UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

ESCOLA DE ENGENHARIA

DEPARTAMENTO DE ELETRÔNICA

IMPLEMENTAÇÃO DE UM PONTO DE ACESSO SEGURO PARA REDES 802.11b BASEADO NO SISTEMA OPERACIONAL OPENBSD

Autor: _____

Demetrio de Souza Diaz Carrión

Orientador: _____

Prof. Luís Felipe M. De Moraes, Ph.D.

Examinador: _____

Prof. Mauros Campello Queiroz, M.Sc.

Examinador: _____

Claus Rugani Topke, M.Sc.

DEL

Abril de 2003

Dedicatória

Dedico este trabalho à minha avó Josefa Carrión Alvarez e à memoria de meu avô Demetrio Diaz Lamas, a partir dos quais a minha história de vida começa a ser contada.

Agradecimentos

Ao meu orientador, Professor Luís Felipe M. de Moraes por ter acreditado quando este projeto era incipiente e por ter fornecido todas as condições para que ele se tornasse tangível.

Ao amigo Bruno Astuto Arouche Nunes que participou na concepção e implementação do sistema isAlive, parte fundamental deste projeto.

Ao amigo Alexandre Pinaffi Andrucioli pelas várias horas dedicadas a diversas partes deste projeto, principalmente na configuração da VPN.

À amiga Flávia Leal Cavalheiro por ter elaborado as figuras presentes neste projeto.

À minha namorada Juliana Santos da Rocha por ... depois a gente conversa.

Resumo

Este projeto final consiste na implementação de um ponto de acesso, denominado WStrike, para redes 802.11b onde a segurança é o principal foco. Neste contexto, utilizou-se o sistema operacional OpenBSD e outras ferramentas de segurança como redes privadas virtuais, um sistema de controle de acesso à rede sem fio, o sistema StrikeIN e um sistema de verificação de atividade das estações na rede, o sistema isAlive.

Palavras-chave

Redes sem fio
Segurança
Criptografia
Controle acesso
Procolo de segurança

Índice

Lista de Figuras

Lista de Tabelas

Abreviaturas e Siglas

AP	Access Point
AH	Authentication Header
BD	Banco de Dados
BSS	Basic Service Set
CA	Certification Authority
DHCP	Dynamic Host Configuration Protocol
DSSS	Direct Sequence Spread Spectrum
ESP	Encapsulating Security Payload
FHSS	Frequency Hopping Spread Spectrum
FTP	File Transfer Protocol
ICP	Infraestrutura de Chave Pública
IPSec	Internet Protocol Security
ISAKMP	Internet Security Association and Key Management Protocol
NAT	Network Address Translation
NIC	Network Interface Card
NFS	Network File System
pf	Packet Filter
PKI	Public Key Infrastructure
SGBD	Sistema Gerenciador de Banco de Dados
SSID	Service Set Identifier
STA	Estação
Wi-Fi	Wireless Fidelity
VPN	Virtual Private Network
WEP	Wired Equivalent Privacy
WLAN	Wireless Local Area Network
HTTPS	Hypertext Transfer Protocol Secure

1. Introdução

1.1. Motivação

As redes de computadores permeiam grande parte das atividades diárias de uma pessoa e representam a grande mudança no estilo de vida e na tecnologia do final do século XX.

A existência de uma rede local em uma empresa é fator comum para o aumento de suas possibilidades de sucesso, pois contribue com a diminuição de custos e aumento de produtividade, devido ao aumento da capacidade de comunicação entre funcionários, fornecedores e clientes. Tudo isto leva a um expressivo aumento nos lucros e na taxa de crescimento da empresa.

As soluções de redes sem fio em ambientes locais representam o novo estágio de integração de usuários e os serviços de diversas redes de computadores. Algumas tecnologias surgiram neste contexto, tais como: Bluetooth, 802.11b (Wi-Fi), HomeRF, entre outras.

O padrão de redes sem fio adotado para o desenvolvimento deste projeto final foi o IEEE 802.11b, que correponde a um padrão de redes locais sem fio atuando na faixa de 2.4 Ghz, utilizando Espalhamento de Espectro por Sequência Direta (DSSS - Direct Sequence Spread Spectrum) ou Espalhamento de Espectro por Saltos em Frequência (FHSS – Frequency Hopping Spread Spectrum) e suportando taxas de transmissão de até 11 Mb/s[1].

1.2. Oportunidades observadas

i. A necessidade de definição de métodos de interligação de uma rede sem fio a uma rede local cabeada, provendo uma estrutura de autenticação, certificação, distribuição de chaves, assim como um grande interesse de se implementar uma solução de forma rápida e segura foram as motivações deste projeto.

ii. O alto custo de um ponto de acesso se torna um empecilho na configuração de uma rede local sem fio, sendo interessante, portanto, implementar-se uma solução de

[1] O padrão especifica a utilização de tecnologias de infra-vermelho e atuação na banda de 5 Ghz , mas não serão características aplicadas neste projeto.

baixo custo e que garante ao usuário um controle rígido sobre as características de segurança do AP (*Access Point* ou Ponto de Acesso) e de sua rede como um todo.

iii. Várias propostas com relação a autenticação dos usuários móveis foram definidas nos últimos anos, como o padrão 802.1x, DHCP seguro e regras de firewall modificadas dinamicamente. A implementação de um sistema de autenticação eficiente é fundamental na segurança de redes sem fio.

iv. O WEP (Wired Equivalent Privacy) foi definido como protocolo de autenticação, autorização e criptografia das transações eletrônicas em redes 802.11b, no entanto várias falhas com relação ao protocolo WEP foram encontradas. Outras soluções foram propostas como a utilização do 802.1x e IPSEC, mas ainda são incipientes, carecem de uma metodoliga para aplicação e por vezes não são soluções de código aberto. Torna-se fundamental a procura por soluções que sejam eficazes na segurança das transações em uma WLAN.

1.3. Objetivos

O objetivo deste projeto pode ser dividido em 4 partes:

1ª) Desenvolver uma distribuição do sistema operacional OpenBSD, denominada WStrike, que capacite um usuário a configurar de forma simples e rápida um computador como Ponto de Acesso em uma rede local.

2ª) Gerar um conjunto de sugestões práticas que contemple a segurança da interconexão de redes Wi-Fi em ambientes com infraestrutura cabeada pré-existente.

3ª) Implementar um sistema de autenticação de usuários baseado em certificados digitais e credenciais fornecidas por login e senha

4ª) Configurar uma VPN entre as estações da rede sem fio e o ponto de acesso.

1.3.1. Ponto de acesso (AP - *Access Point*)

A definição de um AP de acordo com o padrão IEEE 802.11b é a seguinte:

> "Um ponto de acesso é uma entidade que possui funcionalidades de uma estação e que provê acesso aos serviços de distribuição, através do meio sem fio às estações associadas[2]."

Isto implica que as estações (*notebooks, PDAs, desktops* ...) devem estabelecer uma conexão com um AP de forma a utilizar os serviços da rede e se comunicar com outras estações. Os serviços da rede podem ser acesso à Internet, acesso a arquivos (NFS), ftp etc.

Portanto, esta primeira parte será composta pelo sequintes itens:

- tornar um computador *desktop* em um AP
- desenvolver uma distribuição simplificada do OpenBSD configurado como AP Wi-Fi

1.3.2. Proposta de interconexão segura

- procedimentos e medidas que sugerem uma forma adequada e segura de inserção de uma rede sem fio em ambiente de rede local cabeado
- inventário de equipamentos necessários na implantação de uma rede sem fio
- inventário de sistemas necessários na implantação de uma rede sem fio
- descrição da infraestrutura necessária para autenticação e certificação digital

1.3.3. StrikeIn

- desenvolver script que modifique as regras de firewall dinamicamente
- desenvolver métodos para verificar se um cliente está ativo na rede
- configurar servidor web que requisite certificados digitais do cliente e que forneça o seu certificado digital
- Página web de autenticação
- Banco de dados que contém as credenciais dos clientes

[2] IEEE 802.11b, 1999, Capítulo 3, pg 3

1.3.4. VPN

- Configurar o daemon ISAKMPD no servidor para estabelecimento da VPN
- Configurar SSH Sentinel nos clientes para utilização da VPN

2. O Padrão 802.11b

O padrão 802.11b lançado em 1997 pelo IEEE representa um conjunto de especificações para implementação de redes locais sem fio que prevê três técnicas de transmissão, sendo elas: Infra-vermelho, DSSS e FHSS. Os dois últimos métodos utilizam transmissão de ondas curtas de rádio compreendidas pela banda ISM de 2.4GHz. Dentre os métodos mencionados, o DSSS é o mais utilizado na implantação de 802.11b.

O FHSS utiliza 79 canais com largura de 1 MHz cada. Um gerador de números pseudo-aleatório é utilizado para gerar a seqüência de saltos nos 79 canais, desta forma todas as estações que tenham utilizado a mesma semente em seu gerador e que se mantenham sincronizadas, saltarão para os mesmos canais simultaneamente. Cada estação de uma mesma rede que utiliza a mesma seqüência de saltos ficará em cada canal por um período denominado *dwell time*, que é ajustável.

Com o FHSS tem-se um sistema robusto contra ruído de banda estreita que provê um nível de segurança já na camada física, pois somente as estações que conhecem a seqüência de saltos e o *dwell time* poderão escutar o meio de maneira adequada e organizada. No entanto o FHSS possui a desvantagem de oferecer uma baixa largura de banda.

O DSSS espalha o espectro de freqüência de um sinal de banda estreita através de sua modulação com uma seqüência de bits denominada de *chip sequence*. Obtem-se desta forma um sistema robusto contra ruído de banda estreita ao preço de se necessitar de uma controle de potência para transmissão. Com o DSSS foi possível estender a taxa de transmissão do 802.11b de 1 Mb/s para 11 Mb/s.

2.1. WEP (Wired Equivalent Privacy):

O protocolo WEP está definido no padrão 802.11b como protocolo de segurança para as transações eletrônicas no ambiente de redes sem fio. Diversas falhas de segurança foram encontradas no protocolo WEP como descritas em [Walker 2000], [Borisov 2001] e [Fluhrer 2001]. Neste contexto, foram propostas a utilização do padrão 802.1x, de Redes Virtuais Privadas (Virtual Private Networks - VPN) e do DHCP seguro como alternativas à fragilidade do WEP [Casole 2001].

3. Dos sistemas e equipamentos utilizados

A utilização de computadores como APs é uma idéia relativamente nova que a partir de 2002 começou a tomar um grande vulto nos grupos de estudo e patrocínio de WLANs.

Em uma infraestrutura padrão de redes sem fio estruturada, têm-se dois grupos de dispositivos: as estações e o AP[3].

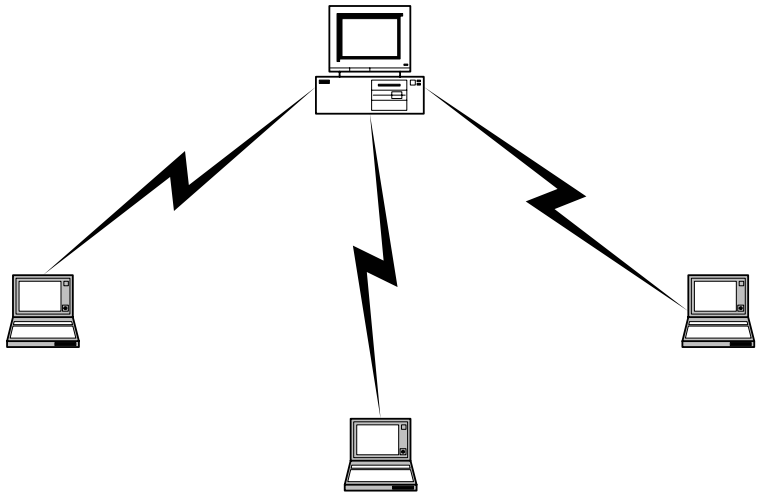

Figura 1 - Arquitetura básica de uma WLAN 802.11b infraestruturada

3.1. AP

O AP representado na figura é um PC padrão i386 que contém uma placa de rede sem fio Wi-Fi e um sistema operacional que possua ferramentas com características que o capacite a exercer funções de um AP.

