet e o servidor centralizado de controle de acesso, ocorre em um canal criptografado. Assim os dados informados pelo usuário são enviados por um canal seguro. O servidor centralizado se encarrega de descriptografar os dados em encaminhar para a maquina de destino. Para garantir um nível maior de segura recomenda-se que o servidor centralizado e a maquina de destino estejam em uma rede segura, pois dessa forma não existe nenhum risco na comunicação. Mesmo que o cliente esteja em uma rede com baixo nível de segurança a comunicação ainda poderia ocorrer com um elevado nível de segurança, graças a criptografia utilizada pelo cliente.

A figura 04 mostra as opções de uso programa. A parte de cima mostra todas as opções disponíveis, já a parte inferior mostra as opções sumarizadas. Basicamente mantiveram-se todas as opções que já existiam no cliente normal do Telnet e foram adicionas novas opções utilizadas pela solução de acesso remoto.

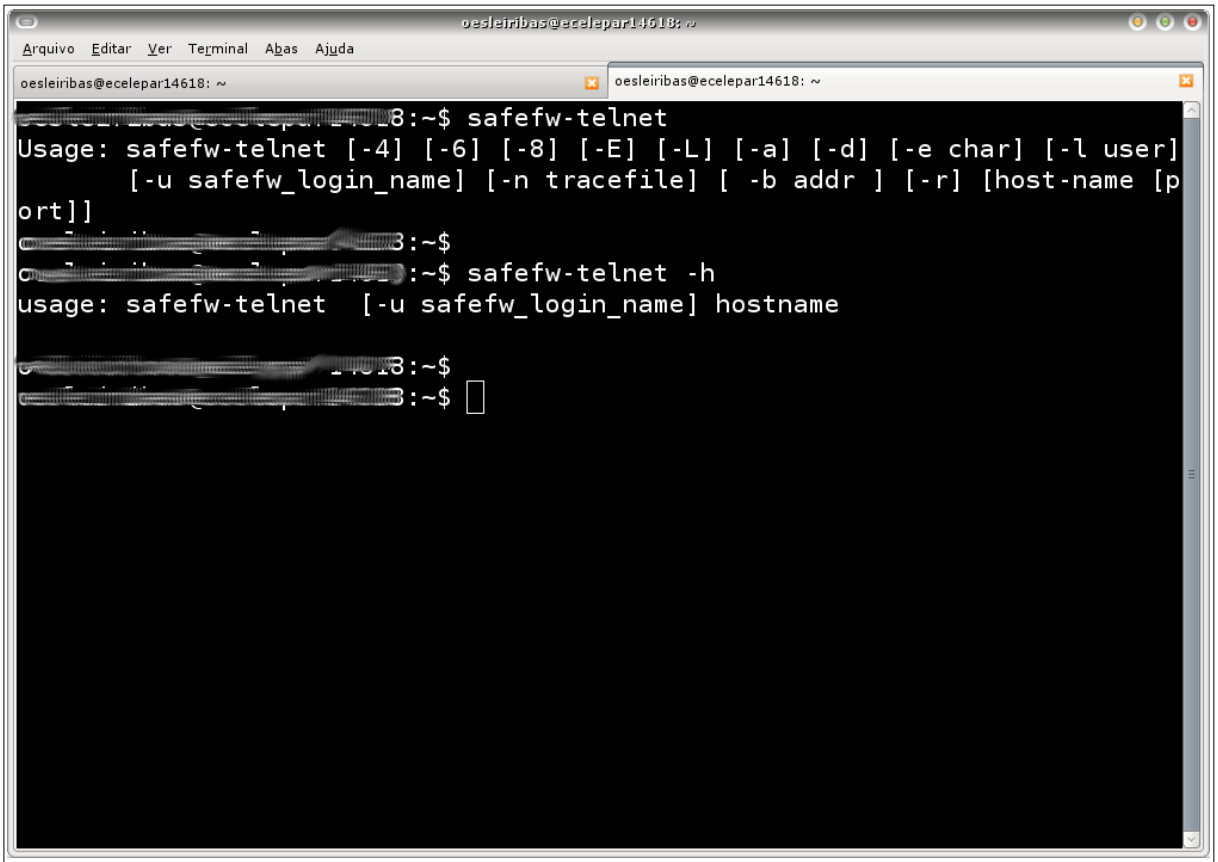


Figura 4: Ajuda do programa safefw-telnet

Foi desenvolvido também um pacote ".deb" para facilitar a instalação do programa. Abaixo constam as principais diretivas de configuração do pacote criado:

- Package: safefw-telnet
- Priority: extra
- Section: net
- Installed-Size: 52
- Architecture: all
- Source: safefw
- Version: 1.0.6

- Depends: safefw-base, openssh-client, net-tools, telnet

- Description: Instala o pacote safefw em C para acesso com o telnet.

A figura 05 mostra o programa realizando um acesso remoto para um servidor. Observe que primeiramente é realizada a autenticação no servidor central e logo após isso é realizado a autenticação no equipamento de destino.

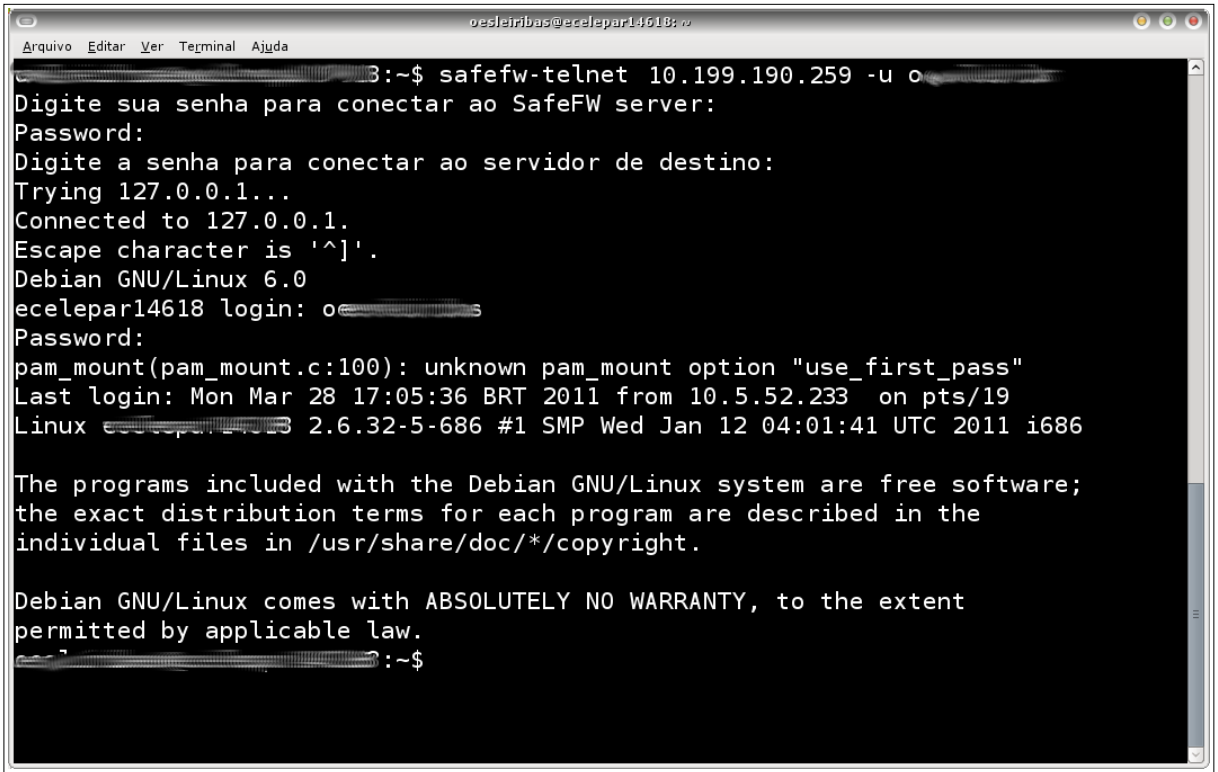


Figura 5: Programa safefw-telnet em uso

1.5.2 Cliente em C para SSH

O SSH é um protocolo que surgiu como sucessor do Telnet. A grande vantagem do SSH em relação ao seu antecessor é a utilização de criptografia. Todas as mensagens trocadas entre o cliente e o servidor são criptografadas, utilizando algoritmos publicamente conhecidos. Possui também vários mecanismos de segurança, entre esses mecanismos está o que dificulta a realização de ataques de homem do meio.

O Cliente SSH, chamado de safefw-ssh, proporciona um nível de segurança ainda maior ao realizar um acesso remoto. Os dados que antes já eram enviados em um canal criptografado criado pelo SSH, agora passam por um segundo canal de criptografia, criado entre o cliente safefw-ssh e o servidor central. Isso significa que para se ter acesso as informações é necessário quebrar a criptografia de dois canais, o que é muito difícil de ser realizado.

Durante o desenvolvimento do cliente buscou-se manter disponível todas as opções que o cliente original oferecia. Desde autenticação através de chaves publicas até autenticação dos hosts.

A figura 06 mostra as opções de uso do programa. A parte superior mostra todas as opções disponíveis, já a parte inferior mostra as opções sumarizadas. Observe a enorme quantidade de opções disponíveis para uso. Um detalhe interessante a ser observado é a possibilidade de execução remota de comandos. Após o parâmetro hostname é possível especificar um comando ou um conjunto de comandos a serem executados após realizada a conexão.

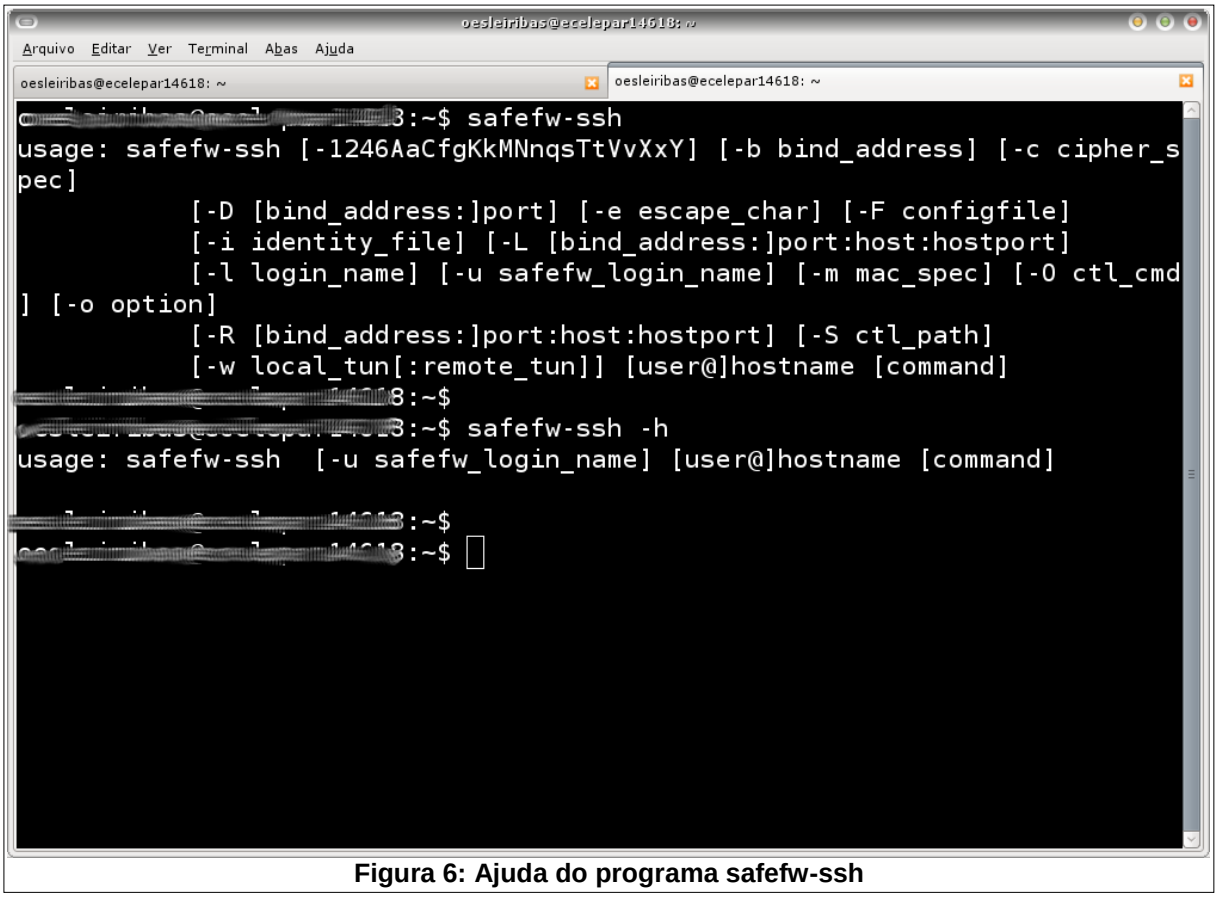


Figura 6: Ajuda do programa safefw-ssh

Foi desenvolvido também um pacote ".deb" para facilitar a instalação do programa. Abaixo constam as principais diretivas de configuração do pacote criado:

- Package: safefw-ssh
- Priority: extra
- Section: net
- Installed-Size: 56
- Architecture: all
- Source: safefw
- Version: 1.0.6
- Depends: safefw-base, openssh-client, net-tools
- Description: Instala o pacote safefw em C para acesso com o SSH.