Um AP possui funções especiais de sincronismo, estabelecimento de uma rede (BSS) etc, logo estas capacidades devem estar implementadas no chipset da placa de rede e algumas outras funções implementadas pelo driver da placa suportado pelo sistema operacional.

[3] A arquitetura proposta por este projeto inclui a implementação de um servidor de banco de dados que centralize todas as contas e senhas dos usuários credenciados a acessar o AP.

3.1.1. Microcomputador

Para a configuração do AP foi utilizado um microcomputador com as seguintes configurações de hardware:

Microcomputador	Pentium III
Clock	500 Mhz
RAM	128 Mb
Disco Rígido	20 Gbytes
Placas de Rede	1 Placa de rede Ethernet
	1 Placa de rede 802.11b

Tabela 1 - Configurações de hardware do AP

3.1.2. Sistema Operacional

Foi escolhido um sistema operacinal que requer pouco poder de processamento por parte da CPU, que suporta placas de rede sem fio em modo de ponto de acesso e oferece características de segurança adequadas para um ambiente permissivo como o de redes sem fio.

As primeiras implementações de um AP utilizando PCs recaíram sobre o Linux, que é um sistema operacional com bom desempenho, mesmo em máquinas com pouco poder de processamento, de código aberto e gratuito, desta forma facilitando o desenvolvimento e implementação de projetos em redes sem fio.

O Linux possui um grande número de desenvolvedores voluntários ao redor do mundo e possui suporte a diversos tipos de hardware, provendo desta forma suporte para uma vasta gama de modelos e marcas de placas de redes fio.

O Linux não possui, de forma geral, um perfil de desenvolvimento e evolução voltado para a segurança dos seus sistemas. A intenção dos desenvolvedores é de se expandir o suporte do Linux nos mais variados *hardwares* e permitir a fácil integração de periféricos no sistema.

Um projeto de redes sem fio deve ter como uma de suas premissas a segurança, já que, a transmissão de dados é feito em um meio altamente permissível, e a atuação de hackers e crackers fica facilitada, haja vista a falta de limites físicos no meio de transmissão dos dados[4].

A utilização do sistemas operacionais UNIX mostra-se bastante favorável no quesito desempenho, código aberto e farta documentação, mas dentre os diversos tipos de UNIX optou-se pelo OpenBSD [OpenBSD 2003], pois é um sistema operacional cujos principais esforços enfatizam a portabilidade, padronização, correção, segurança proativa e criptografia integrada.

O sistema é reconhecido como um dos mais seguros [Vaughan-Nichols 2001], [Dyck 2002] e possui uma instalação rápida, somente o que é estritamente necessário é instalado no sistema, facilitando o gerenciamento e diminuindo as falhas de segurança provocadas pelo desconhecimento da instalação de diversos programas, fato bastante comum em sistemas Windows e Linux hoje em dia.

O OpenBSD conta com um firewall[5], o pf (*packet filter*) e NAT (*Network Address Translation*) prontos para serem usados em sua instalação padrão.

O pf permite que seja implementado sistemas de autenticação no AP de acordo com a mudança sob demanda das regras de acordo com o sucesso do processo de autenticação de um usuário.

O NAT permite que redes internas de uma organização utilizem endereços não-roteáveis, definidos como endereços inválidos, de forma que o gateway faça uma tradução destes endereços para 1 (um) ou mais endereços roteáveis. Conforme a RFC 3022, buscava-se um paliativo para a escassez dos endereços IPv4, permitindo que organizações com um número limitado de endereços IPs pudesse ter um número consideravelmente maior de máquinas em sua rede interna com acesso à Internet. Deve-se notar que esta solução implica em custo adicional na manutenção dos estados das conexões pelos roteadores ou dispositivos de NAT.

[4] Obviamente o AP conta com uma área de abrangência limitada pela antena utilizada para a transmissão de rádio freqüência, no entanto um inimigo poderia sobrepujar esta limitação utilizando-se de uma antena com alto ganho presente na sua estação e desta forma acessar a rede de uma distância muito maior.
[5] Na referência [Hartmeier 2001] tem-se um comparativo de performance entre o iptables, pf e ipf suportados respectivamene pelo, Linux, OpenBSD e FreeBSD

Esta solução paliativa para escassez dos endereços IPs mostrou-se bastante eficaz no obscurecimento da topologia interna de uma rede para um possível inimigo. Deve-se notar que um usuário externo à rede não é capaz de descobrir a topologia interna de uma entidade através da simples observação do tráfego gerado pelas máquinas da rede interna, pois toda a comunicação terá como endereço de origem um conjunto restrito de IPs válidos ou roteáveis presentes na interface do gateway.

Nota-se ainda que o acesso direto às máquinas internas fica extremamente dificultado para usuários externos, pois a princípio o roteador não encaminhará endereços inválidos provenientes da Internet.

3.1.3. Placa de rede 802.11b

No manual do driver wi, para placas de rede sem fio 802.11b do OpenBSD, [MAN 2002] estão listadas as placas de rede sem fio que são suportadas, podendo ser visualizadas no apêndice A.

A placa de rede sem fio a ser utilizada no computador PC deve ter barramento PCI e suportar modo de operação de ponto de acesso. Somente placas com chipset Prism II eram suportadas como ponto de acesso no início deste projeto[6].

Foi possível habilitar o WEP 128 bits no AP e através de uma atualização de driver especialmente requerida do site da 3Com, podendo-se habilitar esta propriedade em placas 802.11b PCI. Ressalta-se, no entanto, que não foi possível estabelecer uma comunicação eficaz entre o AP com WEP 128 bits e microcomputadores equipados com placas 802.11b da 3Com(3CRWE737-E1) com WEP 128 bits.

3.1.4. VPN (*Virtual Private Network*)

Todo tráfego da rede sem fio deve passar por uma rede privada virtual (VPN), configurada através do protocolo IPSec [IPSec 2003], que implementa a privacidade virtual da rede, além

[6] Placas com chipset Prims 2.5 podem funcionar em modo AP no OpenBSD 3.2

de agregar segurança nas transações que atravessam o túnel estabelecido por ela. A seção 4.1 descreve mais detalhadamente uma VPN segura.

O ISAKMPD (Internet Security Association and Key Management Protocol Daemon) é um daemon presente na distribuição do OpenBSD utilizado no estabelecimento de canais seguros de comunicação, através de tráfegos autenticados e/ou encriptados, isto significa tráfego IPSec (IP Security).

O servidor utiliza o ISAKMPD para o estabelecimento do túnel IPSec através da configuração de dois arquivos: isakmpd.conf e o isakmpd.policy.

3.1.5. Servidor WEB

Foi utilizado o servidor Apache versão 1.3.27, com suporte a SSL (mod_ssl 2.8.12 e openssl 0.9.6h) e PHP (php 4.2.3).

Optou-se pelo Apache já que é um software de código aberto, gratuito, utilizado em 60% dos servidores web da internet [NetCraft 2003], possui ótima documentação e apresenta uma resposta eficaz (1 a 2 dias) no desenvolvimento de correções de segurança.

A utilização do mod_ssl como módulo de criptografia para o Apache se deve a fatores bastante semelhantes, como: ser software aberto, gratuito, farta documentação, resposta eficaz para correções de segurança, além de se basear na biblioteca openssl, largamente testada, utilizada e atualizada em diversos softwares de código aberto.

O PHP é uma linguagem de script para páginas web com sintaxe bastante semelhante ao C. O site do PHP possui vasta documentação, o tempo de desenvolvimento e correções de segurança é bastante curto e apresenta uma ótima curva de aprendizado.

3.2. Servidor de Banco de Dados (BD)

3.2.1. Microcomputador

Para a instalação do BD foi utilizado um microcomputador com as seguintes configurações de hardware:

Microcomputador	Pentium IV
Clock	2 GHz
RAM	512 Mb
Disco Rígido	4 Gbytes
Placa de Rede	1 Placa de rede Ethernet

Tabela 2 - Configurações de hardware do microcomputador que contém o BD

3.2.1. Sistema Operacional

OpenBSD 3.2.

3.2.2. Sistema Gerenciador de Banco de Dados (SGBD)

O sistema de autenticação dos usuários para utilização dos recursos oferecidos pelo ponto de acesso utilizam-se de um banco de dados implementado em MySQL.

O MySQL é um SGBD de código aberto que utiliza o paradigma de banco de dados relacional, cuja configuração é bastante simples e que corresponde às demandas de sistemas de produção de média escala [MySQL 02].

3.3. Estações Clientes

Os testes do WStrike e do StrikeIN foram feitos utilizando-se como clientes notebooks com placas de rede da 3COM 3CRW737E-96B e Orinoco, sem habilitação do WEP e estações desktops equipadas com placas de rede 3COM 3CRWE777A sem WEP.

Para a formação do túnel IPSEC é necessário que o cliente apresente um software cliente que encapsule o tráfego que passa através de sua interface de rede. Neste projeto foi utilizado o SSH Sentinel devido a sua fácil instalação e configuração, além de uma larga documentação e suporte no site da empresa SSH (www.ssh.com).

3.3.1. Notebook 1

Notebook	Pentium II
Clock	500 Mhz
RAM	128 Mb
Disco Rígido	15 Gb
Placas de Rede	3CRW737E-96B
Sistema Operacional	Windows 98

Tabela 3 - Configurações de hardware do notebook 1

3.3.2. Notebook 2

Notebook	Pentium II
Clock	1.8 GHz
RAM	256 Mb
Disco Rígido	30 Gb
Placas de Rede	3CRW737E-96B
	Orinoco
Sistema Operacional	Windows XP

Tabela 4 - Configurações de hardware do notebook 2

3.3.3. Estação 1

Estação 1	Pentium II
Clock	233 Mhz
RAM	128 Mb
Disco Rígido	20 Gbytes
Placas de Rede	3CRWE777A
Sistema Operacional	Windows 2000 / XP

Tabela 5 - Configurações de hardware da estação 1

3.3.4. Estação 2

Estação 2	Pentium
Clock	100 Mhz
RAM	128 Mb
Disco Rígido	4 Gbytes
Placas de Rede	3CRWE777A
Sistema Operacional	Windows 98

Tabela 6 - Configurações de hardware da estação 2

4. Arquitetura de Segurança Proposta

Diversas medidas de segurança são adotadas na configuração do AP WStrike. Algumas considerações de segurança são relativas à própria configuração segura de um sistema operacional e outras são relativas aos mecanismos de autenticação, autorização, confidencialidade e integridade do fluxo de informações presentes na rede local sem fio gerenciada pelo AP WStrike.

Está descrito abaixo o processo pelo qual um cliente ou estação (STA) passa desde o momento em que deseja se associar ao AP até o fechamento da conexão.

i. STA procura uma rede no domínio definido pela SSID7
ii. STA e AP se sincronizam e a associação é estabelecida
iii. STA requisita um IP (cliente DHCP)
iv. AP fornece um IP à STA (servidor DHCP)
v. STA envia mensagens UDP para formação do túnel VPN
vi. AP verifica credenciais do usuário móvel, formando o túnel VPN
vii. Usuário da STA acessa página web a fim de se autenticar
viii. O AP requisita o certificado digital do usuário da STA
ix. O usuário da STA apresenta o seu certificado digital
x. O AP apresenta o seu certificado digital à STA
xi. O AP acrescenta o IP da STA no banco de dados de IPs autorizadas
xii. O AP reconfigura as regras de firewall
xiii. A STA está pronta para utilizar os recursos de rede oferecidos através do AP
xiv. Durante o período de conexão, o AP verifica se a STA continua ativa, a fim de que possa controlar de forma adequada as regras do firewall (implementado do sistema isAlive descrito no item 3.1.4)

A Figura 2 contém um diagrama de rede corresponde a uma implementação padrão do Wstrike e nela constam:

a) AP – Contém o firewall, NAT, servidor WEB, servidor DHCP, daemon IsAlive

[7] Service Set Identification

b) BD & CA – Contém o SGBD MySQL e a chave privada da Autoridade Certificadora. Estão descritos na figura como servidor de autenticação.
c) Estações (STAs) – Contém clientes SSH Sentinel e o cliente isAlive
d) Servidor de Arquivos – Representa um servidor SAMBA. Não faz parte da implementação do WStrike, mas exemplifica um funcionamento básico de redes que se utilizam de autenticação em domínios.

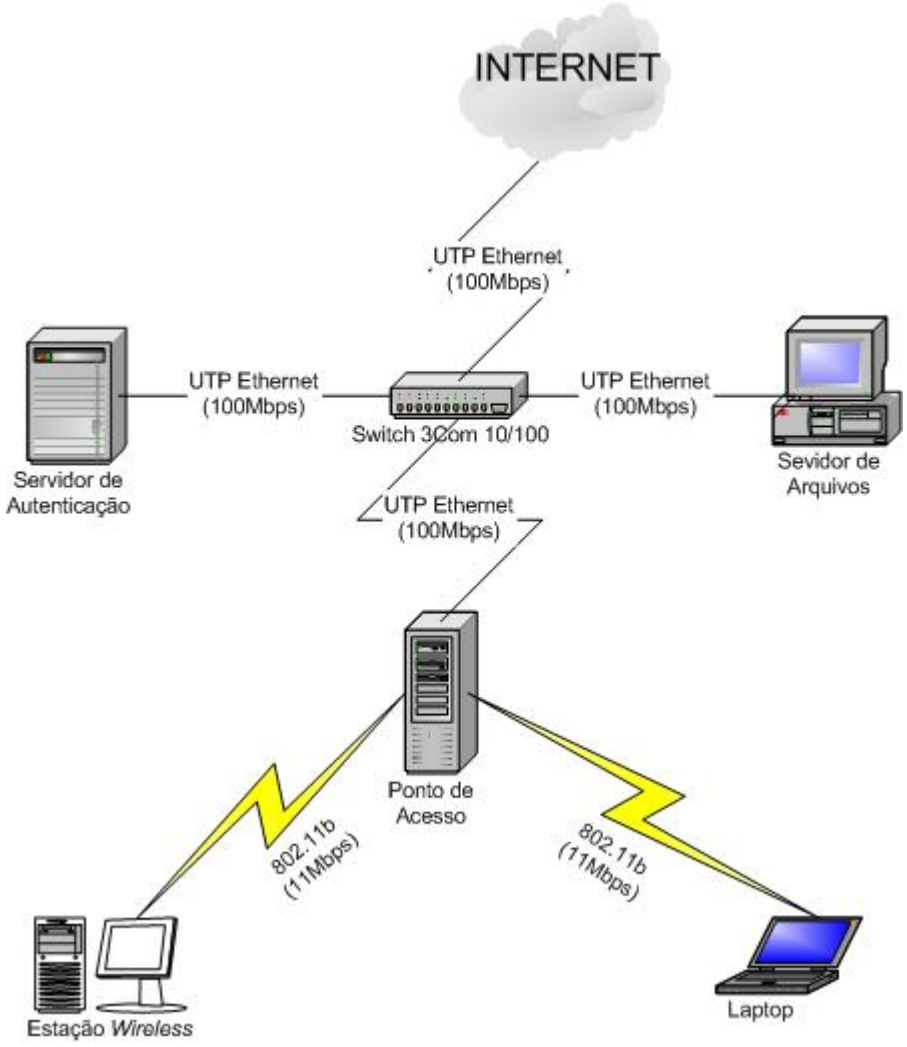

Figura 2 - Diagrama da Rede WStrike

4.1. VPN

O sigilo do tráfego da rede, a autenticação inicial dos usuários, a integridade das mensagens da rede sem fio serão garantidos através da implementação de uma VPN segura, para tanto utiliza-se do protocolo IPSec a fim de garantir a privacidade virtual da rede e a segurança das eletrôncias que por ela passam .

Nesta arquitetura todo o tráfego entre as estações e o AP é encriptado independente do destino dos pacotes enviados pelas estações. A VPN poderia ser configurada de forma que somente alguns pacotes com endereços de destino definidos fossem encriptados, sendo esta uma implementação bastante comum onde estão presentes dois gateways de duas redes diferentes.

Para a configuração da VPN utiliza-se o IPSec que é uma proposta de segurança mandatória para o IPv6 e que está sendo adotada para redes IPv4. Diversas RFCs definem a arquitetura proposta pelo IPSec, estando presentes os métodos de criptografia, de hash, DOI (Domain of Interpretation) etc. A RFC 2411, estrutura a documentação do IPSec definida pelo IEEE.

O IPSec estabelece entre dois pontos uma SA (Security Association ou Associação de Segurança) que é um canal simplex, sendo então necessário para uma comunicação bidirecional o estabelecimento de duas SAs.

Para o estabelecimento de um túnel VPN e as SAs correspondentes necessita-se de um procedimento de troca de chaves, que serão utilizadas para a autenticação inicial das duas entidades, podendo ser dado pela apresentação de credenciais como uma senha compartilhada ou troca de certificados digitais, para a negociação dos algoritmos simétricos utilizados para a troca de mensagens encriptadas, os algoritmos de hash para garantir a integridade das mensagens e todo o processo de reautenticação.

O estabelecimento de um túnel IPSec pode seguir dois modos: transporte e túnel. No modo túnel todo o pacote IP é protegido e todas as mensagens provenientes das estações saem com o endereço do AP como endereço de destino. Garante-se com esta última característica uma privacidade maior para os usuários, dificultando a análise passiva do tráfego da rede.

Dentre estas escolher pode-se optar pelos protocolos ESP (Encapsulating Security Payload) e o AH (Authentication Header). No ESP tem-se a proteção de todo o payload do pacote IP e no AH tem-se a proteção do cabeçalho IP. Na Tabela 7 baseada em [Stalings 1999] pode-se verificar a proteção oferecida em cada caso:

	SA em modo Transporte	SA em modo Túnel
AH	Autentica o payload IP e partes selecionadas do cabeçalho IP e as extensões de cabeçalho doIPv6	Autentica todo o cabeçalho do pacote IP mais porções selecionadas do cabeçalho exterior IP
ESP	Encripta o payload IP	Encripta a parte interna do pacote IP
ESP com autenticação	Encripta e autentica o payload IP, mas não autentica o cabeçalho IP	Encripta e autentica a parte interna do pacote IP

Tabela 7- Segurança oferecida pelo IPSec em seus diversos modos de operação

4.1.1. Implementação da VPN

A VPN foi implementada no modo túnel utilizando o ESP (Encapsulating Security Payload) [Kent 1998] encriptando e autenticando a parte interna do cabeçalho IP [Stalings 1999]. O gateway da VPN é representado pelo AP que executa o daemon ISAKMPD (Internet Security Association and Key Management Protocol Daemon), que efetua o gerenciamento automático das chaves e estabelecimento das SAs (Security Associations) .

Todas as estações da WLAN devem ter instalado um cliente VPN. Nesta implementação utilizou-se o SSH Sentinel [SSH 2003], que é um software proprietário para plataforma Windows utilizada no ambiente inicial de testes. A opção pelo SSH Sentinel se deve à farta documentação presente no site da empresa, à facilidade de configuração do programa e a um grande número de opções para o estabelecimento de VPN.

4.1.2. Configuração do AP

a) Editar o arquivo /etc/sysctl.conf

```
net.inet.esp.enable=1          # 0=Disable the ESP IPsec protocol
net.inet.ah.enable=1           # 0=Disable the AH IPsec protocol
```

b) Incluir o arquivo /etc/isakmpd/isakmpd.conf

Ver apêndice J.

c) Incluir o arquivo /etc/isakmpd/isakmpd.policy

Ver apêndice J.

d) Editar o arquivo /etc/rc.conf

```
isakmpd_flags=""        # for normal use: ""
```

4.1.3. Configuração do SSH Sentinel

Um passo a passo para a configuração do SSH Sentinel se encontra no Apêndice K

4.2. Autenticação

4.2.1. Propostas

Três propostas divulgadas na literatura se propõem a implementar o processo de autenticação de uma estação no AP:

i. WEP
ii. 802.1x
iii. DHCP Seguro
iv. Regras de Firewall

a) WEP

Descrito no seção 2.1.

b) 802.1x

A idéia principal desta norma é autenticar as interfaces de rede ligadas entre si (ponto-a-ponto) e subseqüentemente liberar o acesso aos demais recursos da rede, por exemplo, um computador conectado-se a um switch.

A camada MAC implementada no switch teria a função de verificar que esta máquina sendo conectada não foi autenticada e iniciaria o processo de autenticação, como demonstrado na Figura 3. Ao final do processo de autenticação a máquina que fora conectada terá acesso aos demais recursos da rede. Observe que este processo é ativado na camada MAC.

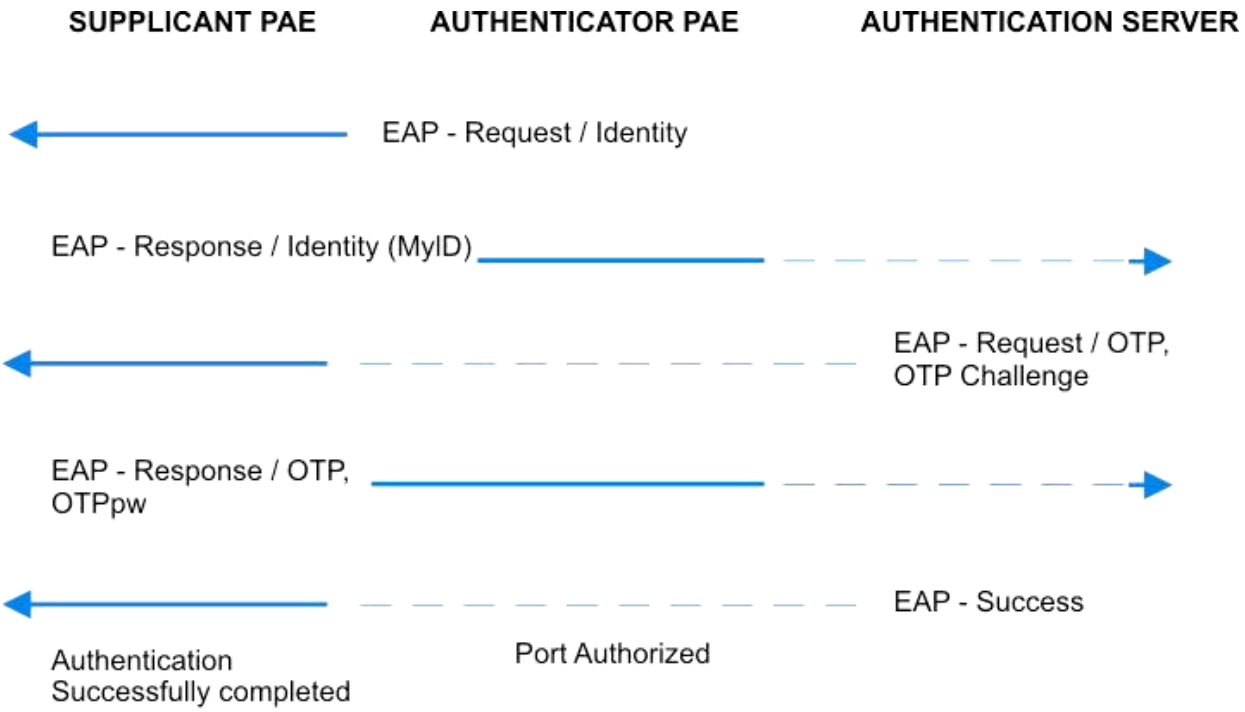

Figura 3 - Processo de Autenticação Iniciado pelo Autenticador 802.1x

Este grupo de atuação do IEEE desenvolveu um padrão que controla as portas de acesso aos recursos da rede. Porta neste caso, identifica o ponto pelo qual um dispositivo troca informações na rede, podendo ser uma interface de rede ou uma porta de um *switch* por exemplo.