A figura 07 mostra o programa realizando um acesso remoto para um servidor. Observe que primeiramente é realizada a autenticação no servidor central e logo após isso é realizado a autenticação no equipamento de destino.

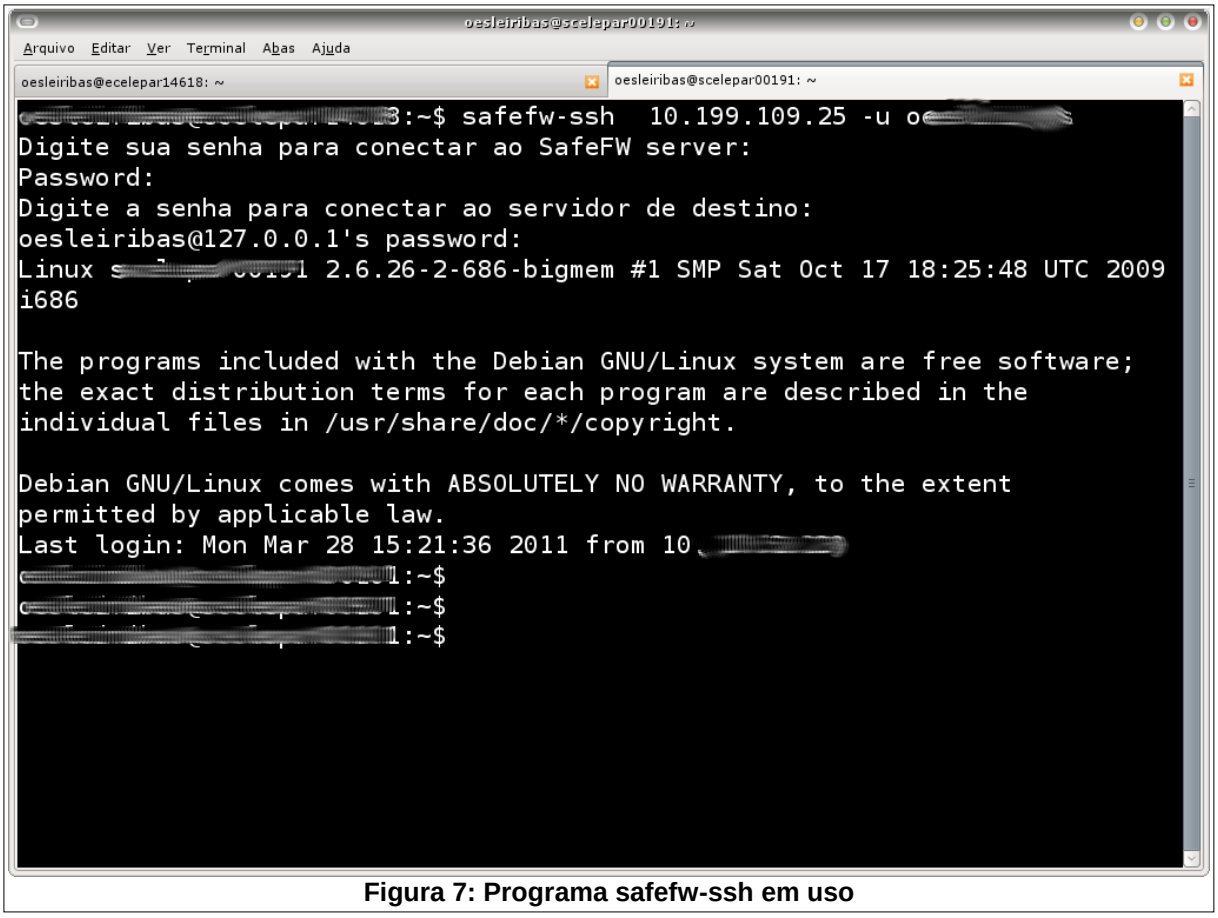


Figura 7: Programa safefw-ssh em uso

1.5.3 Cliente em C para SCP

O SCP é um aplicativo para copia segura de arquivos entre *hosts* de uma rede. Utiliza o protocolo SSH para realizar a comunicação entre os *hosts*, garantindo o mesmo nível de segurança que se têm ao utilizar o SSH. Para que o programa funcione basta que ambos: servidor e cliente suportem o protocolo SSH. É preferível a utilização do scp para copia de arquivos a utilizar protocolos como FTP o qual não utiliza nenhum mecanismo de criptografia dos dados.

Uma característica interessante do SCP é a flexibilidade para copiar arquivos. É possível realizar a copia tanto do cliente para o servidor quando do servidor para o cliente. No momento da copia pode-se passar o caminho especifico da origem do arquivo e o caminho de destino que o arquivo deve ser salvo. Caso o usuário deseje copiar um conjunto de arquivos de única vez pode-se utilizar *wildcards* como o asterisco, desse modo todos os arquivos correspondentes ao *wildcard* serão copiados. Todas essas opções de uso foram preservadas no cliente desenvolvido, chamado de safefw-scp.

A figura 08 mostra as opções de uso programa. A parte superior mostra todas as opções disponíveis, já a parte inferior mostra as opções sumarizadas. Verifique que existem na ajuda do programa os exemplos file1 e file2, que respectivamente corresponde a origem do arquivo e ao destino.

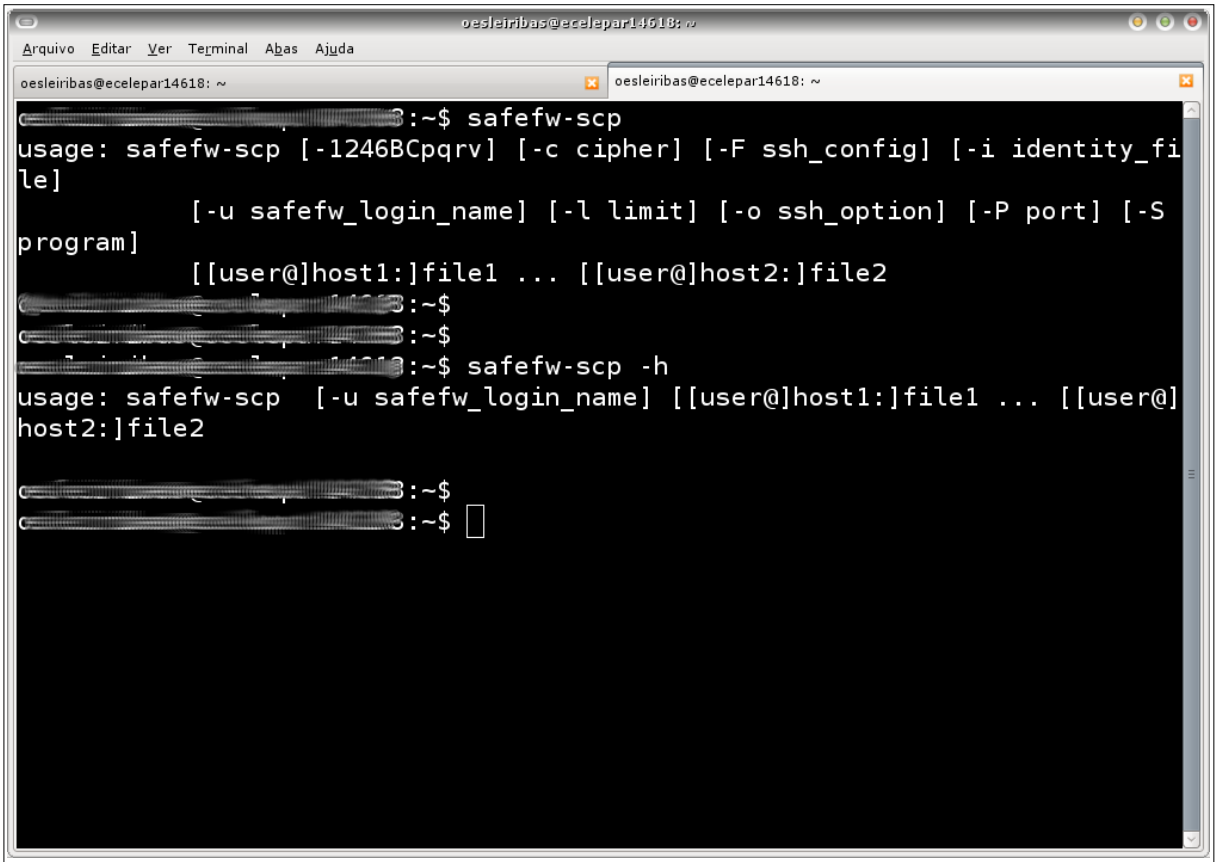


Figura 8: Ajuda do programa safefw-scp

Foi desenvolvido também um pacote ".deb" para facilitar a instalação do programa. Abaixo constam as principais diretivas de configuração do pacote criado:

- Package: safefw-scp
- Priority: extra
- Section: net
- Installed-Size: 52
- Architecture: all
- Source: safefw
- Version: 1.0.6
- Depends: safefw-base, openssh-client, net-tools
- Description: Instala o pacote safefw em C para copia segura de arquivos utilizando o SCP

A figura 09 mostra o programa realizando a copia de arquivos entre o cliente e um servidor. Observe que primeiramente é realizada a autenticação no servidor central e logo após isso é realizado a autenticação no equipamento de destino. Verifique que nesse exemplo é utilizado um *wildcard* asterisco, o qual possibilita copiar todos os arquivos que estão na pasta "/tmp" e possuem o nome iniciando com "teste" e terminando com ".txt"