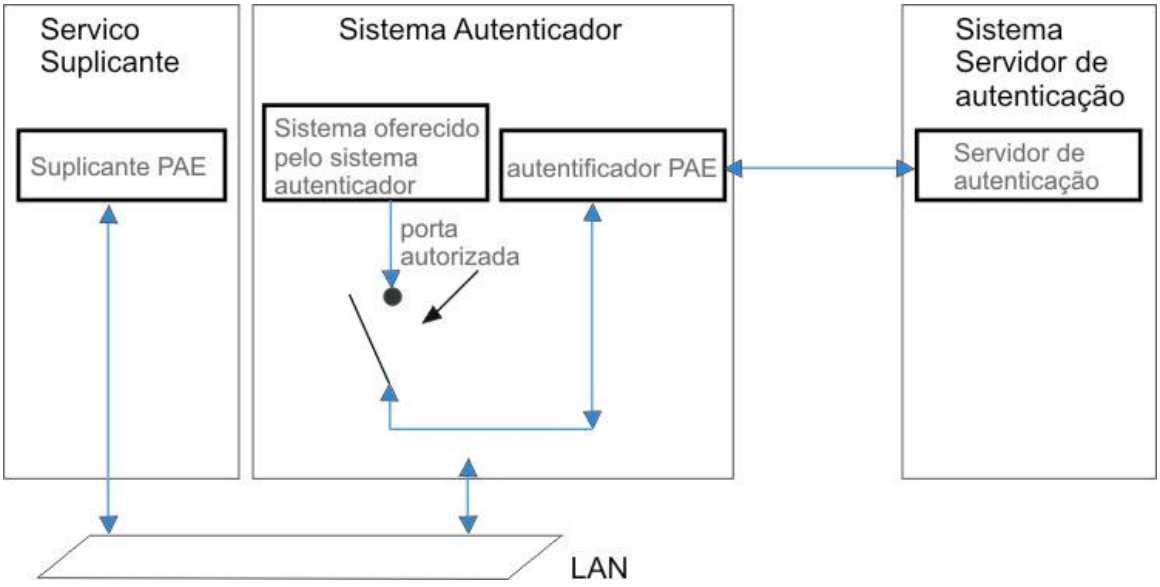

Figura 4 - Arquitetura proposta pelo 802.1x

Na Figura 4 o AP, denominado sistema autenticador, possui inicialmente sua porta de comunicação com as demais estações da rede no estado não autorizado.

Um cliente, denominado suplicante, que queira utilizar os recursos da rede deve se autenticar no AP; o AP enviará estas informações para um servidor de autenticação remoto. Caso as informações sejam válidas, o AP irá liberar a porta de comunicação para a estação suplicante.

Este conceito pode ser aplicado não somente em redes 802.11, como também em redes 802.3 etc.

O processo de autorização utiliza-se de troca de certificados e algoritmos de criptografia.

Um dos pontos fortes deste mecanismo é a existência da norma 802.1x e o início da implementação desta solução em soluções saídas de fábrica. O Windows XP já possui embutido em seu sistema o cliente (suplicante) para o EAP/TLS que é uma implementação do 802.1x.

Um projeto OpenSource da Universidade Maryland do cliente (suplicante) e do autenticador já está em andamento há menos de 1 ano, mas pouquíssima documentação existe, inviabilizando a implementação de uma solução baseada nele.

c) DHCP Seguro

Uma estação que possua um cliente DHCP rodando, que se conecte há uma rede que tenha um servidor DHCP ativo, poderá facilmente obter um IP e acessar os recursos da rede.

A RFC 3118, contém procedimentos e medidas que buscam resolver este problema, utilizando-se de mecanismos de autenticação do cliente antes de fornecer-lhe um IP.

É uma proposta interessante sendo implementada pelo ISC (Internet Software Consortium – http://www.isc.org) na distribuição do seu cliente e servidor DHCP.

d) Regras de Firewall

Um cliente pode obter um IP de um servidor DHCP, mas ele somente terá acesso a uma página específica de autenticação que pode estar contida no AP ou em um servidor de autenticação remoto. Desta forma o cliente deve passar por um processo de autenticação a fim de que as regras no firewall que limitam o seu acesso aos recursos da rede sejam modificadas.

Esta abordagem foi adotada para o desenvolvimento do sistema StrikeIn, descrito adiante.

4.2.2. A Proposta Adotada - O StrikeIn

Uma estação que queira fazer parte de uma rede sem fio local deve possuir um IP para poder se comunicar com os demais membros da rede. Na implementação de redes WLAN, que tem com características a mobilidade das estações, torna-se mais interessante prover estes IPs de forma dinâmica, utilizando-se do DHCP. Portanto, uma estação que já tenha se associado a uma BSS[8] irá requisitar um IP do AP de forma a poder enviar e receber informações através do mesmo.

Este IP é fornecido à estação, mas alguns procedimentos a mais devem ser completados antes da estação poder enviar e receber informações através da rede.

[8] Basic Service Set ou Unidade Básica de Serviço se assemlha à uma célula de comunicação formada ao redor de uma Estação Base existente na telefonia celular.

Ou seja, existe no AP um firewall que bloqueia ou permite acesso aos recursos da rede de acordo com procedimentos de autenticação, detalhados na próxima seção.

Algumas ferramentas OpenSource foram desenvolvidas com este objetivo, mas nenhuma delas mostrou-se adequada para este projeto. Podem ser citadas:

i. Oasis [OASIS 2003] – Ferramenta de autenticação desenvolvida por um grupo StockholmOpen.net, cujo principal empecilho em sua utilização é o fato de se destinar a plataformas Linux e FreeBSD e utilizar PAM, que não é suportado no OpenBSD.

ii. NoCAT [NoCat 2003] – Projeto desenvolvido por uma comunidade de usuários de redes 802.11. Esta ferramenta roda em plataforma Linux e só verifica a saída de um cliente da rede através de timeout

iii. NetLogon [NetLogon 2003] - roda em plataforma Linux e não faz autenticação em banco de dados.

a) Procedimentos de autenticação

Os procedimentos de autenticação e autorização se baseiam nos seguintes pontos:

i) Servidor DHCP modificado [Shaw 2002][9]
ii) Formação da VPN
iii) Regras de firewall
iv) Certificados Digitais
v) Login e senha

O servidor DHCP, no momento em que oferece um IP para um cliente, cria uma entrada na tabela dos IPs válidos e ativos com este IP, permitindo que o firewall tenha regras mais específicas.

[9] Esta característica não foi implementada.

O cliente troca informações com o daemon isakmpd presente no servidor de forma a formar um túnel VPN. A VPN será implementada no modo túnel utilizando o protocolo ESP (Encapsulating Security Payload). Com esta configuração todos os pacotes originados da rede sem fio serão encapsulados pelo cabeçalho do ESP contendo o endereço de destino o endereço do AP. Desta forma qualquer inimigo que observe o tráfego passivamente na rede, não poderá dizer quais os endereços que estão sendo acessados pelos dispositivos da rede sem, pois todos possuirão endereço de destino o IP do AP.

As regras de firewall indicam o que pode ser acessado e por quem em determinado momento e estas regras mudam dinamicamente de acordo com a entrada de uma nova estação na rede e sua devida autenticação e detecção da saída do cliente da rede. A princípio somente mensagens para formação da VPN (udp/500) e as mensagens encriptadas pela VPN podem passar das estações sem fio pelo AP e dentre as mensagens encriptadas somente serão aceitas as destinadas à página web de autenticação (HTTPS) presentes no AP e as de autenticação no domínio SAMBA.

Aceitar o login no domínio SAMBA é uma concessão que deve ser feita pelo momento, pois um usuário que deseje utilizar uma estação terá necessidade de fornecer suas credencias para se conectar na máquina e por conseguinte no domínio, mas até este momento as regras do firewall não foram atualizadas, pois a páginas web de autenticação não foram acessadas, ou seja, o problema é recorrente.

Uma proposta que se faz neste ponto se refere a implementação de um plugin para o código do SAMBA de forma que ele não só seja responsável pelo login no domínio, como também por validar as credenciais no AP e a conseqüente modificação das regras do firewall. Ressalta-se que esta proposta ainda deve ser avaliada e ampliada.

Os certificados digitais permitem que o cliente e o ponto de aceso estabeleçam uma relação mútua de confiança, servindo com um primeiro nível de autenticação.

A autenticação através de login e senha representa um segundo nível de segurança. A princípio o administrador de uma rede poderia instalar o cliente VPN nas máquinas, neste caso o SSH Sentinel, e desta forma qualquer usuário com as credencias para se conectar nesta máquinas poderia se utilizar dos recursos oferecidos pelo AP. A partir do momento em que se

requerem credenciais como login e senha, permite-se um controle dos clientes que vão acessar o AP e um log de acesso granularizado.

Outro aspecto fundamental diz respeito à troca de certificados que autentica mutuamente os clientes e os APs.

A Figura 5 demonstra os passos que são adotados desde que uma estação recebe um IP do Ponto de Acesso até o momento em que ela é desligada da rede.

Inicialmente o AP possui 4 (quatro) regras no seu firewall concernentes às novas estações:

i. Bloqueia tudo por padrão
ii. Permite que somente informações DHCP trafeguem através da interface de rede WLAN
iii. Permite tráfego udp pela porta 500 pela interface WLAN, de forma que uma VPN seja estabelecida.
iv. Permite tráfego encriptado (Tráfego ESP) através da interface WLAN.
    i) Dentre o tráfego encriptado somente aquele destinado à página web de autenticação e à autenticação no domínio samba devem ser permitidos

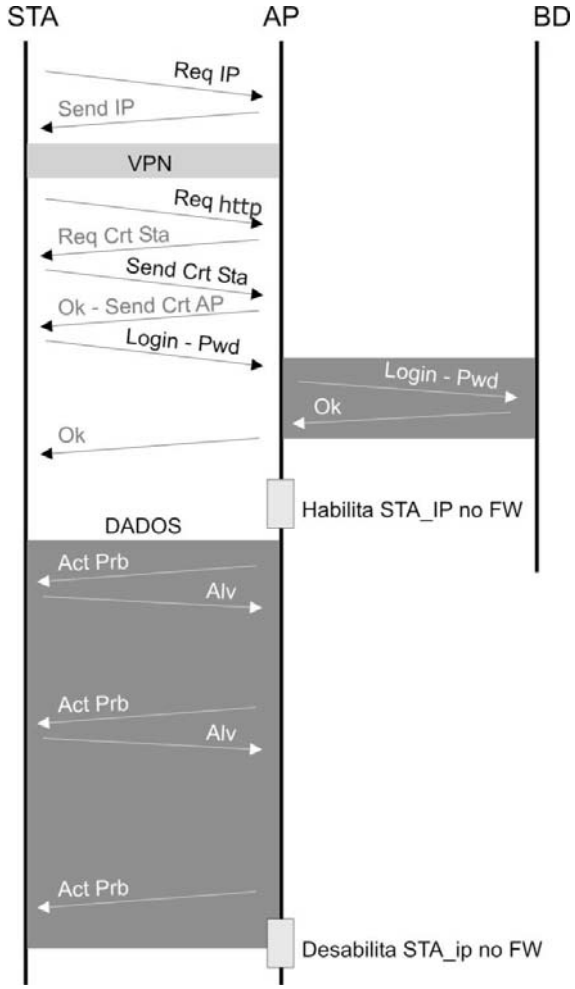

Figura 5 - Troca de Mensagens de Autenticação no ambiente WStrike

Os seguintes passos são adotados ao longo de uma autenticação válida de uma estação:

ii) A estação requisita um IP do servidor DHCP presente na rede, sendo neste caso o AP.

iii) O servidor DHCP no AP envia um IP para a estação

iv) A estação requisita, através do browser, acesso a página WEB de autenticação no AP

v) O servidor web envia o seu certificado digital assinado por uma CA e requisita o servidor do usuário

vi) O usuário aceita o certificado do servidor e envia o seu certificado digital

vii) O servidor web aceita o certificado do cliente e uma página de autenticação com login e senha é apresentada no browser do cliente

viii) O cliente fornece o login e senha

ix) O servidor web encaminha ao banco de dados, através de um túnel SSL, o login e senha do usuário

x) O BD verifica a validade do par login e senha

xi) O Ap recebe uma confirmação positiva do BD e habilita o IP desta estação no firewall

xii) Transferência de dados

xiii) As estações são sondadas em intervalos definidos de tempo (*active probing*)

xiv) Caso a sondagem não obtenha resposta do cliente o IP da STA é desabilitado no firewall

Com esta abordagem pode-se construir regras mais específicas no firewall destinadas somente aos IP's que estão presentes na rede em vez de regras que abrangeriam uma sub-rede.

Um log de entrada e saída dos usuários pode ser mantido de forma bastante organizada em um banco de dados, facilitando tarefas administrativas.

A entrada de um cliente na rede pode ser desta forma controlada através de certificados e senhas, mas uma cautela maior deve ser tida com relação à saída de um cliente da rede.

b) Desligamento de uma estação

A autenticação e autorização de uma estação implica em mudanças nas regras do firewall, desta forma deve-se prover um mecanismo para mudança das regras quando a mesma estação se desliga da rede.

Não é um pressuposto válido contar com uma mensagem de requerimento de desassociação da rede por parte de uma estação, pois os motivos de sua saída podem corresponder ao próprio comprometimento do sistema da estação, motivo este que a impediria de enviar uma mensagem de desligamento.

Uma proposta de fácil implementação seria a de se adotar um tempo de conexão limitado para as estações requerendo uma reautenticação após este tempo ter decorrido. Isto implica em um inconveniente para os usuários, que deverão se reautenticar diversas vezes ao longo de uma conexão.

A escolha deste intervalo de conexão estaria limitada pela falta de funcionalidade que um curto período de tempo implicaria ao usuário e a diminuição na segurança da rede para usuários que se desconectassem antes da conexão chegar ao fim.

Está claro, portanto que propostas mais avançadas devem ser implementadas e estas estão descritas abaixo.

i. Sondagem Ativa (*Active Probing*)

Um processo como o envio de um ping para o cliente é uma forma de se verificar se um cliente continua ativo na rede, porém algumas considerações devem ser feitas.

O envio de um ping não previne que um cliente válido se desligue da rede e que em um instante subseqüente uma estação inimiga utilizado o mesmo IP entre na rede. Desta forma a estação inimiga roubaria a seção da estação válida[10].

Uma proposta intrínseca à construção deste projeto se baseia no fato de que após o processo de autenticação descrito ao longo desta seção seja completado, tem-se que toda a comunicação entre o AP e a estação passam por uma VPN e portanto uma estação inimiga que roubasse a seção da estação válida não teria meios de decriptar o *probe* representado pelo ping e não teria como respondê-lo de forma adequada.

Uma terceira proposta, independente da configuração de uma VPN seria a instalação de um plugin no cliente que responde-se a uma requisição de um daemon presente no firewall, semelhante a um ping, mas neste caso informações adicionais seriam trocadas e garantiriam a autenticidade da estação.

[10] A presença de duas estações com o mesmo IP e mesmo endereço MAC fariam com que o servidor DHCP desabilita-se o IP em questão (release do IP)

Nesta proposta o servidor enviaria a cada mensagem uma chave para decriptar a próxima mensagem e um timestamp, sendo que esta primeira mensagem seria encriptada utilizando-se da senha do sistema do usuário

A estação decriptaria esta primeira mensagem utilizando-se de sua senha e descobriria qual a próxima chave a ser utilizada no processo. A fim de garantir a sua autenticidade e informar ao servidor de que ela continua ativa na rede, a estação enviaria uma mensagem ACK encriptada com a chave recém fornecida pelo servidor. Estes procedimentos garantem a autenticidade da estação e evita ataques replay. O processo está apresentado na Figura 6.

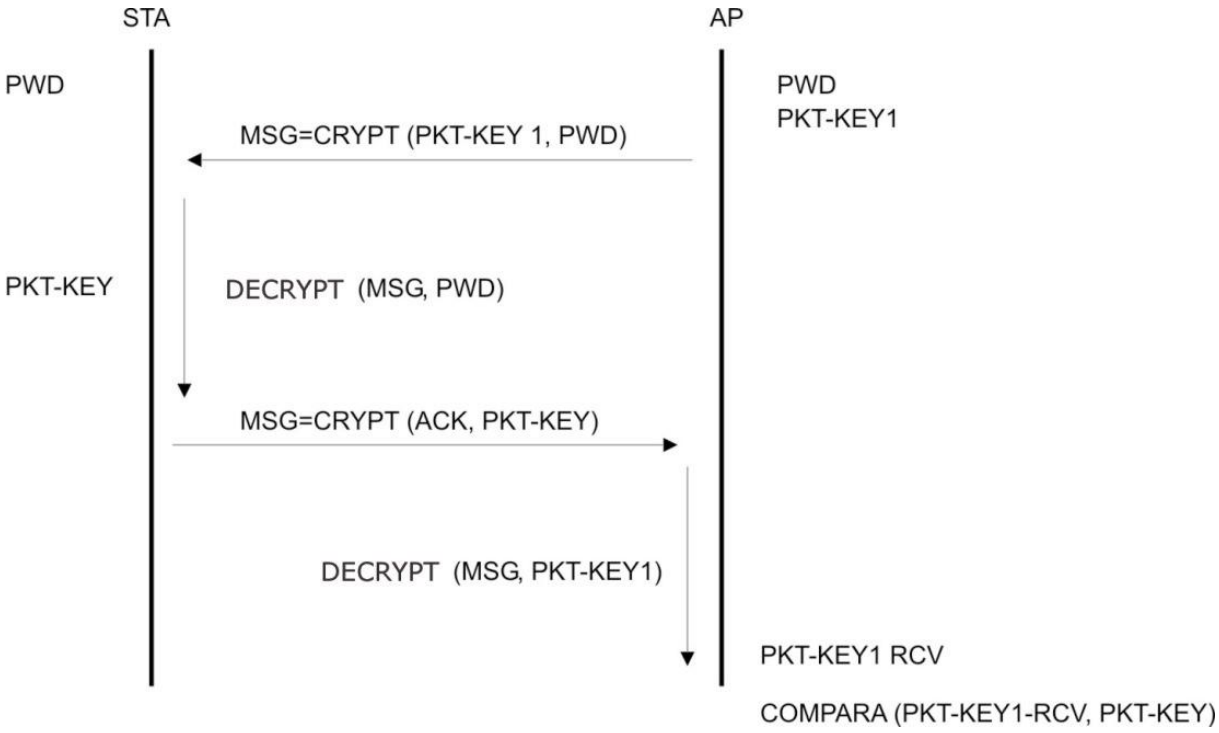

Figura 6 - Troca típica de mensagens do sistema isAlive

ii. **Sondagem Passiva (*Passive Probing*)**

O firewall contido no ponto de acesso mantém uma tabela de estados das conexões que um cliente mantém ativas. Um mecanismo poderia ser implementado que verificasse se um determinado cliente está na tabela de estados do firewall, caso ele não estivesse, isto serviria como indicativo de que a conexão foi desligada. Algumas conjecturas podem ser feitas a este respeito.

Primeiramente, recairía-se no caso do ping trafegando em uma rede sem criptografia, ou seja, um cliente válido que se desligasse e a entrada de um inimigo indicariam a este mecanismo proposto que o cliente continua ativo.

O uso da sondagem ativa requer cuidados com relação a utilização da banda da rede e na definição do período entre verificações de cada cliente e para cada cliente. O uso de sondagem passiva conta somente com o processamento no AP e não consome banda, no entanto sofre da falha do roubo de uma seção por um inimigo.

5. Configuração do Wstrike

Esta seção estará dividida em 4 partes, correspondentes aos 4 níveis de segurança e/ou funcionalidade presentes no WStrike.

Os 4 níveis de segurança estão descritos abaixo:

- Ponto de Acesso
- DHCP
- VPN
- StrikeIN

O WStrike é a proposta destes níveis de segurança e a implementação destes níveis em um microcomputador i386 baseado em OpenBSD, transformando-o em AP e o suporte de um sistema de autenticação StrikeIN baseado em um microcomputador i386 baseado em OpenBSD.

Claramente o Nível 4 requer a existência de um Servidor de Banco de Dados e de uma CA além da existência do AP[11]. Uma opção à utilização de uma CA seria a compra de um Certificado Digital juntamente a diversas empresas que são habilitas para isto, por exemplo a VeriSign (http://www.verisign.com), mas uma opção baseada em software de código aberto e gratuito pode ser adotada, necessitando desta forma da implementação de uma Infraestrutura de Chave Pública (ICP) ou do inglês PKI (Public Key Infrastrucure)[12]. A correta implementação da ICP é fundamental para a validação da arquitetura de segurança proposta neste projeto.

Um diagrama da rede WStrike pode ser observado na Figura 2, onde a CA e BD aparecem como servidores separados (situação ideal), mas no projeto ambos estão implantados na mesma máquina.

---

[11] O Banco de Dados e a Autoridade Certificadora poderiam tecnicamente coexistir no ponto de acesso, mas por questões de segurança, escalabilidade e gerência optou-se por implanta-los em separado.

[12] A implementação de uma ICP para o projeto WStrike foi discutida na seguinte monografia [Carrión 2003].

5.1. AP

5.2.1. Instalação do Sistema Operacional

Utilizou-se para a instalação do AP um CD com o ISO do OpenBSD. O processo de criação do ISO do OpenBSD está descrito no apêndice B. Na Tabela 8 estão descritas as partições instaladas:

Ponto de montagem	Tamanho (MB)
/	500
Swap	256
/usr	1500
/var	1500
/tmp	128

Tabela 8 - Partições do AP

5.2.2. Retirar o sendmail do sistema

A instalação padrão do OpenBSD vem com o sendmail instalado e habilitado para rodar toda vez que o sistema iniciado. Torna-se portanto, um procedimento de segurança retirar o sendmail do sistema e desabilitar a sua inicialização.

a. rm –f /usr/sbin/sendmail
b. No /etc/rc.conf comentar a linha
   sendmail_flags="-L sm-mta -C/etc/mail/localhost.cf -bd -q30m"
c. No /etc/rc.conf acrescentar a linha
   sendmail_flags=NO

5.2.3. Adicionar usuário

Adicionar um usuário para manutenção do sistema. Desta forma previne-se o login remoto do usuário root através do SSH. Neste projeto foi incluído o usuário strikein com e uma senha correspondente.

5.2.4. Configurar o servidor SSH

Algumas modificações de segurança devem ser efetuadas na configuração do servidor SSH. A mais importante diz respeito a somente aceitar o protocolo SSH 2, pois o protocolo SSH 1 apresenta falha de segurança como pode ser observado na Nota de Vulnerabilidade VU#684820 apresentada pelo CERT (www.kb.cert.org/vuls/id/684820).

5.2.5. Configurações /etc/sysctl.conf

Estas modificações habilitam a utilização do IPSEC e possibilita o reenvio pacotes de IP entre as placas de rede presentes na estação:

```
net.inet.ip.forwarding=1        # 1=Permit forwarding (routing) of packets
net.inet.esp.enable=1           # 0=Disable the ESP IPsec protocol
net.inet.ah.enable=1            # 0=Disable the AH IPsec protocol
```

5.2.6. Placas de rede

a) Rede Ethernet

No OpenBSD cada placa de rede possui um arquivo denominado hostname.if no diretório etc, onde a extensão if corresponde à placa de rede (por exemplo: rl0, xl0 etc), onde estão armazenadas as suas configurações.