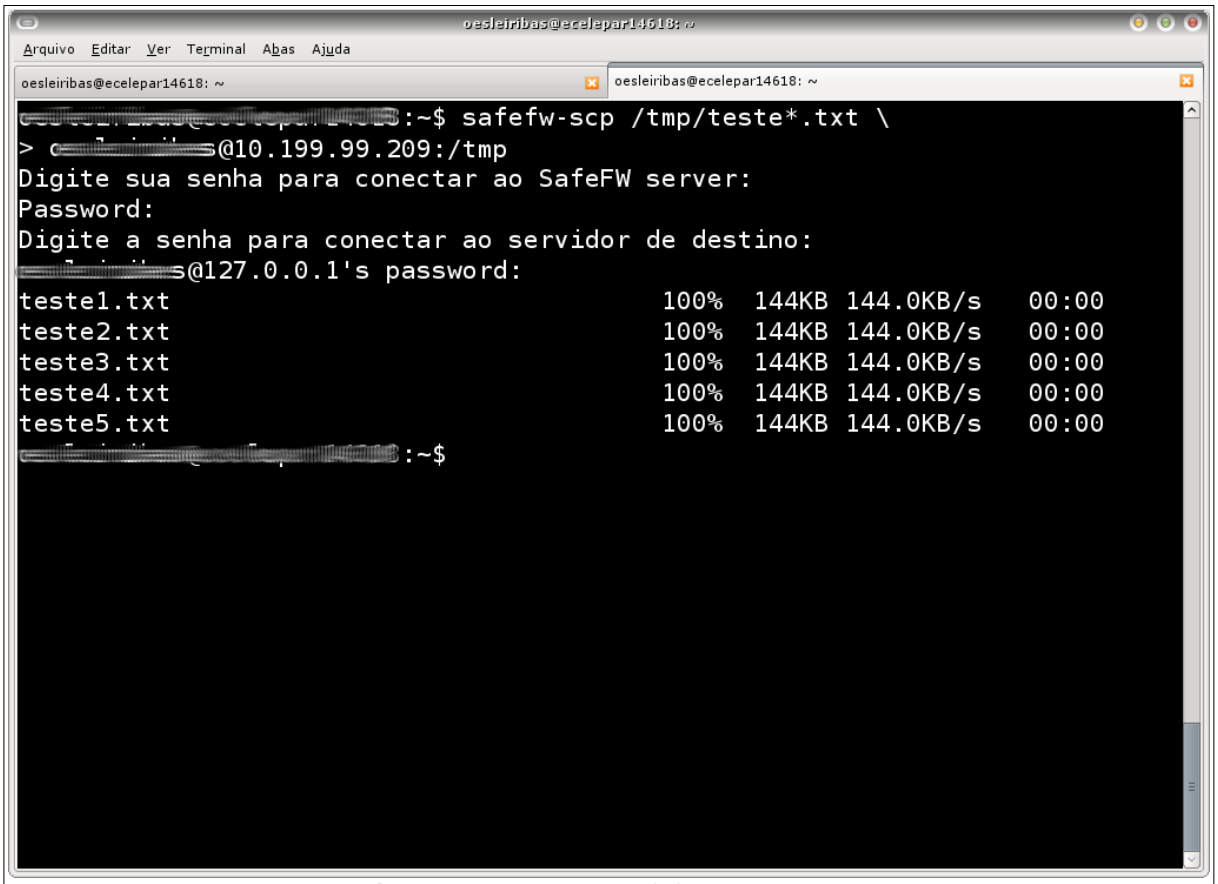


Figura 9: Programa safefw-scp em uso

1.5.4 Cliente em C para RFB

O RFB é um protocolo que permite acesso remoto a interface gráfica de um equipamento. Existem várias aplicações que implementam esse protocolo, sendo a mais conhecida de todas, o VNC. Através do protocolo é possível visualizar todas as

informações que aparecem na tela do equipamento, é possível até mesmo ver a movimentação do cursor do mouse. Dessa forma é possível "compartilhar" a interface gráfica entre diversos usuários.

O cliente desenvolvido para suportar o RFB chama-se safefw-vnc, o nome foi escolhido para deixar mais claro aos usuários que eles podem utilizar esse cliente para realizar acessos em equipamentos com VNC. Embora, o mesmo cliente pode ser utilizado para outros aplicativos que implementem o protocolo RFB. O cliente também implementa suporte a criptografia assim como os demais clientes, assim toda a comunicação é realizada em um canal seguro.

A figura 10 mostra as opções de uso programa. Verifique que não existe muitas opções de uso para o protocolo RFB, os menus completo e o sumarizado apresentam o mesmo conteúdo. Basicamente deve-se informar o nome do *host* de destino, caso deseje é possível informar o nome de usuário para realizar a autenticação no servidor central.

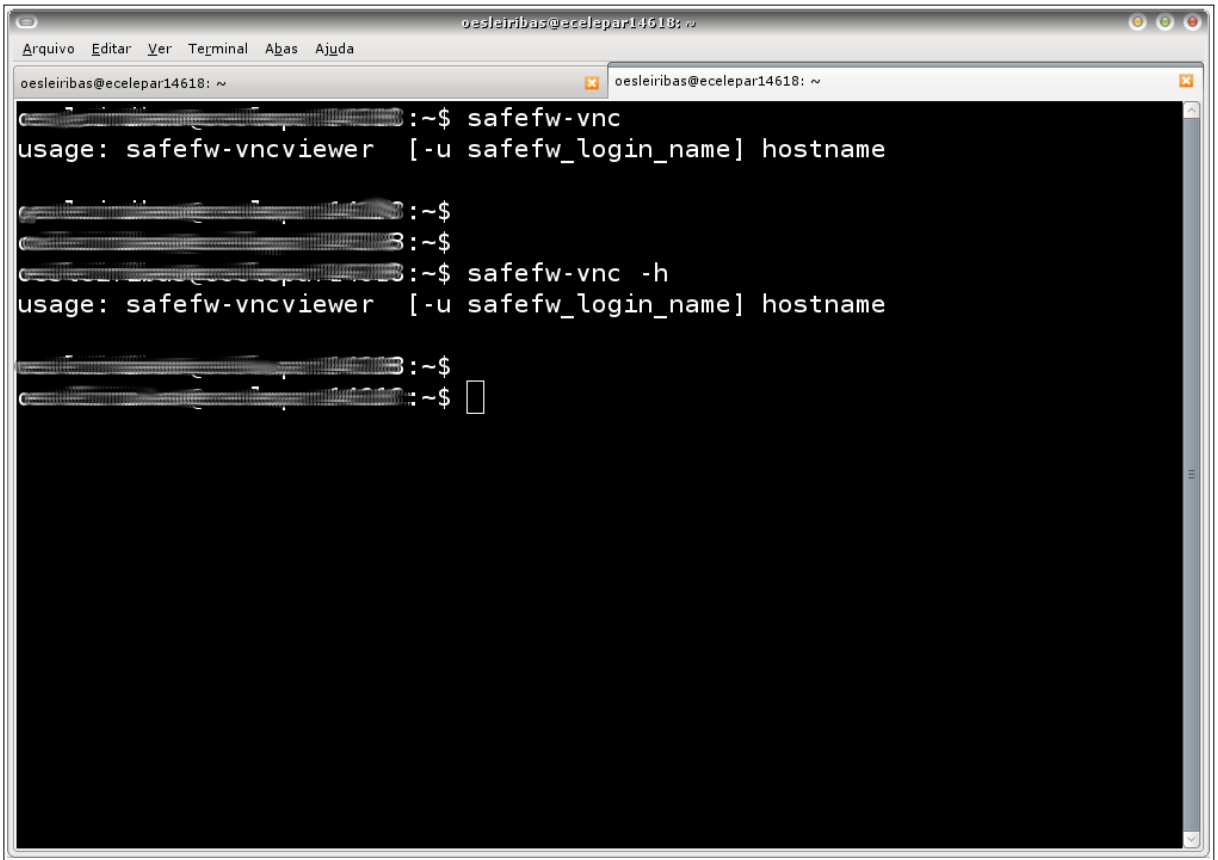


Figura 10: Ajuda do programa safefw-vnc

Foi desenvolvido também um pacote ".deb" para facilitar a instalação do programa. Abaixo constam as principais diretivas de configuração do pacote criado:

- Package: safefw-vnc
- Priority: extra
- Section: net
- Installed-Size: 52
- Architecture: all
- Source: safefw
- Version: 1.0.6
- Depends: safefw-base, openssh-client, net-tools, vncviewer
- Description: Instala o pacote safefw em C para acesso remoto utilizando o protocolo RFB.

A figura 11 mostra o programa realizando um acesso remoto. Verifique que primeiramente é realizada a autenticação no servidor central e em seguida é

realizada a autenticação no servidor de destino. Por fim, é realizada a configuração da parte gráfica entre o cliente e o servidor.

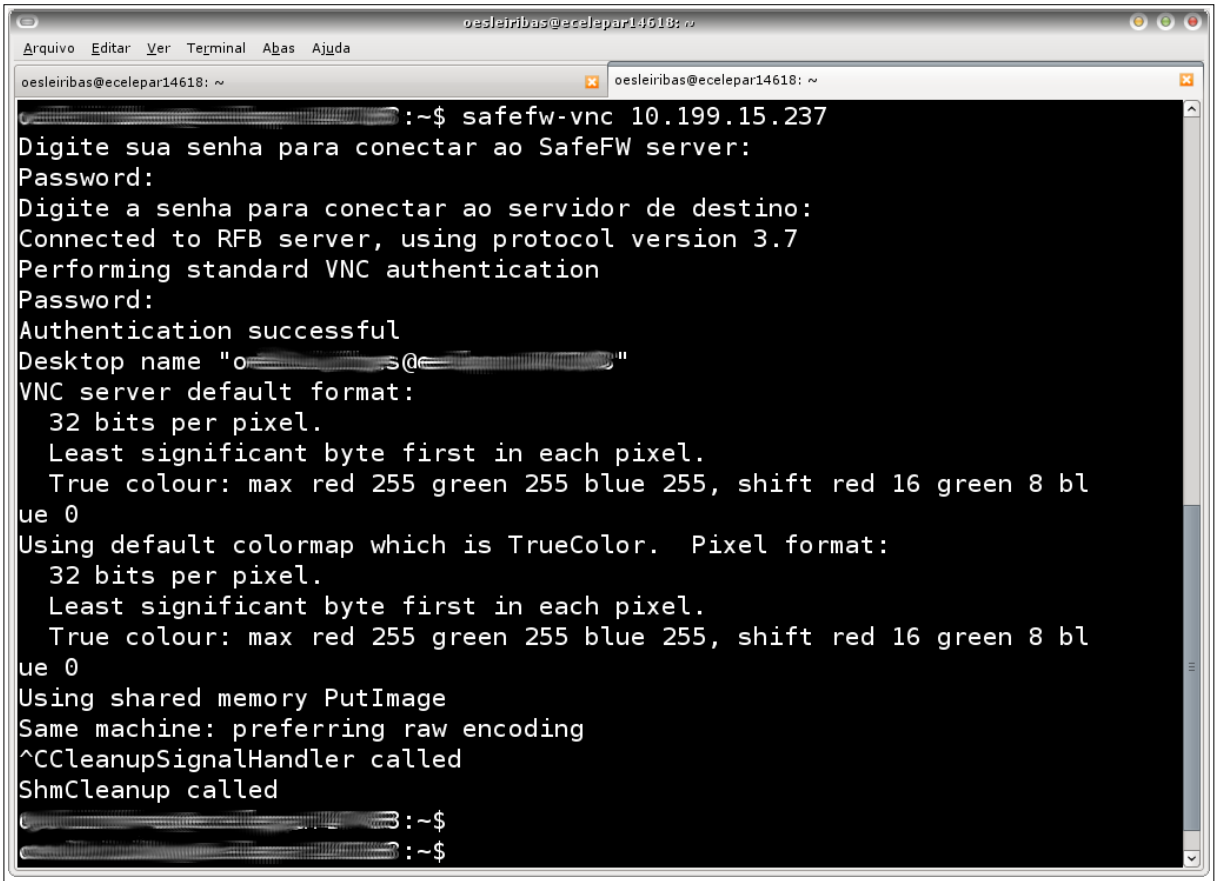


Figura 11: Programa safefw-vnc em uso

1.5.5 Cliente em C para Terminal Service

O RDP é um protocolo proprietário desenvolvido pela Microsoft para acesso remoto. Esse protocolo é utilizado por várias versões do sistema operacional Windows, principalmente as versões servidores. Embora seja proprietário existe cliente para outros sistemas operacionais como o Linux. Desse modo é possível através de uma estação Linux realizar um acesso remoto para um servidor Windows.

O cliente desenvolvido chama-se safefw-rdesktop. O nome é uma referência ao programa rdesktop, versão do cliente Linux que suporta o protocolo RDP. O

cliente desenvolvido possui suporte a todas as funcionalidades que o RDP fornece, como: redirecionamento do sistema de impressão e mapeamento remoto de unidades de rede. O cliente também utiliza um canal criptografado para comunicação entre a estação e o servidor central.

A figura 12 mostra as opções de uso do programa. Na parte superior são mostradas as opções sumarizadas, já na parte de baixo são mostrados todos os parâmetros suportados pelo cliente. Conforme mostra a figura 12 existe um grande número de parâmetros que podem ser utilizados ao chamar o programa.