Configuração do arquivo hostname.rl0

```
inet 10.10.0.16 255.255.255.0 10.10.0.255
```

b) Placa de rede 802.11b

Configuração do arquivo hostname.wi0

```
inet 192.168.0.1 255.255.255.0 192.168.0.255 nwid wstrike mediaopt hostap
```

A linha acima contida no arquivo de configuração da interface de rede 802.11b descreve a seguinte implementação:

Endereço IP	192.168.0.1
Máscara de rede	255.255.255.0
Broadcast	192.168.0.255
SSID	Wstrike
Mediaopt	Hostap

Tabela 9 - Configurações de rede do AP

O SSID é a definição de domínio de uma rede 802.11b. Desta forma diversas redes podem compartilhar a mesma freqüência e estabelecer domínios de existência diferentes.

A opção mediaopt define que esta estação irá se comportar como um ponto de acesso. Como descrito na seção --- o ponto de acesso tem uma função especial na formação da BSS em redes 802.11b, dada a necessidade de se configurar a placa de rede neste modo especial.

5.2.7. Configuração do firewall e do NAT

a) Editar o arquivo /etc/pf.conf com as seguintes regras:

```
# Regras de NAT
nat on rl0 from 192.168.0.0/24 to any -> 10.10.0.16

# Regras de firewall
pass in all
pass out all
```

As regras de NAT fazem com que todos os pacotes provenientes da subrede 192.168.0.0/24 (interface de rede 802.11b) sejam traduzidos para o endereço 10.10.0.16 (interface de rede ethernet do AP) e encaminhados através desta interface.

As regras de firewall são bastantes permissivas neste momento, deixando todos os pacotes saírem e entrarem nas interfaces de rede.

b) Editar o arquivo /etc/rc.conf

Deve-se editar a seguinte linha:

```
pf=NO                # Packet filter / NAT
```

Mudar para:

```
pf=YES               # Packet filter / NAT
```

Desta forma as regras de firewall e NAT serão carregadas toda a vez que o sistema for iniciado.

Neste momento tem-se um AP configurado e qualquer computador contendo uma placa de rede 802.11b, utilizando-se de um IP da subrede 192.168.0.0/24 e gateway 192.168.0.1, pode se utilizar dos recursos da rede.

5.2.8. Cliente NTP

a) Download do pacote:

http://www.openbsd.org/3.2_packages/i386/ntp-4.1.72p1.tgz-long.html

b) Instalação do pacote:

```
$ pkg_add ntp-4.1.72p1.tgz
```

c) Editar o arquivo /etc/rc.conf

```
-------------------- Arquivo rc.conf ----------------------------------------------------
ntpdate_flags=10.10.0.76 # for normal use: NTP server; run before ntpd starts
ntpd=NO              # run ntpd if it exists
-------------------------------------------------------------------------------------------
```

d) Incluir no Cron:

$ crontab –e

Incluir a próxima linha
0,30 * * * * /usr/local/sbin/ntpdate 10.10.0.76 > /dev/null

5.2.9. Servidor DHCP

Dois arquivos são responsáveis pela configuração do servidor DHCP. O primeiro deles é o dhcpd.interfaces, que indica de qual interface de rede será fornecidos os IPs, ou seja, em que porta o servidor DHCP irá ouvir requisições de clientes DHCP. O segundo arquivo é o dhcpd.conf que indica os endereços IPs, servidor DNS, servidor WINS a serem oferecidos aos clientes.

a) Editar o arquivo dhcpd.interfaces

Este procedimento é vital para a segurança do servidor DHCP, pois restringe em quais interfaces de rede o servidor estará escutando as requisições dos clientes DHCP.

---------- Início do arquivo dhcpd.interfaces ----------
$OpenBSD: dhcpd.interfaces,v 1.1 1998/08/19 04:25:45 form Exp $
#
List of network interfaces served by dhcpd(8).
#
wi0
---------- Fim do arquivo dhcpd.interfaces ----------

b) Editar o arquivo /etc/dhcpd.conf:

Neste arquivo estarão definidos:

- Conjunto de IPS a ser distribuído para os clientes mediante requisições DHCP
- Servidores de DNS presentes na rede

- Servidores NETBIOS presentes na rede
- Domínio da rede

```
---------- Início do arquivo dhcpd.conf----------
#       $OpenBSD: dhcpd.conf,v 1.1 1998/08/19 04:25:45 form Exp $
#
# DHCP server options.
# See dhcpd.conf(5) and dhcpd(8) for more information.
#

option  domain-name "ravel.ufrj.br";
option  domain-name-servers 146.164.32.67;
option netbios-name-servers 10.10.0.76;

shared-network MOVEL {

      subnet 192.168.0.0 netmask 255.255.255.0 {
            option routers 192.168.0.1;
            range 192.168.0.2 192.168.0.11;
      }

}
---------- Fim do arquivo dhcpd.conf ----------
```

Na configuração está estabelecido que somente 10 estações poderão estar ativas ao mesmo tempo, já que o grupo de endereços reservados às estações varia de 192.168.0.2 até 192.168.0.11.
Para iniciar o servidor DHCP (dhcpd) deve-se modificar a linha:

```
dhcpd_flags=NO        # for normal use: "-q"
```

para:

```
dhcpd_flags="-q"        # for normal use: "-q"
```

A opção “-q” previne a apresentação de mensagens de copyright durante a inicialização do sistema.

5.2.10. Configuração do Servidor WEB

O AP possuirá uma página web para autenticação dos usuários da rede local sem fio WStrike, e portanto necessita de um servidor web com suporte a PHP e SSL.

O processo de instalação do servidor WEB e PHP pode ser observado no Apêndice C, onde o script instalaapache.sh executa o processo de instalação.

No processo de instalação utilizado, optou-se por baixar os arquivos para o diretório /usr/src, descompactá-los[13] e a partir deste diretório executar o script.

Serão necessários os seguintes arquivos para a instalação correta do Apache com PHP e SSL.

Programa	Arquivo	Versão	Site
Apache	apache_1.3.27.tar.gz	1.3.27	www.apache.org
PHP	php-4.2.3.tar.gz	4.2.3	www.php.net
mod_ssl	mod_ssl-2.8.12-1.3.27.tar.gz	2.8.12	www.modssl.org
openssl	openssl-0.9.6h.tar.gz	0.9.6h	www.openssl.org

Tabela 10 - Programas configurados para o funcionamento do servidor WEB

Após a execução do script de instalação deve-se proceder com os seguintes passos:

i. Copiar o arquivo /usr/src/php-4.2.3/php.ini-dist para /usr/local/lib/php.ini
ii. Modificar as seguintes diretivas:

    safe mode = Off

    register_globals = On
iii. Configuração do Apache

[13] tar xzvf nome-arq

A configuração do Apache diz respeito as informações que devem ser efetuadas no arquivo /var/www/conf/httpd.conf. Informações como nome do servidor web, configurações de segurança e outras devem ser inseridas ou modificadas para o devido funcionamento do servidor. No Apêndice D, podem-se verificar as modificações que foram efetuadas no arquivo httpd.conf default do apache.

5.2.11. Página web e scripts

Deve ser criada uma página web com um formulário e dois campos. Um para o login (nome do usuário) e outro para senha. Um script php receberá estas informações e verificará se existe o usuário e a respectiva senha presentes no banco de dados (tabela passwd).

Se estas informações forem confirmadas o script incluirá o endereço IP da estação, juntamente com o login e senha na tabela ipsValidos. Ao final as regras de firewall serão recarregadas.

a) Página WEB

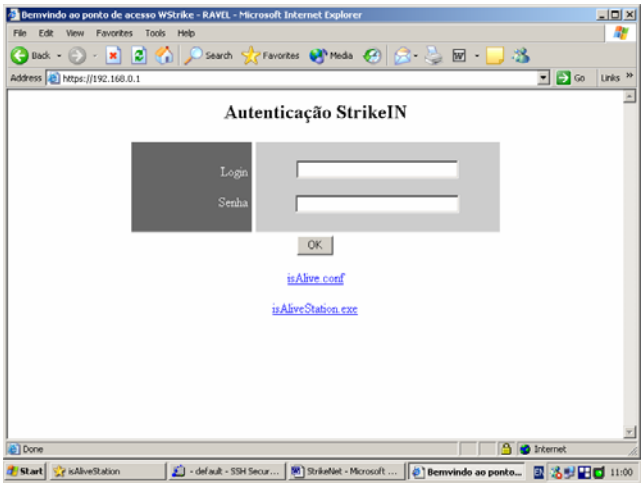

Figura 7- Página de autenticação do AP

b) Script PHP

Responsável por verificar a validade do par login / senha e recarregar as regras do firewall. O código fonte do script está apresentado no apêndice H.

c) SUDO

As regras de firewall são carregadas e recarregadas através do comando:

pfctl –f /var/www/htdocs/pf.conf

Este comando só pode ser executado pelo superusuário do sistema (root) e o programa que será responsável por recarregar as regras do firewall não roda com permissão de superusuário, sendo esta uma boa prática de segurança.

Para tanto utiliza-se o sudo que é um programa que entre outras coisas permite a um usuário executar alguns comandos como superusuário, bastando configurar de forma adequada o arquivo /etc/sudoers, mas para editá-lo deve-se executar o comando *visudo*.

A seguinte linha foi adicionado no /etc/sudoers[14]:

nobody ALL=NOPASSWD:/sbin/pfctl

d) IsAlive[15]

O isAlive é uma implementação em C++ do método de sondagem ativa para verificar o desligamento de um estação do AP, que utiliza a biblioteca Cygwin (www.cygwin.com) para suporte ao desenvolvimento dos algoritmos criptográficos.

A implementação conta com a existência de um daemon no servidor e um processo rodando nos clientes de forma que o AP e as estações possam se comunicar de acordo com o protocolo estabelecido.

i. Código fonte do daemon isAlive:

Ver Apêndice G

ii. Código fonte do cliente isAlive:

Ver Apêndice G.

[14] O servidor web roda com permissão do usuário nobody
[15] A implementação do código em C++ do isAlive foi feita por Bruno Astuto Arouche Nunes

iii. Softwares instalados:

As estações devem ter instalados o programa isAliveClient.exe e o arquivo isAlive.conf que conterá a senha inicial.

- O AP deve ter instalado o cliente mySQL através dos seguintes passos:

Fazer o download do mySQL para o diretório /usr/src e descompactar.

```
$ cd /usr/src/mysql-version
$ ./configure –without-server
$ make
$ make install
```

- API C++ para mySQL

Baixar e instalar a API para C++ para o mySQL (www.mysql.com/downloads/api-mysql++.html)

Fazer o download do mySQL++ para o diretório /usr/src e descompactar.

```
$ automake
$ autoconf
$ ./config
$ make
$ make install
```

- Instalar o isAlive no servidor

Basta copiar o arquivo fonte do isAliveDaemon para uma pasta e executar o seguinte comando no Shell.

```
$ ./SHisAlive.sh
```

O arquivo isAlive.sh possui o seguinte conteúdo.

```
------------------------- Início do arquivo isAlive.sh ----------------------------------------
echo "Compilando..."
g++ -I/usr/local/include/ -I/usr/local/include/mysql -O2 -c isAliveDaemon.cpp tcp_lib.cpp

echo "Linkando..."
g++ -g -O2 -L/usr/local/lib/mysql -o isAliveDaemon isAliveDaemon.o -L/usr/local/lib/ -
lsqlplus -lz -lmysqlclient -lz -lmysqlclient tcp_lib.cpp -lmcrypt -lltdl libcrypto.a

echo "Terminando..."
rm *.o
------------------------- Início do arquivo isAlive.sh ----------------------------------------
```

- Configurar o AP para iniciar o daemon isAlive no *boot*

Incluir a seguinte linha no arquivo /etc/rc.local

```
 echo -n 'isAliveDaemon';       /usr/local/bin/isAliveDaemon >> /dev/null
```

- Instalar o isAliveStation nos clientes.