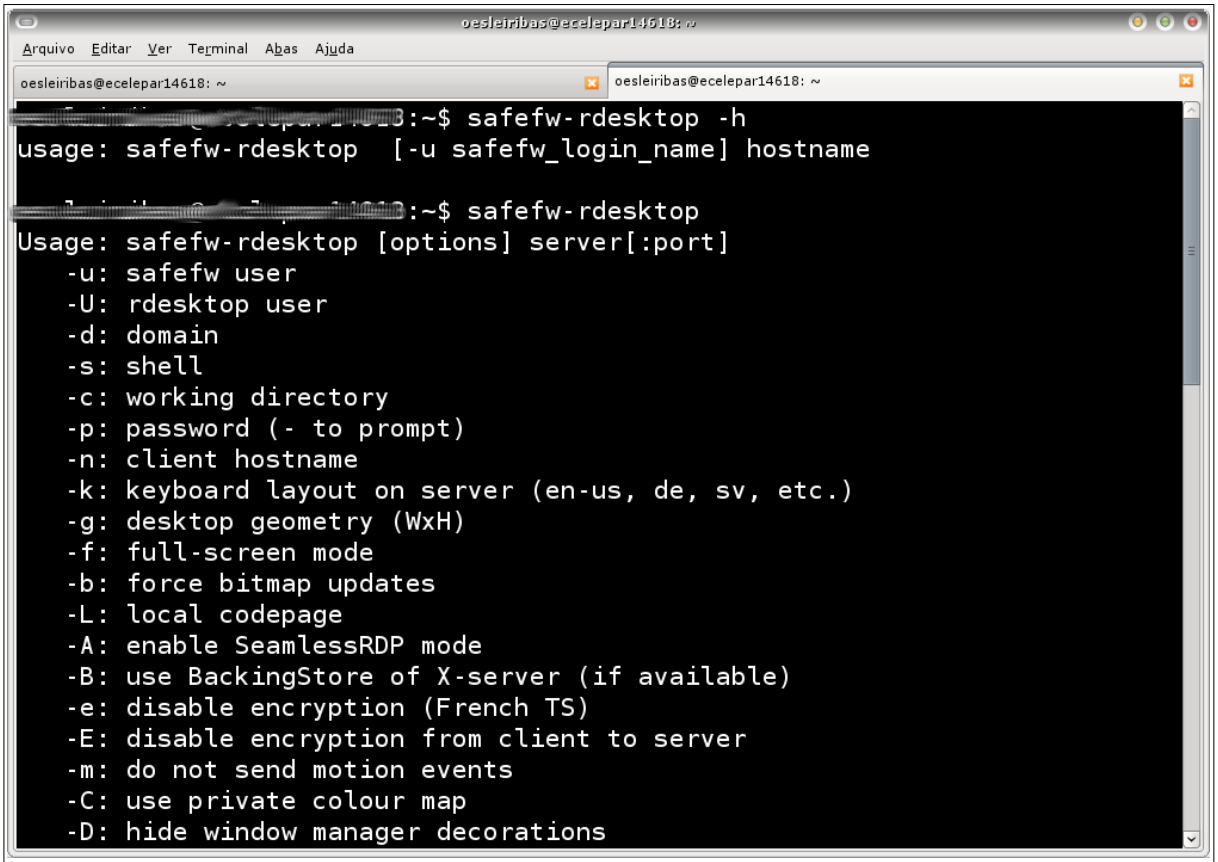


Figura 12: Ajuda do programa safefw-rdesktop

Foi desenvolvido também um pacote “.deb" para facilitar a instalação do programa. Abaixo constam as principais diretivas de configuração do pacote criado:

- Package: safefw-vnc
- Priority: extra
- Section: net
- Installed-Size: 52

- Architecture: all

- Source: safefw

- Version: 1.0.6

- Depends: safefw-base, openssh-client, net-tools, vncviewer

- Description: Instala o pacote safefw em C para acesso remoto utilizando o protocolo RFB.

A figura 13 mostra o programa realizando um acesso remoto. Na parte da frente é mostrado o cliente realizando o acesso. Na parte de trás é mostrado o cliente já conectado a um servidor Windows 2000 Server. Conforme pode-se observar é especificado um usuário para realizar a autenticação no servidor central e outro usuário para acessar o servidor.

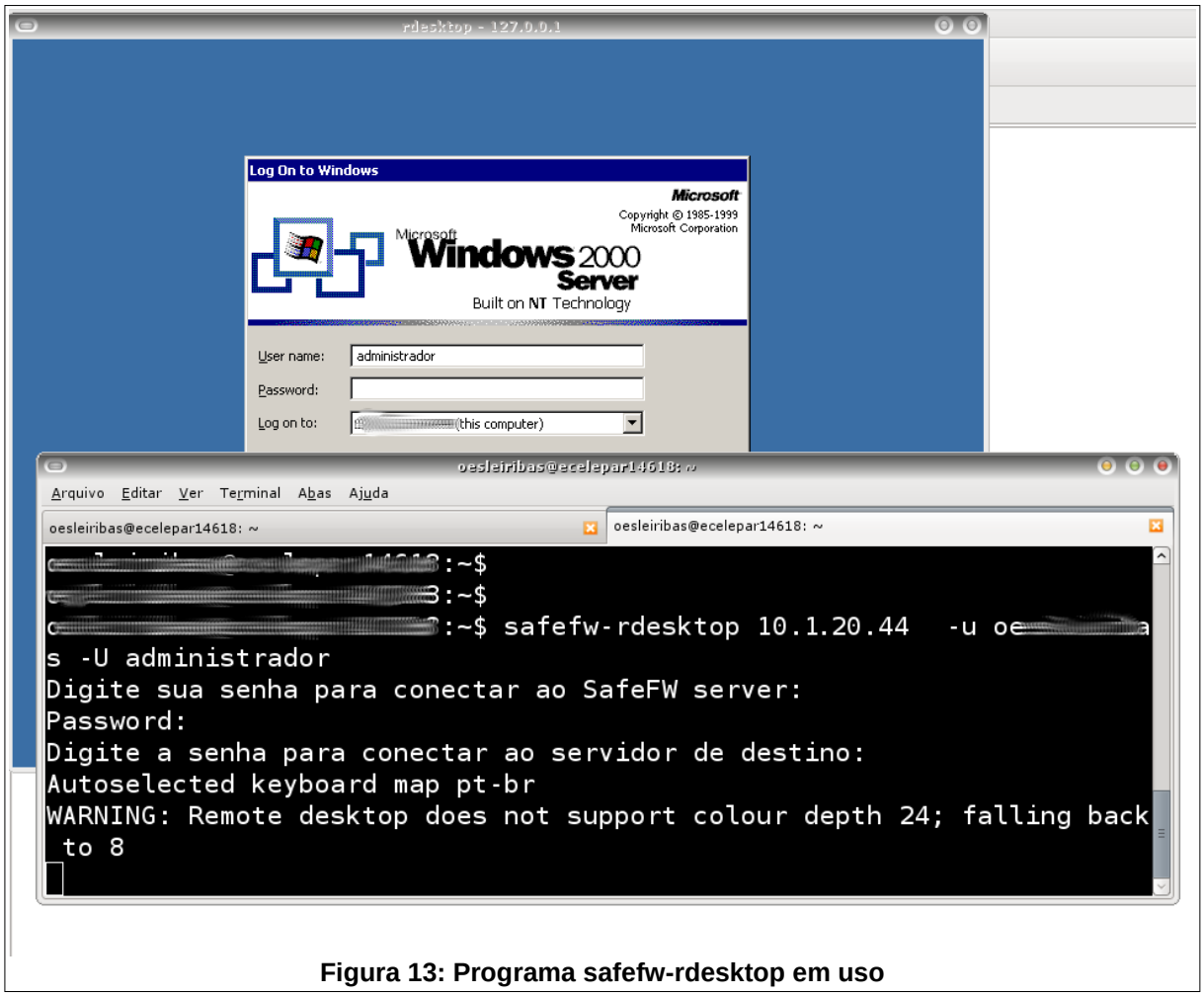


Figura 13: Programa safefw-rdesktop em uso

1.5.6 Cliente em Java

Um dos requisitos do projeto é que seja possível o acesso de plataformas heterogêneas. Entende-se como plataformas heterogêneas os diferentes sistemas operacionais utilizados pelos usuários. Para atingir esse objetivo foi construído um cliente gráfico em Java. O Java foi utilizado justamente pela questão da portabilidade. Uma vez escrita uma aplicação em Java, ela pode funcionar em qualquer plataforma que possua uma JVM (Java *Virtual Machine*). Isso significa que um mesmo código pode funcionar em sistemas com Windows, Linux ou MAC OS, sem que seja necessário recompilar o programa ou realizar qualquer outra operação do gênero.

Outro motivo que levou a construção desse novo cliente é que os demais clientes são todos os por linha de comando. Assim sendo, faltava uma opção de aplicativo gráfico. A aplicação desenvolvida vem justamente para suprir essa necessidade. Embora os clientes por linha de comando sejam extremamente úteis e desejáveis em ambientes como o Linux, em ambiente como o Windows eles não são tão comuns e nem tão populares entre os usuários.

Para o desenvolvimento da aplicação foi utilizado a biblioteca J2SSH. Essa biblioteca prove as funcionalidades relacionadas a conexões SSH, funcionalidades como: abertura de conexão, criptografia e todos os demais itens que o protocolo SSH suporta. Outra biblioteca utilizada foi a *commons-logging*. A *commons-loggins* é responsável por toda a parte de registro das atividades, desde o recebimento de mensagem até o encaminhamento para arquivos textos ou para a tela do usuário. Foi feito uso também da biblioteca cglib-nodep, para geração automática de código. Por fim foi utilizada a biblioteca xstream para tratamento de arquivos XML. A xstream realiza desde o parse de arquivos XMLs até a criação de objetos em tempo de execução, permite ainda realizar a persistência de objetos para arquivos XML. O diagrama de componentes que consta na Figura 14 explica melhor a relação entre as bibliotecas.

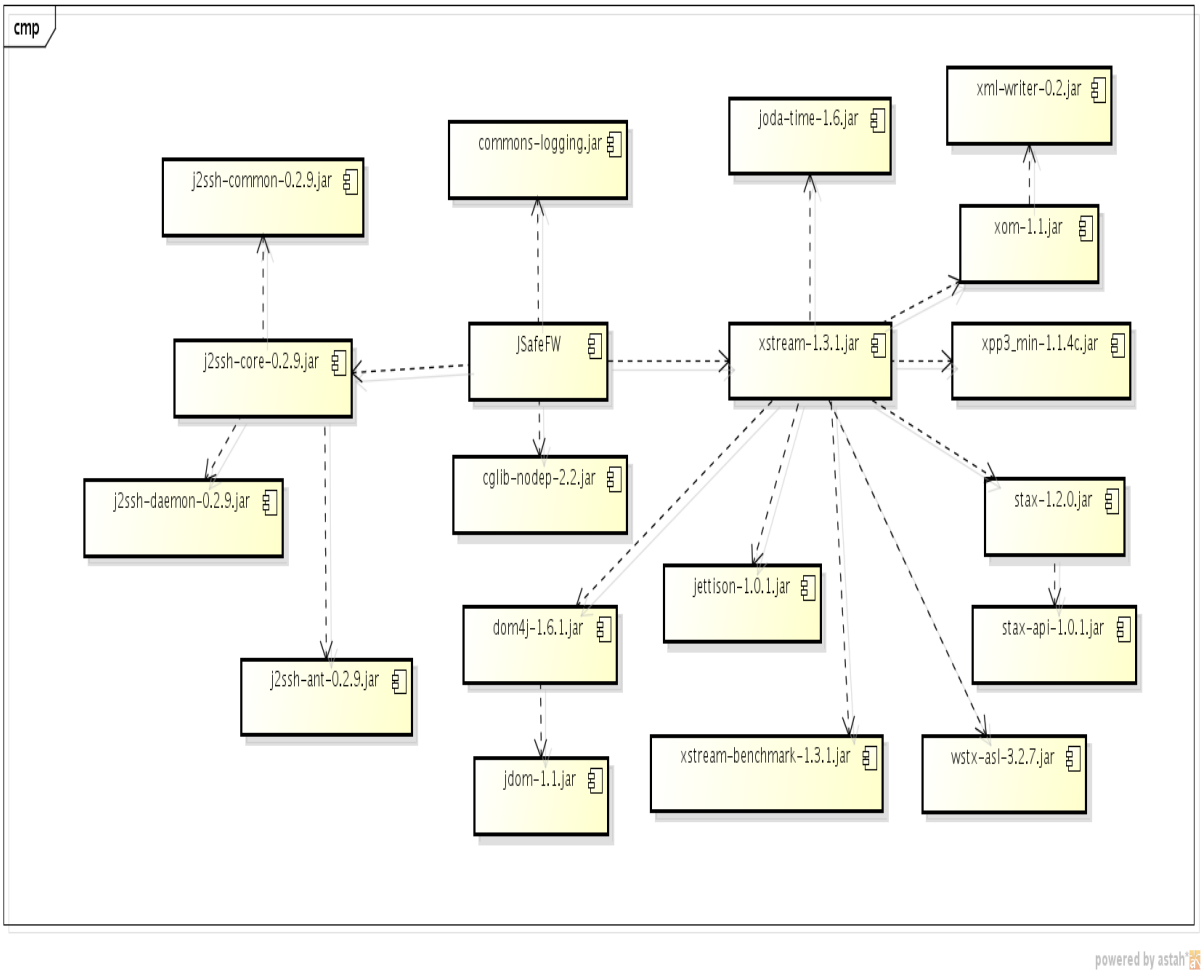


Figura 14: Diagrama de componentes

Foi dado ao aplicativo desenvolvido o nome de JSafeFW. O "J" do nome é uma referência a linguagem Java, que foi utilizada para o seu desenvolvimento. De todos os clientes que foram desenvolvidos o cliente em Java é o que possuí o maior número de funcionalidades. Caso o usuário deseje, ele pode utilizar somente esse cliente e dispensar o uso dos demais.

A Figura 15 mostra uma cópia do programa, na qual é possível visualizar as opções disponíveis para o usuário.

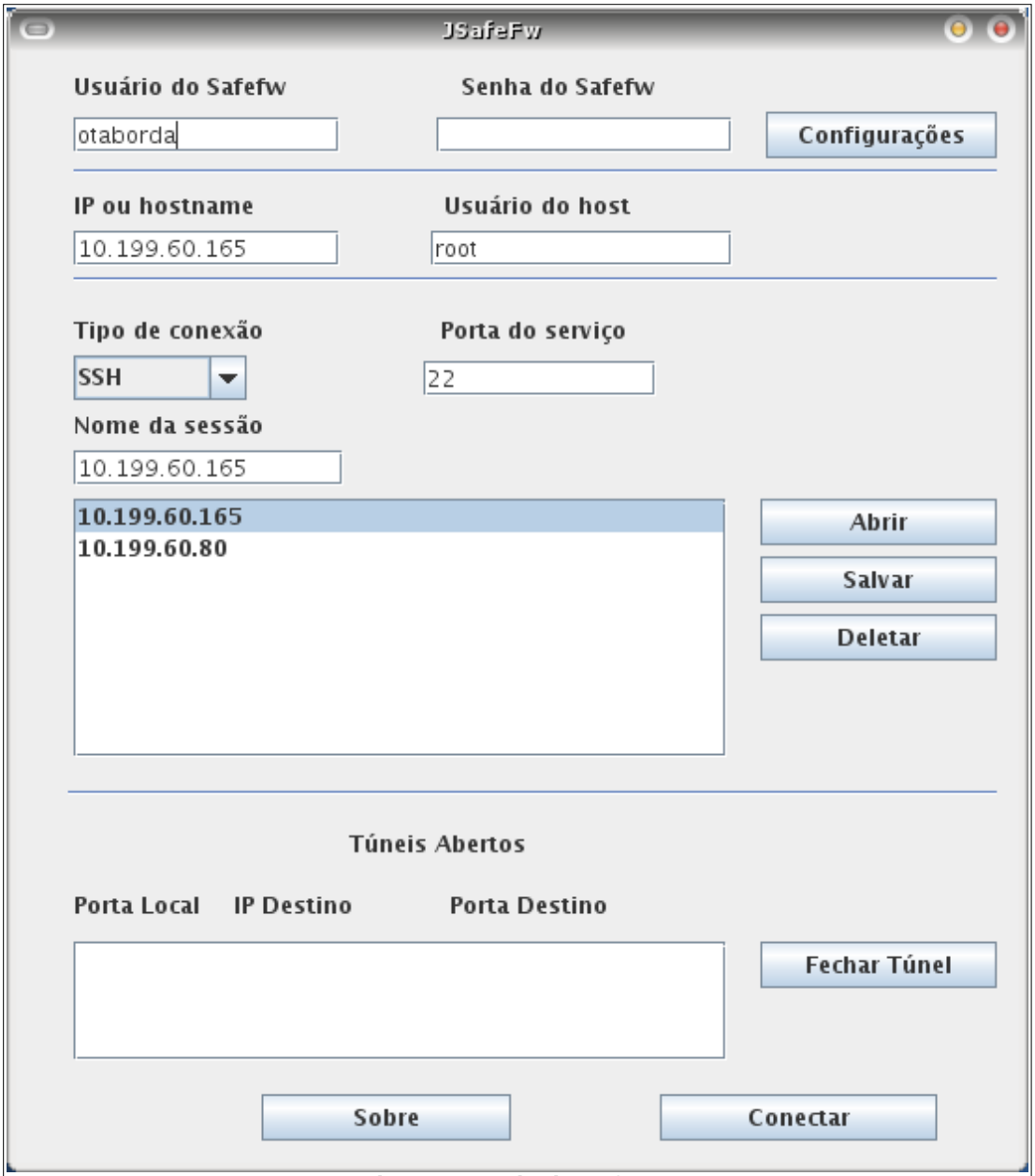


Figura 15: Tela do SafeFW

Abaixo uma descrição das principais funcionalidades oferecidas:

- Configuração do servidor SafeFW. No botão configurações é possível editar o endereço do servidor SafeFW, informando outro endereço IP ou então um nome que possa ser resolvido pelo serviço de DNS (*Domain Name System*).

- Seleção do tipo de conexão. É possível escolher entre os quatro protocolos de acesso remoto suportados pelo cliente: Telnet, SSH, RDP, RFB. Ao escolher um determinado protocolo o programa preenche automaticamente o campo porta.

- Lista de conexões abertas. Na parte de baixo do programa é exibida uma lista de conexões abertas, chamadas de "túneis abertos". O nome túneis refere-se ao túnel criptografado criado pelo programa. A lista de túneis contém as seguintes informações: porta local, IP destino e Porta Destino. Além de visualizar os dados o usuário ainda possuí a opção de fechar túnel, através do botão ao lado direito.

- Salvar os dados das sessões. Essa funcionalidade permite que o usuário salve os dados da sessão aberta, com isso é possível posteriormente recuperar esses dados e iniciar uma nova sessão. Isto é interessante, pois poupa tempo do usuário que não precisa toda a vez que for realizar um acesso remoto informar todos os dados, basta recuperá-los. Para salva uma sessão o usuário deve preencher os campos da tela, informar um nome da sessão e clicar no botão salvar. Um menu ao lado direito oferece também a opção de deletar as sessões salvas.

- Configuração automática de usuário. O nome de usuário do SafeFW é preenchido automaticamente quando o programa é iniciado. Por padrão o programa coloca o mesmo nome de usuário que está executando o sistema operacional no qual o programa está instalado. Caso o usuário deseje, ele pode mudar o nome do usuário, assim quando o programa for iniciado novamente, ele irá utilizar o novo nome de usuário.

- Persistência das configurações. Todas as configurações realizadas pelo usuário são salvas em um arquivo oculto chamado ".safefw.xml", localizado no diretório padrão do usuário. Dessa forma quando o programa é aberto novamente as configurações são recuperadas, não sendo necessário configurá-lo novamente.

A figura 16 mostra uma conexão aberta para um servidor Windows. Pode-se ver em primeiro plano o programa com os dados utilizados para a conexão e em segundo plano a conexão aberta para o servidor. Observe também que a conexão é listada em "Túneis abertos", caso o usuário deseje ele pode fechar a conexão utilizando a opção "Fechar Túnel".

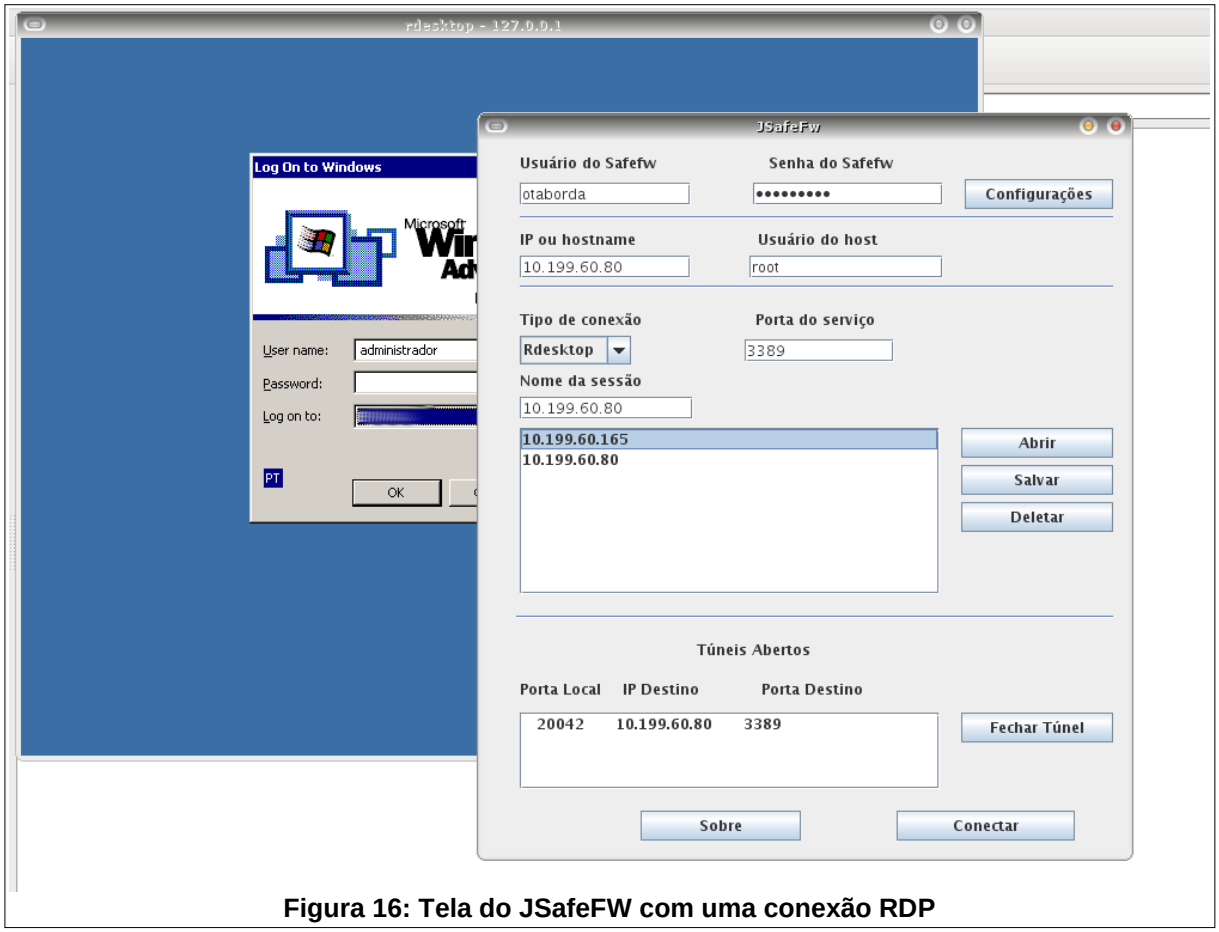


Figura 16: Tela do JSafeFW com uma conexão RDP

A figura 17 exibe uma conexão aberta para um servidor Linux. Na imagem vemos em primeiro plano o programa JSafeFW com todas as informações utilizadas para realizar o acesso e em segundo plano a conexão aberta para o servidor. Veja na imagem que a conexão foi realizada na porta 22, que é a porta padrão do protocolo SSH. Caso o servidor estivesse com o SSH em outra porta bastaria informá-la no momento da conexão, substituindo o valor padrão.