Os arquivos isAliveStation.exe, cygcrypto-0.9.7.dll, Cygwin1.dll e isAlive.conf. O arquivo isAlive.conf deve conter a senha de acesso à página web de autenticação.

5.3. StrikeIN

O processo de autenticação pelo sistema StrikeIN está baseado na existência de um servidor WEB, um servidor de Banco de Dados e uma Autoridade Certificadora (CA). Nesta implementação o servidor web será implementado no próprio AP, enquanto que o banco de dados e a CA serão instalados em uma máquina em separado[16].

[16] Para o processo de instalação do servidor de banco de dados consultar a próxima seção.

5.3.1. Instalação do Sistema Operacional (BD & CA[17])

O processo de instalação é semelhante ao detalhado da seção 5.1.1 até a seção 5.1.4, exceto pela configuração das partições.

Na Tabela 11 estão descritas partições instaladas:

Ponto de montagem	Tamanho (MB)
/	500
Swap	256
/usr	1500
/var	1500
/tmp	128

Tabela 11 - Partições do StrikeIN

5.3.2. Banco de Dados MySQL

a) Instalação

A primeira etapa do processo corresponderia a baixar o arquivo mysql-VERSION.tar.gz para o diretório /usr/src e executar a série de comandos abaixo.

```
$ groupadd mysql
$ useradd -g mysql mysql
$ gunzip < mysql-VERSION.tar.gz | tar -xvf -
$ cd mysql-VERSION
$ ./configure --prefix=/usr/local/mysql
$ make
$ make install
$ scripts/mysql_install_db
$ chown -R root  /usr/local/mysql
```

[17] Estará indicado em parênteses em qual máquina o processo de instalação e configuração estará sendo efetuado, onde AP indica a máquina Ponto de Acesso e BD & CA representa a máquina que contém o banco de dados e a Autoridade Certificadora

```
$ chown -R mysql /usr/local/mysql/var
$ chgrp -R mysql /usr/local/mysql
$ cp support-files/my-medium.cnf /etc/my.cnf
$ /usr/local/mysql/bin/safe_mysqld --user=mysql &
```

Deve-se configurar o MySQL de forma que ele seja invocado toda vez que o sistema for inicializado, para tanto devem-se acrescentar as seguintes linhas de comando no arquivo /etc/rc.local:

```
if [ -x /usr/local/mysql/share/mysql/mysql.server ]; then
        echo -n ' mysqld';      /usr/local/mysql/share/mysql/mysql.server start
fi
```

b) Usuário wstrike

O MySQL permite que se incluam permissões de acessos utilizando-se de três prerrogativas:

i. Usuário
ii. Máquina (host)
iii. Banco de Dados

Desta forma estabelece-se que um usuário a partir de uma máquina tem acesso a um determinado banco de dados. Para tanto, foi criado um usuário denominado strikein com permissão de acesso da máquina wstrike (AP) ao banco de dados strikein.

O banco de dados strikein conterá as informações pertinentes ao processo de autorização e autenticação dos usuários da rede sem fio local.

Para criar este usuário, foram seguidos os seguintes passos:

strikein# mysql –h localhost –u root –p

{Observe que a senha de root está em branco}

mysql > use mysql;

mysql > insert into table user values(“wstrike”, “wstrike”, PASSWORD(“wstrike”), “Y”, “Y”, “Y”, “Y”, “Y”, “Y”, “Y”, “Y”, “Y”, “Y”, “Y”, “Y”, “Y”, “Y”, “Y”,);

mysql > insert into table db values(“wstrike”, “strikein”, “wstrike” , “Y”, “Y”, “Y”, “Y”, “Y”, “Y”, “Y”, “Y”, “Y”, “Y”, “Y”,);

mysql > flush privileges;

c) Modelo de Dados do StrikeIN

Para criar o modelo de dados foi utilizado o Erwin 4.0 desenvolvido pela Computer Associates.

d) Criando o BD StrikeIN

O Erwin 4.0 possui uma ferramenta de geração de scripts para a construção do banco de dados a partir do modelo físico criado nele. Existem muitas opções de configuração e diversas escolhas quanto ao tipo de banco de dados a ser utilizado de forma que o script seja customizado.

No Apêndice E encontra-se o script utilizado para a criação do BD StrikeIN.

5.3.3. Autoridade Certificadora (BD & CA)

A Autoridade Certificadora será responsável por assinar e revogar os certificados do ponto de acesso e dos clientes da rede. Este ponto é fundamental na implementação de segurança proposto dentro do StrikeIN.

Através de certificados assinados pela autoridade certificadora (Ravel) será possível garantir a autenticidade do ponto de acesso junto às demais estações, assim como possibilitará ao ponto de acesso verificar a autenticidade dos usuários, tem-se portanto uma mútua autenticação.

O primeiro passo neste sentido é de se criar uma autoridade certificadora que possa assinar e revogar os certificados e que seja reconhecida como a autoridade máxima em certificação dentro do ambiente de implantação do WStrike. Neste caso todos devem confiar, a priori neste CA[18].

Para a criação de uma CA no OpenBSD foram adotados os seguintes passos [SSL 02]:

$ openssl genrsa –des3 –out ca.key 1024
$ openssl req –new –x509 –day 365 –key ca.key –out ca.crt

O primeiro comando irá criar uma chave RSA de 1024 bits a ser armazenada no arquivo ca.key.

O segundo comando cria um certificado (ca.crt) assinado pela própria chave da CA (ca.key). Ou seja, esta certificadora é a raiz no ambiente de confiança a ser estabelecido (no caso deste projeto o próprio laboratório Ravel e seus usuários).

Certificados Digitais

Tanto o servidor WEB quanto os usuários da rede sem fio local devem possuir certificados digitais assinados pela CA do ambiente estabelecido.

a) Certificado do Servidor WEB [SSL 02]:

Assumindo-se que o processo de criação do certificado esteja sendo iniciado no AP, tem-se:

Gera uma chave RSA para o servidor WEB
wstrike$ openssl genrsa –des3 –out server.key 1024

Cria uma requisição de assinatura de certificado (CSR – Certificate Signing Request)
wstrike$ openssl req –new –key server.key –out server.csr

[18] O processo de confiança em que se desdobra o processo de certificação digital é encontrado na implantação das politicas de certificação digital como a corrente no Brasil. Em determinado ponto deste processo, deve-se confiar incondicionalmente em alguma entidade como sendo a raiz de todas, no caso denomidada CA raiz.

Neste
ponto deve-se enviar o CSR para a CA, a fim de que seja assinado.

Estando no computador que abriga a CA, deve-se executar o passo abaixo de forma a transformar o arquivo server.csr em um server.crt, ou seja, para transformar uma Requisição de Assinatura de Certificado para um Certificado Digital.

$ ./sign.sh server.csr

Este script pode ser encontrado juntamente com a distribuição do código fonte do mod_ssl no diretório pkg.contrib.

A CA deve remeter ao servidor WEB (AP) o arquivo server.crt e este deve-se ser colocado em um diretório apropriado, assim como o arquivo ca.crt.

No projeto optou-se por colocar todas as informações relativas aos certificados (chaves, requisições e certificados) no diretório /var/www/conf/cert.

Para que o servidor WEB utilizar estes novos certificados deverão ser feitas algumas modificações no arquivo httpd.conf, listadas abaixo.

Configurando corretamente a localizações dos certificados e chaves
SSLCertificateFile /var/www/conf/cert/server.crt
SSLCertificateKeyFile /var/www/conf/cert/server.key
SSLCACertificatePath /var/www/conf/cert

Neste momento o servidor WEB estará fornecendo o seu certificado digital toda vez que uma conexão segura for requerida pelo cliente (https), mas isto não basta. Deve incluir mais algumas linhas indicando que o servidor deve requisitar o certificado digital do cliente.

Solicitando certificado digital do cliente[19]

[19] Ver referência [SSL 01]

SSLVerifyClient require
SSLVerifyDepth 1
SSLCACertificateFile conf/cert/ca.crt

Deve-se reiniciar o apache para que estas modificações sejam lidas:

wstrike$ apachectl stop
wstrike$ apachectl startssl

b) Certificados dos Clientes [Mooney 2001]

O processo de criação do certificado dos clientes é bastante semelhante, porém somente um passo adicional deve ser feito ao final. Os browsers utilizam o padrão PKCS#12 para os certificados digitais pessoais, portanto um processo de conversão deve ser efetuado ao final.

cliente$ openssl genrsa –out client.key 1024
cliente$ openssl req –new –key client.key –out client.csr

O arquivo deve ser enviado para a CA a fim de que seja assinado:

CA$ ./sign client.csr

Neste passo processa-se a conversão para pkcs12
cliente$ openssl pkcs12 –export –clcerts –in client.crt –inkey client.key –out client.p12

O arquivo deve ser enviado ao cliente, juntamente com o certificado da CA.

Por que o certificado da CA deve ser importado?

Obviamente o certificado do cliente deve ser importado pelo browser que ele utilizará para a autenticação juntamente ao servidor web (AP). O certificado da CA deve ser incorporado

pelo browser também, porque a CA criada para este projeto não consta como CA raiz no conjunto previamente configurado nos browsers[20].

De forma simples tem-se:

i. Importa-se o certifica do cliente na categoria de certificado pessoal
ii. Importa-se o certificado da CA na categoria CA raiz

No Apêndice F estão descritos os passos para importação dos certificados da CA e do usuário.

[20] A importação dos certificados foi feita no Internet Explorer

6. Conclusão

Foi possível montar um AP para redes 802.11b utilizando-se de softwares gratuitos e de código aberto, sendo este um facilitador para a implantação de redes locais sem fio. A farta documentação disponível na Internet foi fundamental para que o projeto tivesse esta abrangência, podendo-se implementar diversas características de segurança.

Importante ressaltar que as características de segurança inicialmente propostas foram implementadas com sucesso garantindo o sigilo e integridade das informações que passam pelo meio aéreo.

A utilização de certificados digitais para autenticação mútua das estações e do AP é peça fundamental na segurança dos usuários e da entidade que implementa a WLAN. O servidor web Apache tornou possível a implementação desta característica de maneira simples eficaz e ressalta-se também que a farta documentação existente no site do Apache e do openssl facilitou a implementação segura e adequada desta funcionalidade.

A arquitetura de segurança proposta, em conjunto com os métodos de autenticação, autorização e criptografia utilizados, permitem o uso em produção de uma WLAN em uma rede local cabeada. O nível de segurança da rede sem fio através da utilização da VPN e do StrikeIN permitem um controle maior sobre as máquinas e usuários ativos na rede em comparação com uma LAN, além de se estar protegido contra ataques passivos como de sniffers, já que uma rede local cabeada normalmente não implementa VPN.

Neste mesmo contexto tem-se o padrão 802.1x que autentica cada máquina que deseja utilizar os recursos da rede, tem-se como exemplo a autenticação de um microcomputador em um switch antes que o primeiro possa efetuar transações na LAN. Este tipo de controle está presente através da implementação do StrikeIN e é bastante reforçado pela utilização do isAlive.

A concepção do isAlive como forma de garantir que não haja roubo de seção mostrou-se eficaz no seu propósito e sua funcionalidade pode ser extendida para outras ferramentas que necessitam a verificação de usuários ativos na rede. Diversos desafios estiveram presentes na

elaboração do protocolo a ser seguido pelo isAlive, além da necessidade de que uma implementação do código seguro.

A geração de números pseudo-aleatórios é fundamental para garantir a validade do isAlive, outros algoritmos devem ser implementados, pois certamente o uso da função rand da linguagem C não é a mais adequada.

A implementação do código seguro onde funções do tipo strcat e strcp devem ser substituídas por funções mais seguras como strncat e strcp. O armazenamento da chave inicial do isAlive é feita através de um arquivo texto na máquina do usuário, obviamente esta solução não é adequada, portanto outras formas de se guardar esta senha inicial devem ser vislumbrara e implementadas de forma segura.