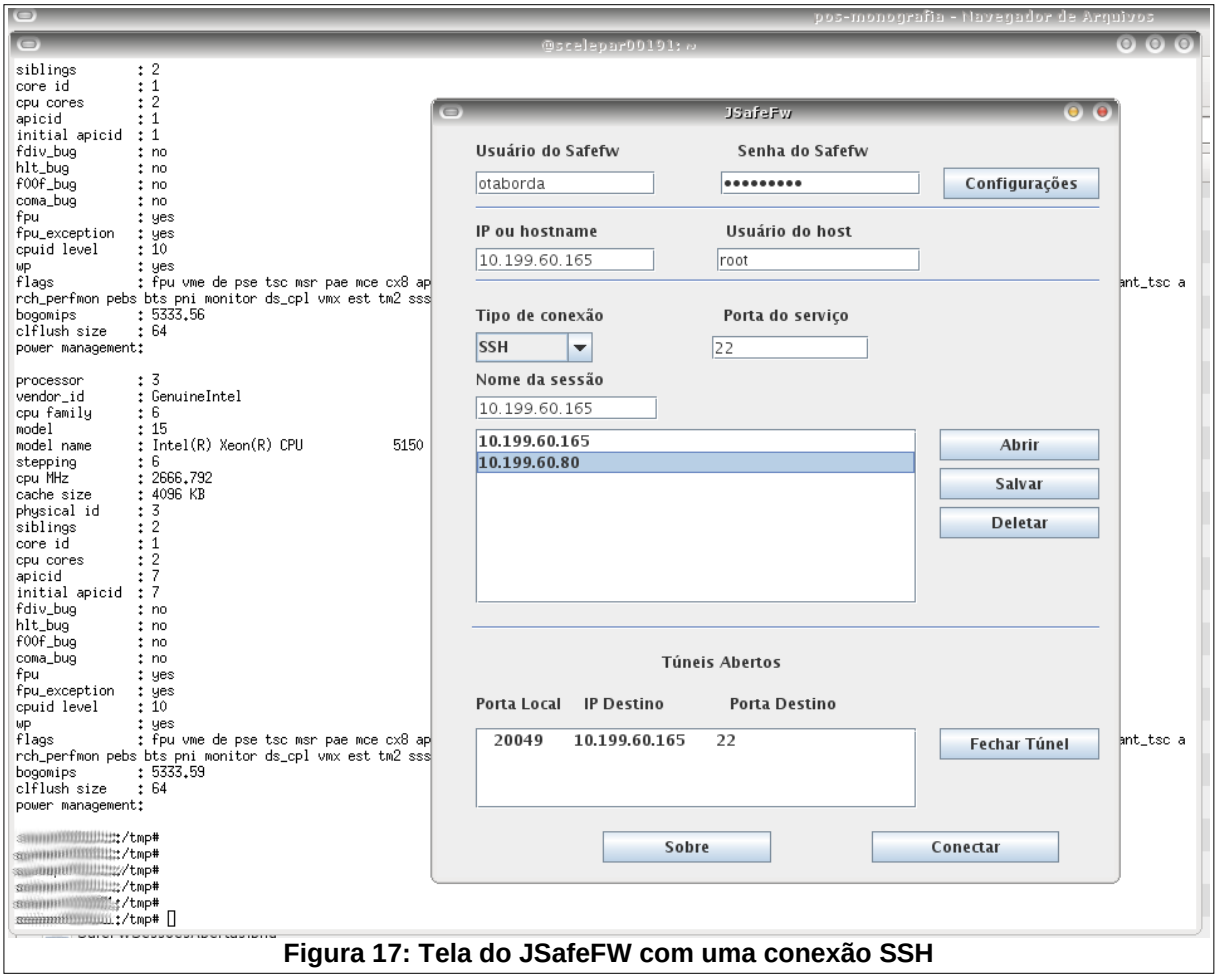


Figura 17: Tela do JSafeFW com uma conexão SSH

1.6 INTERFACES DE CONSULTAS

Uma vez que os dados sobre os acessos estejam armazenados em um banco de dados, é necessário fornecer uma interface que possibilite a consulta a esses dados. A interface deve permite que sejam consultados todos os acessos realizados por um determinado usuário, bem como todos os acessos recebidos por um equipamento.

As opções de aplicativos de consultas são inúmeras. Após uma análise decidiu-se por criar um aplicativo web. Os motivos para essa decisão foram os benefícios oferecidos pela arquitetura, tais como: flexibilidade e mobilidade. Flexibilidade em relação a atualizações, para se alterar o aplicativo basta realizar a alteração no servidor, sem necessidade de realizar alterações nos clientes.

Mobilidade em relação aos acessos, qualquer computador que possua um navegador pode acessar a aplicação, não é necessário executar modificações nos clientes.

A aplicação foi desenvolvida na plataforma Java web. Para diminuir o tempo de desenvolvimento foi utilizada uma plataforma de desenvolvimento já existente, chamada de layer8. A plataforma escolhida é utilizada por outros aplicativos de segurança da informação e oferece vários recursos. Entre os recursos oferecidos estão pool de conexão com banco de dados, sistema de registro, modelo de páginas e todo o serviço de autenticação na aplicação.

Abaixo o detalhamento das principais funcionalidades oferecidas pela interface de consulta.

Consulta simples (figura 18). Essa funcionalidade permite selecionar um servidor pelo IP e visualizar todos os acessos remotos realizados na maquina entre o intervalo de datas especificadas pela data inicial e data final.

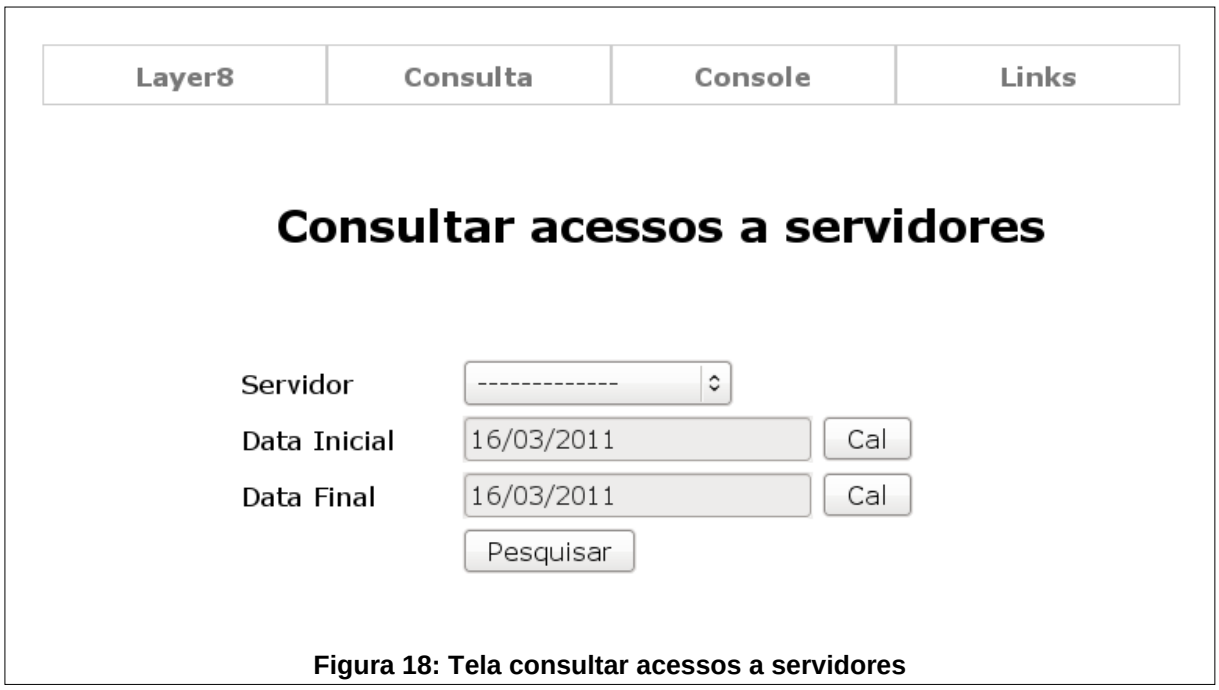


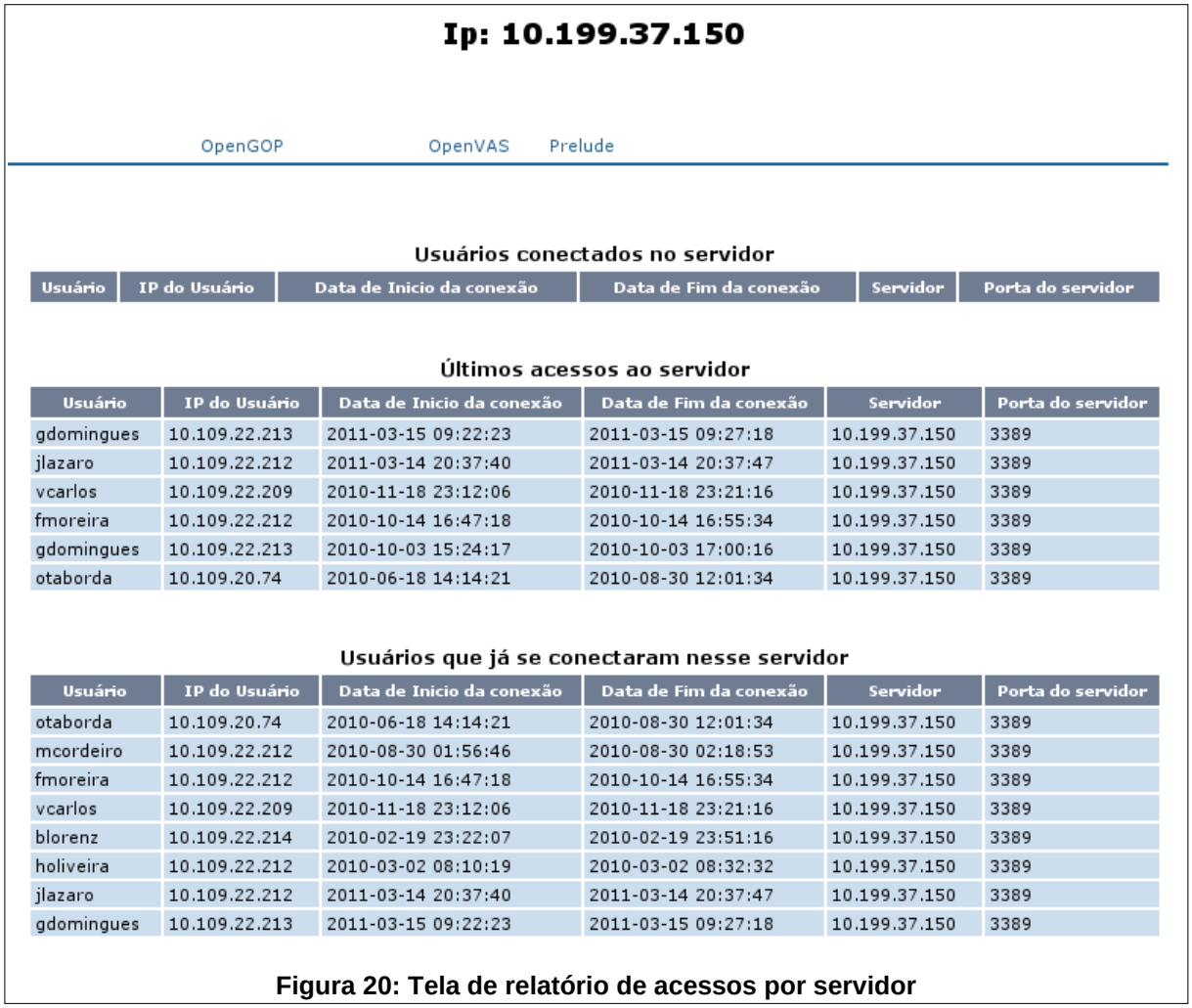
Figura 18: Tela consultar acessos a servidores

Consulta avançada (figura 19). Essa funcionalidade possibilita uma consulta mais refinada dos acessos. Além de realizar a consulta por IP e por um intervalo de datas é possível especificar outros parâmetros, tais como: Porta do servidor, node

do servidor SafeFW, IP do usuário e nome do usuário. Para realizar a consulta basta informar o intervalo de datas e outro parâmetro, como o nome do usuário, não é necessário informar todos os parâmetros.

Figura 19: Tela consulta avançada: acessos a servidores

As telas de consultas geram um relatório com várias informações. Entre as informações geradas estão o nome do usuário o IP da estação de trabalho do usuário, Data e hora de inicio da conexão, data e hora do fim da conexão e também o número da porta TCP que recebeu o acesso. O relatório é dividido em três seções. A primeira sessão informa todos os usuários que estão atualmente conectados no servidor. A segunda sessão informa a lista de usuários que realizaram os últimos acessos ao servidor. Já a última sessão informa o nome de todos os usuários que alguma vez já tenham realizado uma conexão naquele servidor. Um exemplo desse relatório pode ser observado na figura 20.



Ip: 10.199.37.150

OpenGOP OpenVAS Prelude

Usuários conectados no servidor

Usuário	IP do Usuário	Data de Inicio da conexão	Data de Fim da conexão	Servidor	Porta do servidor

Últimos acessos ao servidor

Usuário	IP do Usuário	Data de Inicio da conexão	Data de Fim da conexão	Servidor	Porta do servidor
gdomingues	10.109.22.213	2011-03-15 09:22:23	2011-03-15 09:27:18	10.199.37.150	3389
jlazaro	10.109.22.212	2011-03-14 20:37:40	2011-03-14 20:37:47	10.199.37.150	3389
vcarlos	10.109.22.209	2010-11-18 23:12:06	2010-11-18 23:21:16	10.199.37.150	3389
fmoreira	10.109.22.212	2010-10-14 16:47:18	2010-10-14 16:55:34	10.199.37.150	3389
gdomingues	10.109.22.213	2010-10-03 15:24:17	2010-10-03 17:00:16	10.199.37.150	3389
otaborda	10.109.20.74	2010-06-18 14:14:21	2010-08-30 12:01:34	10.199.37.150	3389

Usuários que já se conectaram nesse servidor

Usuário	IP do Usuário	Data de Inicio da conexão	Data de Fim da conexão	Servidor	Porta do servidor
otaborda	10.109.20.74	2010-06-18 14:14:21	2010-08-30 12:01:34	10.199.37.150	3389
mcordeiro	10.109.22.212	2010-08-30 01:56:46	2010-08-30 02:18:53	10.199.37.150	3389
fmoreira	10.109.22.212	2010-10-14 16:47:18	2010-10-14 16:55:34	10.199.37.150	3389
vcarlos	10.109.22.209	2010-11-18 23:12:06	2010-11-18 23:21:16	10.199.37.150	3389
blorenz	10.109.22.214	2010-02-19 23:22:07	2010-02-19 23:51:16	10.199.37.150	3389
holiveira	10.109.22.212	2010-03-02 08:10:19	2010-03-02 08:32:32	10.199.37.150	3389
jlazaro	10.109.22.212	2011-03-14 20:37:40	2011-03-14 20:37:47	10.199.37.150	3389
gdomingues	10.109.22.213	2011-03-15 09:22:23	2011-03-15 09:27:18	10.199.37.150	3389

Figura 20: Tela de relatório de acessos por servidor

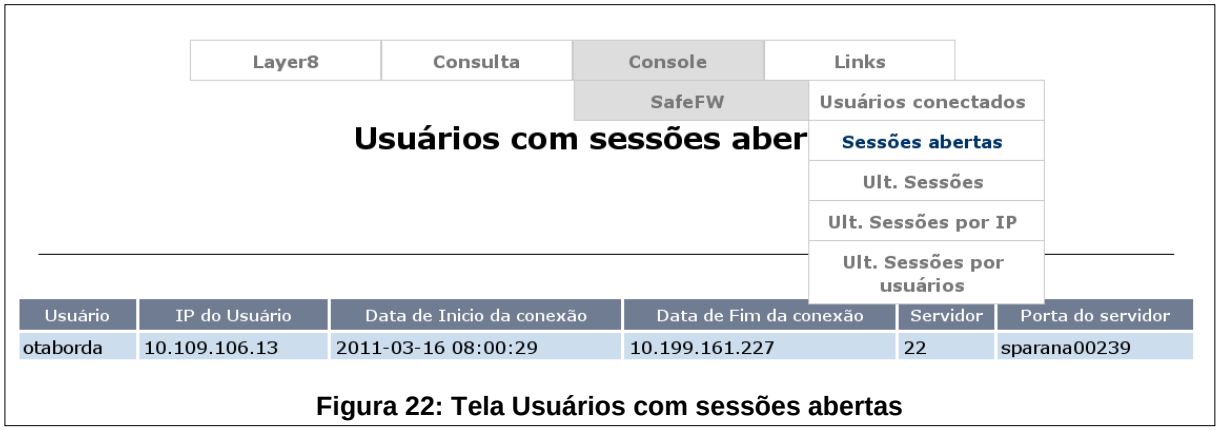
Além das telas de consultas que recebem alguns dados de entrada, existem as consoles que possuem relatórios já prontos. Ao todo existe cinco consoles. Console de "usuários conectados", essa console lista o nome de todos os usuários que estão conectados no aplicativo servidor. Console "Sessões abertas", essa console lista todos os usuários que estão com alguma sessão de acesso remoto aberta, além do nome do usuário é exibido o IP do servidor em que o usuário está conectado. Console "Ult. Sessões", essa console listas quais foram as últimas sessões de acesso remoto realizadas no ambiente. Console " Ult. Sessões por IP", essa console mostra qual foi o último acesso remoto que foi realizado em cada servidor. Console "Ult. Sessões por usuários", essa console mostra qual foi o último servidor acessado por cada usuário.

Um exemplo da console "Ult. Sessões por usuários" pode ser observado na figura 21.

Usuário	IP do Usuário	Data de Inicio da conexão	Data de Fim da conexão	Servidor	Porta do servidor
acarvalho	10.109.22.212	2010-12-02 00:25:47	2010-12-02 00:29:21	10.199.61.127	22
amandaj	10.109.22.210	2011-03-16 00:00:37	2011-03-16 00:27:43	10.199.61.3	22
blorenz	10.109.22.214	2011-03-15 13:33:05	2011-03-15 13:33:08	10.199.60.94	22
bsantos	10.109.22.210	2010-08-04 00:50:19	2010-08-04 01:19:55	10.199.60.58	22
jsilva	10.109.22.210	2010-08-04 00:50:19	2010-08-04 01:19:55	10.199.60.58	22
fsantos	10.109.18.5	2010-12-26 08:33:09	2010-12-26 08:38:51	189.199.113.107	22
fmoreira	10.109.22.212	2011-03-15 14:35:29	2011-03-15 14:46:18	189.199.113.49	22
fsumma	10.109.22.213	2010-01-20 09:18:31	2010-01-20 15:49:38	10.199.166.100	22
kfranca	10.109.22.210	2011-03-01 07:11:17	2011-03-01 08:20:54	189.199.113.206	22
gtavares	10.109.22.213	2011-03-15 14:27:03	2011-03-15 14:53:14	10.199.60.52	3389
hcarvalho	10.109.22.212	2011-03-16 08:13:16	2011-03-16 08:13:22	10.199.16.22	22
jaraujo	10.109.22.214	2011-03-04 01:18:55	2011-03-04 01:19:04	10.199.22.213	22
lcordeiro	10.109.22.212	2011-03-16 03:58:51	2011-03-16 03:58:51	10.199.0.11	3389
ljunior	10.109.22.214	2011-03-16 00:07:28	2011-03-16 00:27:01	10.199.61.3	22
lmazur	10.109.22.212	2010-06-24 14:43:47	2010-06-24 14:43:55	189.199.113.201	22
tmarcal	10.109.17.111	2011-03-03 15:39:23	2011-03-03 15:48:38	189.199.113.107	22
fmoser	10.109.22.214	2010-11-14 19:18:34	2010-11-14 19:50:39	10.199.60.5	22
otaborda	10.109.16.13	2011-03-16 08:00:29	2011-03-16 08:07:05	10.199.61.127	22
pjunqueira	10.109.22.215	2010-09-01 19:26:33	2010-09-01 19:26:39	189.199.220.71	22
rsantiago	10.109.22.210	2010-12-23 18:39:04	2010-12-23 18:39:39	10.199.22.207	22
rnascimento	10.109.22.210	2010-12-23 20:08:21	2010-12-23 20:08:28	10.199.22.207	22
rthamia	10.109.22.210	2011-03-04 16:16:46	2011-03-04 16:21:51	10.199.1.26	22
tpires	10.109.22.214	2010-10-17 03:52:33	2010-10-17 03:52:38	10.199.61.54	22
vnapo	10.109.22.210	2011-03-03 17:57:57	2011-03-03 17:58:00	10.199.60.232	22
vmiranda	10.109.22.209	2011-03-15 23:27:08	2011-03-16 00:41:37	10.199.60.38	22
wrocha	10.109.22.210	2010-11-12 07:54:38	2010-11-12 07:54:38	10.109.17.199	22

Figura 21: Tela últimas sessões finalizadas por usuários

Um exemplo da console "Sessões abertas" pode ser observado na figura 22.



Usuário	IP do Usuário	Data de Inicio da conexão	Data de Fim da conexão	Servidor	Porta do servidor
otaborda	10.109.106.13	2011-03-16 08:00:29	10.199.161.227	22	sparana00239

Figura 22: Tela Usuários com sessões abertas

1.7 RESULTADOS OBTIDOS

A solução desenvolvida foi implementada e testada em um ambiente computacional de grande porte. O ambiente em questão pertence a uma empresa brasileira de informática. Atualmente essa empresa possui mais de 1.200 funcionários e escritórios em diversas cidades do interior do Paraná. No cenário nacional é uma das referências na implantação e utilização de software livre.

O ambiente escolhido para implementação foram os *Datacenters* da empresa. O objetivo era controlar os acessos que são realizados aos equipamentos que ali estão hospedados. A empresa possui ao todo três *Datacenters*, com mais de 600 servidores hospedados, além de outros equipamentos como roteadores, *switches* e *appliances*. Para garantir a disponibilidade dos serviços, a empresa conta com um grupo de monitoramento, composto por profissionais que estão presentes 24 horas por dia 7 dias por semana. A equipe de monitoramento possui mais de 40 profissionais, garantindo que o serviço esteja disponível em qualquer hora do dia.

Para realizar a instalação da solução foram instalados dois servidores, responsáveis por realizar a autenticação dos usuários em uma base de dados LDAP da empresa e por gravar todos os acessos em um banco de dados. Foi instalado um *switch* de balanceamento de carga que distribui os pedidos de conexão entre os dois servidores. O *switch* também realiza o monitoramento de disponibilidade, caso um dos servidores fique indisponível ele envia todas as conexões para o outro servidor. Nas estações de trabalhos da equipe de monitoramento, foram instalados todos os clientes de linha de comando e o cliente gráfico.

Após realizado a instalação de toda a infra-estrutura necessária, foi avisado a todos os envolvidos sobre o projeto: usuários, gestores e responsáveis pelos equipamentos. O trabalho realizado foi de conscientização sobre a importância da ferramenta e de seus benefícios. Felizmente, obteve-se um grande índice de aceitação por parte dos envolvidos, os quais apreciaram a ideia de se ter uma aplicação na qual fosse possível consultar os acessos realizados. Para facilitar a adaptação dos usuários foram criados alias no Linux, assim quando o usuário digitava por exemplo: "rdesktop", era chamado automaticamente o aplicativo safefw-rdesktop. O mesmo procedimento foi realizado para os outros aplicativos.

Após vários meses de funcionamento, os resultados alcançados foram bem satisfatórios e acima do que se esperava.

O primeiro resultado positivo foi a estabilidade da aplicação servidora. Após realizado a instalação, não houve relatos de problemas que ocasionassem a sua interrupção. Atualmente a aplicação está funcionando a vários meses sem nenhuma reinicialização. O servidor no qual a aplicação está rodando está com um *uptime* (tempo desde a última vez em que foi reiniciado) de 280 dias. Nas logs da aplicação não consta nenhuma mensagem de erro. Observando os dados de consumo de recursos computacionais, pode se verificar que não existe nenhum problema de vazamento de memória ou de desalocação.

Outro número surpreendente foi o número de acessos realizados. Ao todo foram mais de 17 mil acessos realizados e registrados no banco de dados. Calculando uma media por mês, daria mais de mil e quatrocentos acessos. Esse número demonstra que o volume de requisições não é pequeno. Demonstra também a necessidade de uma solução que proporcione um melhor controle e um nível maior de segurança nessas comunicações.

Quanto aos aplicativos clientes, os resultados também foram positivos. Nas maquinas que foram instalados não houve problemas, tanto na instalado quanto posteriormente. Não houve também reclamações por parte dos usuários quanto a mau funcionamento ou erros. Foram realizados testes deixando conexões abertas por mais de 24 horas, nos testes as conexões continuaram persistente após esse período, não foram finalizadas. Os dados de consumo de recursos computacionais também mostram que não ocorreu um aumento de uso de memória, o que demonstra que os programas não apresentam problemas quanto a desalocação de memória.

Por fim, vale ressaltar os benefícios que foram proporcionados pela ferramenta para os profissionais de segurança da informação. Os profissionais passaram a contar com uma fonte de informação confiável sobre os acessos, permitindo a realização de auditorias e revisões de políticas de segurança. Outro beneficio foi a camada adicional de segurança proporcionada pela ferramenta, eliminando a realização de ataques de força bruta e garantindo um nível de confidencialidade maior, através do uso de criptografia.

1.8 PRINCIPAIS DIFICULDADES ENCONTRADAS

Durante a execução dos trabalhos algumas dificuldades se apresentaram principalmente de ordem técnica. Entre as principais dificuldades pode-se citar: a questão de adaptação do código fonte do openssh, integração com os protocolos de acesso remoto e escolher uma biblioteca estável do protocolo SSH para a linguagem JAVA.

Em relação ao openssh, a dificuldade encontrada foi a questão de análise do código fonte para poder fazer as alterações necessários. Devido ao programa ser escrito para diversas plataformas a análise de seu código é uma tarefa trabalhosa e complexa. Além disso, o período de compilação do código era grande, o que fazia com que após serem realizadas pequenas alterações no código fonte fosse necessário esperar longos períodos até ser possível testar a alteração realizada.

Outro problema encontrado foi a integração da ferramenta com todos os protocolos de acesso remoto. Isso pode ser explicado pela heterogeneidade dos protocolos envolvidos. Para o correto funcionamento foi necessário fazer algumas alterações pontuais nos clientes para que todos funcionassem adequadamente.

A tarefa de encontrar uma biblioteca do protocolo SSH para a linguagem Java também foi uma dificuldade. Muitas bibliotecas das que estão disponíveis são instáveis. Assim sendo levou-se um tempo até encontrar uma que estivesse em um nível de maturidade adequado. Para confirmar a estabilidade das bibliotecas foram realizadas testes em diversas condições.

Todos os problemas encontrados puderam ser resolvidos sem que fossem necessárias grandes alterações no projeto. Algo que deve ser mencionados é que a maioria dos problemas foi de ordem técnica e não em relação a arquitetura em si. O que demonstra que a arquitetura é viável e pode contribuir com o aumento da segurança em situações aonde se faz necessário a disponibilização de serviços de acesso remoto.

1.9 TRABALHOS FUTUROS

Os resultados atingidos após a implantação da solução demonstram que ela é estável e que novas funcionalidades podem ser agregadas. Dessa forma, evidencia-se que trabalhos futuros podem ser realizados visando proporcionar esses novos recursos.

Um dos recursos que pode ser adicionado é a integração com algum framework de controle de acesso. Utilizando o framework o funcionário que necessitar realizar o acesso a um equipamento envia uma solicitação de acesso informando o motivo para realizar tal ação, outra pessoa então é responsável por analisar esse pedido, podendo aprovar ou não o acesso. Com isso o funcionário não possui um acesso previamente autorizado, sendo necessário justificar a necessidade de cada acesso. Embora esse modelo possa parecer burocrático ele aumenta consideravelmente a segurança do ambiente.

Outro recurso que pode ser adicionado futuramente é um modelo que proporcione um controle mais granular dos acessos, possibilitando criar regras que permitam acesso apenas em determinado horário, em determinados dias da semana, para determinados endereços IP entre outras regras. Com essa granularidade maior pode-se evitar que acessos indesejados ocorram, garantindo que todos os acessos sejam realizados apenas por pessoas previamente autorizadas.

Por fim vislumbra-se como trabalho futuro a utilização de autenticação via *token*. O *token* é um mecanismo adicional de segurança que gera números randômicos, para acessar o sistema o usuário deve informar qual é o número que está sendo apresentado por ele, sendo que esse número muda em intervalos curtos de tempo, não podendo ser reaproveitado. Pode-se pensar em uma solução de autenticação hibrida na qual o usuário além de informar qual é o seu usuário e senha deve também informar o número gerado pelo *token*. Através dessa solução hibrida consegue-se elevar a um patamar ainda maior a segurança das conexões, pois mesmo que um invasor saiba qual é o usuário e senha ele não poderá acessar o ambiente se não estiver de posso do token.

CONCLUSÃO

Este trabalho apresentou uma solução para os problemas relacionados ao acesso remoto. A solução proposta consiste de um servidor central, responsável por registrar todos os acessos realizados e por executar a identificação e autenticação dos usuários. Para utilizar a solução foram desenvolvidos diversos clientes por linha de comando e um cliente gráfico.

A solução proposta prove vários benefícios, tais como: mitigação de ataques de força bruta, confidencialidade na comunicação entre cliente e servidor com o uso de criptografia, proporciona um nível adicional de segurança através da autenticação no servidor central, além de prover uma fonte de informações para auditoria. Contudo, possui a desvantagem de aumentar a complexidade da rede e criar custos de manutenção com o servidor central.

Foi realizada a implantação da ferramenta em uma rede de grande porte. Os resultados atingidos foram satisfatórios e acima do que se esperava. A aplicação servidor se mostrou plenamente estável, funcionando por longos períodos sem a ocorrência de problemas. Os aplicativos clientes também apresentaram um bom comportamento, sem relatos de erros. Além das ferramentas houve também um retorno positivo por parte dos profissionais de segurança da informação, os quais aprovaram o aumento da segurança do ambiente.

Um resultado interessante foi o número de acessos remoto atendidos pela ferramenta, ao todo foram mais de 17 mil acessos. Esse número demonstra a importância da ferramenta e também a sua robustez para suportar a quantidade de acessos.

Os resultados demonstram que a solução proposta atingiu seus objetivos, mostrando-se como uma forma de solucionar os problemas oriundos do uso de protocolos de acesso remoto.

REFERÊNCIAS

BARRETT, Daniel; SILVERMAN, Richard; BYRNES, Robert. SSH, The Secure Shell: The Definitive Guide. 2. ed. São Paulo: O'reilly, 2005.

Brazilian Honeypots Alliance. Disponível em: <http://www.honeypots-alliance.org.br/stats/portsum/monthly/2011/01.html> . Acesso em: 01 Fev. 2011.

CERT - Estatísticas sobre incidentes notificados ao CERT.br. Disponível em: <http://www.cert.br/stats/>. Acesso em: 01 Fev. 2011.

DEITEL, Paul J; DEITEL, HARVEY M. Java Como Programar. 1. ed. São Paulo: Bookman, 2002.

FILHO, João Eriberto Mota. Descobrindo o Linux. 1. ed. São Paulo: Novatec, 2006.

GONÇALVES, Edson. Dominando Netbeans Construa Aplicativos Java. 1. ed. São Paulo: Ciência Moderna, 2006.

GONZAGA, Jorge Luiz. Dominando o PostgreSQL. 1. ed. São Paulo: Ciência Moderna, 2007.

Governança de T.I. Disponível em: <http://www.artigos.com/artigos/sociais/administracao/governanca-de-t.i.-14002/artigo/>. Acesso em: 01 Fev. 2011.

KPMG : Fraudes preocupam corporações e movimentam indústria. Disponível em: <http://computerworld.uol.com.br/gestao/2010/07/23/fraudes-preocupam-corporacoes-e-movimentam-industria/> . Acesso em: 01 Fev. 2011.

MANZANO, José M. POSTGRESQL 8.3.0 - Interativo: Guia de orientação e desenvolvimento. 1. ed. São Paulo: Érica, 2008.

MCPHIE, David; DEITEL, Harvey M.; NIETO, Tem R. C. Perl - Como Programar. 1. ed. São Paulo: Bookman, 2002.

MORIMOTO, Carlos Eduardo. Redes e servidores linux: guia prático. 2. ed. Porto Alegre: Sul editores, 2006.

Ponemon: Research Studies and White Papers. Disponível em: <http://www.ponemon.org/data-security> . Acesso em: 01 Fev. 2011.

RECKS, Ronald P.; LOVE, Paul; TERPSTRA, John H. Segurança para Linux. 1. ed. São Paulo: Campus, 2004.

Rede Privada Virtual. Disponível em: <http://www.rnp.br/newsgen/9811/vpn.html> . Acesso em: 01 Fev. 2011.

RFC 3164: The BSD syslog Protocol. Disponível em: <http://tools.ietf.org/html/rfc3164> . Acesso em: 01 Fev. 2011.

RFC 4251: The Secure Shell (SSH) Protocol Architecture. Disponível em: <http://tools.ietf.org/html/rfc4251> . Acesso em: 01 Fev. 2011.

RFC 854: TELNET PROTOCOL SPECIFICATION. Disponível em: <http://tools.ietf.org/html/rfc854> . Acesso em: 01 Fev. 2011.

SCHILDT, Herbert. C Completo e Total. 3. ed. São Paulo: Pearson, 1996.

SIERRA, Kathy; BATES, Bert; BRIAN, Basham. Use a cabeça!: Servlets and JSP. 1. ed. São Paulo: Alta Books, 2005.

SILVA, Lino Sarlo da. Virtual private network: aprenda a construir redes privadas viruais em plataformas linux e windows. 2. ed. São Paulo: Novatec, 2005.

SOUZA, Maxuel Barbosa. Obtendo e Instalando o GNU / Debian. 1. ed. Porto Alegre: Ciência Moderna, 2010.

STALLINGS, William. Criptografia e Seguranca de Redes. 4. ed. São Paulo: Pearson, 2008.

Suporte C. Disponível em: <http://www.cprogramming.com/> . Acesso em: 01 Fev. 2011.

Suporte Debian. Disponível em: <http://www.debian.org/intro/about . Acesso em: 01 Fev. 2011.

Suporte Java. Disponível em: <http://www.java.com/pt_BR/about/> . Acesso em: 01 Fev. 2011.

Suporte JSP. Disponível em: <http://java.sun.com/products/jsp/docs.html> . Acesso em: 01 Fev. 2011.

Suporte Microsoft. Disponível em: <http://support.microsoft.com/kb/186607> . Acesso em: 01 Fev. 2011.

Suporte Netbeans. Disponível em: <http://netbeans.org/kb/index.html> . Acesso em: 01 Fev. 2011.

Suporte OpenLDAP. Disponível em: <http://www.openldap.org/> . Acesso em: 01 Fev. 2011.

Suporte OpenSSH. Disponível em: <http://www.openssh.com/> . Acesso em: 01 Fev. 2011.

Suporte Perl. Disponível em: <http://www.perl.org/learn.html> . Acesso em: 01 Fev. 2011.

Suporte Postgresql. Disponível em: <http://www.postgresql.org/about/> . Acesso em: 01 Fev. 2011.

TANENBAUM, Andrew S. Redes de Computadores . 4. ed. São Paulo: Campus, 2003.

The RFB Protocol. Disponível em: <http://www.realvnc.com/docs/rfbproto.pdf> . Acesso em: 01 Fev. 2011.

TRIGO, Clodonil H. OpenLDAP: Uma abordagem integrada. 1. ed. São Paulo: Novatec, 2007.