Diversos aspectos deste projeto devem ser refinadas e até mesmo adicionados o que leva a oportunidades para o desenvolvimento de trabalhos futuros.

- Implementação de uma PKI para dar suporte ao gerenciamento dos certificados digitais do AP e das estações
- Teste de performance relacionado ao overhead introduzido devido aos processos criptográficos da VPN
- Utilização de outros algoritmos na VPN como por exemplo o AES
- Melhorias na segurança do isAlive, como por exemplo, projetar um novo gerenciador de números pseudoaleatórios e desenvolver um método mais seguro para proteger o arquivo que contém a senha inicial do cliente isAlive
- Estudar a possibilidade de utilizar o SAMBA como um sistema autenticador de estações da WLAN
- Implementar modificações no servidor DHCP de forma que este interaja com o firewall (pf) de forma que as regras de segurança sejam mais rígidas
- Submeter os isAlive a um teste de segurança, como por exemplo na implementação de uma honeynet sem fio
- Verificar a usabilidade do sistema com usuários beta, a fim de promover melhorias na interfaces de autenticação e nos processos necessários para que um usuário se autentique até o momento em que é capaz de utilizar os recursos da rede

7. Bibliografia

Allard, Grupo de discussão da empresa Allard onde é apresenta uma explicação para o problema encontrado entre MTU e o IPSec, Página visitada em 2003

http://www.allard.nu/pipermail/openbsd-ipsec-clients/2002-February/000160.html

ANSI/IEEE 802.11 “Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, Edição de 1999

Apache Web Server, Página web do servidor web desenvolvido pela Apache Software Foundation, Página visitada em 2003

http://httpd.apache.org/

Arsenault, A. e Turner, S. (2002) “Internet X.509 Public Key Infrastructure: Roadmap”, draft-ietf-pkix-roadmap-09.txt

Borisov, N., Goldberg, I. e Wagner D. (2001) “Intercepting Mobile Communications: The Insecurity of 802.11”, 7th Annual International Conference on Mobile Computing and Networking.

Carrión, Demetrio S. D. (2003) “Implementando uma PKI para uma WLAN baseada no WStrike”, Monografia para o Curso de Redes Integradas de Faixa Larga e ATM do Mestrado em Redes de Computadores, COPPE/UFRJ

Casole, M. (2002) “WLAN Security – Status, Problems and Perspective”, European Wireless 2002

Cisco, Indica como modificar o MTU no Windows XP e como desabilitar o PMTUD (Path Maximum Transmission Unit Discovery), Página visitada em 2003

http://www.cisco.com/warp/public/105/38.shtml#win2k

Drach, S. (1999), “DHCP Option for The Open Group's User Authentication Protocol”, RFC 2485

Dyck, T. (2002) “OpenBSD 3.2 Gets It Right”, artigo publicado em

http://www.eweek.com/article2/0,3959,640713,00.asp

Fluhrer, S., Mantin, I. e Shamir, A. (2001) “Weaknesses in the key scheduling algorithm of RC4”, Eighth Annual Workshop on Selected Areas in Cryptography.

Hartmeier, D. (2001), “Design and Performance of the OpenBSD Stateful Packet Filter”

Hedenfalk, M. (2002) “Access Control in an Operator Neutral Public Access Network”, Tese de Mestrado defendida no Royal Institute of Technology (KTH), Department of Microelectronics and Information Technology (IMIT)

IPSec, Página web do Grupo de Trabalho IPSec do IEEE, Página visitada em 2003, http://www.ietf.org/html.charters/ipsec-charter.html

Kent, S. e Atkinson, R. (1998) “IP Encapsulating Security Payload (ESP)” , RFC 2406

Manual MySQL, Página de manual do MySQL, Página vistada em 2002

http://www.mysql.com/doc/en/index.html

Manual OpenBSD, Página de manual do OpenBSD sobre o driver wi para placa de redes sem fio, Página visitada em 2002

www.openbsd.org/cgi-bin/man.cgi?query=wi&apropos=0&sektion=0&manpath=

OpenBSD+Current&arch=i386&format=html

Martins, A. et al (2001) “IP Seguro (IPSEC) e Redes Virtuais Privadas (VPNs)”, Apostila de Treinamento em Segurança, Laboratório RAVEL

Martins, Alessandro et al (2001) “PKI”, Apostila de Treinamento em Segurança, Laboratório Ravel

Mooney, R. 2001, “Client Certificate Authentication with Apache (An Example)”, www.aboveground.cx/~rjmooney/projects/misc/clientcertauth.html

MySQL 01, Página web do SGBD MySQL, Página visitada em 2003, http://www.mysql.com

MySQL 02, Comparativo de performance entre vários SGBDs em diversas plataformas, Página visitada em 2002, http://www.mysql.com/information/benchmarks.html

NetLogon, Ferramenta de autenticação centralizada, Páfina visitada em 2002, http://www.unit.liu.se/dokument/natverk/netlogon.html

NoCatAuth, Ferramenta de autenticação centralizada, Página visitada em 2002,
http://www.nocat.org

OASIS, Ferramenta de autenticação centralizada, Página visitada em 2002,
http://software.stockholmopen.net/

Open1X, Implementação de software aberto do padrão 802.1x, Página visitada em 2002, http://www.open1x.org

OpenBSD, Sistema operacional de código aberto e gratuito baseado no BSD 4.4, Página visitada em 2002, http://www.openbsd.org

OpenSSL, Projeto OpenSSL que implementa o SSL e o TLS, Página visitada em 2003 http://www.openssl.org

PHP, Grupo de desenvolvimento da linguagem de scripts php, Página visitada em 2003 http://www.php.net

Shaw, Derek G. e Boscia, Nichole K. (2002) "Wireless Firewall Gateway White Paper – Revision 3", NASA Advanced Supercomputing Division

Srisuresh, P. e Egevang K. (2001) "Traditional IP Network Address Translator (Traditional NAT)", RFC 3022

SSH Sentinel, Cliente VPN para a plataforma Windows, página web visitada em 2003
http://www.ssh.com/products/security/sentinel/

SSL 01, Página do servidor web Apache indicando como requisitar certificados dos clientes, Página visitada em 2002, httpd.apache.org/docs-2.0/ssl/ssl_howto.html

SSL 02, Página do servidor web Apache indicando como criar uma CA e assinar certificados, Página visitada em 2002, httpd.apache.org/docs-2.0/ssl/ssl_faq.html

Stalings, William, "Cryptography and Network Security – Principles and Practice", 2 ed, Prentice Hall, 1999, p. 399-440

Tanenbaum, Andrew S., "Computer Networks", 3ª ed, PH PTR, 1996, 813p

Thayer, R., Doraswamy, N. e Glenn, R. (1998) "IP Security Document Roadmap", RFC 2411

Vaughan-Nichols, Steven J. (2001) “OpenBSD: The most secure OS around”, artigo publicado http://techupdate.zdnet.com/techupdate/stories/main/0,14179,2822483,00.html

Walker, Jesse R. (2000) “Unsafe at any key size: An Analysis of the WEP encapsulation”, IEEE Document 802.11-00/362.

Weise, J. (2001) “ Public Key Infrastructure Overview”, Sun BluePrints OnLine, http://ww.sun.com/solutions/blueprints/0801/publickey.pdf

Apêndice A – NICs 802.11b suportadas pelo OpenBSD 3.2

Placa	Chip	Barramento
3Com AirConnect 3CRWE737A	Spectrum24	PCMCIA
3Com AirConnect 3CRWE777A	Prism-2	PCI
ACTIONTEC HWC01170	Prism-2.5	PCMCIA
Addtron AWP-100	Prism-2	PCMCIA
Agere Orinoco	Hermes	PCMCIA
Apple Airport	Hermes	macobio
Buffalo AirStation	Prism-2	PCMCIA
Buffalo AirStation	Prism-2	CF
Cabletron RoamAbout	Hermes	PCMCIA
Compaq Agency NC5004	Prism-2	PCMCIA
Contec FLEXLAN/FX-DS110-PCC	Prism-2	PCMCIA
Corega PCC-11	Prism-2	PCMCIA
Corega PCCA-11	Prism-2	PCMCIA
Corega PCCB-11	Prism-2	PCMCIA
Corega CGWLPCIA11	Prism-2	PCI
Dlink DWL520	Prism-2.5	PCI
Dlink DWL650	Prism-2.5	PCMCIA
ELSA XI300	Prism-2	PCMCIA
ELSA XI325	Prism-2.5	PCMCIA
ELSA XI325H	Prism-2.5	PCMCIA
ELSA XI800	Prism-2	CF
EMTAC A2424i	Prism-2	PCMCIA
Ericsson Wireless LAN CARD C11	Spectrum24	PCMCIA
Gemtek WL-311	Prism-2.5	PCMCIA
Hawking Technology WE110P	Prism-2.5	PCMCIA
I-O DATA WN-B11/PCM	Prism-2	PCMCIA
Intel PRO/Wireless 2011	Spectrum24	PCMCIA
Intersil Prism II	Prism-2	PCMCIA
Intersil Mini-PCI	Prism-2.5	PCI
Linksys Instant Wireless WPC11	Prism-2	PCMCIA

Linksys Instant Wireless WPC11 2.5	Prism-2.5	PCMCIA
Linksys Instant Wireless WPC11 3.0	Prism-3	PCMCIA
Lucent WaveLAN	Hermes	PCMCIA
NANOSPEED ROOT-RZ2000	Prism-2	PCMCIA
NDC/Sohoware NCP130	Prism-2	PCI
NEC CMZ-RT-WP	Prism-2	PCMCIA
Netgear MA401	Prism-2	PCMCIA
Netgear MA401RA	Prism-2.5	PCMCIA
Nokia C020 Wireless LAN	Prism-I	PCMCIA
Nokia C110/C111 Wireless LAN	Prism-2	PCMCIA
NTT-ME 11Mbps Wireless LAN	Prism-2	PCMCIA
Proxim Harmony	Prism-2	PCMCIA
Proxim RangeLAN-DS	Prism-2	PCMCIA
Samsung MagicLAN SWL-2000N	Prism-2	PCMCIA
Symbol Spectrum24	Spectrum24	PCMCIA
SMC 2632 EZ Connect	Prism-2	PCMCIA
TDK LAK-CD011WL	Prism-2	PCMCIA
US Robotics 2410	Prism-2	PCMCIA
US Robotics 2445	Prism-2	PCMCIA

Apêndice B - Script para criação do ISO do OpenBSD 3.2

```
# Produzido por André Sarmento Barbosa
#!/bin/sh

mkisofs -v -r -l -L -T \
        -V "Strike_OpenBSD" \
        -A "Strike_OpenBSD v3.2 - Release 01/11/2002" \
        -b cdrom32.fs \
        -c boot.catalog \
        -o $1 \
       “path”/OpenBSD32/
```

Apêndice C - Script de instalação do Apache com suporte a PHP e SSL

```
#!/bin/sh
# Procedimentos para instalação do Servidor Web no OPenBSD
# Apache + SSL + PHP (suporte ao mysql)
# Roteiro criado por Demetrio Cárrion
# Script desenvolvido por Alexandre Pinaffi Andrucioli

# Instruções Gerais
# 1) Baixar apache, openssl, mod_ssl e php para /usr/src
#  - Descompar os arquivos
#  - Retirar Versão do apache no arquivo XXXX (ainda em teste)
#  - Altere as variáveis abaixo para o nome crrespondente da pasta após descompactada
APACHE=apache_1.3.27
VERSAOAPACHE=1.3.27
OPENSSL=openssl-0.9.7a
PHP=php-4.3.1
MODSSL=mod_ssl-2.8.12-1.3.27

# 2) Para o Apache e fazer backup do /htdocs e do /conf
if [ -f /usr/sbin/apachectl ];then
/usr/sbin/apachectl stop
fi

if [ -d /var/www/conf ];then
mv /var/www/conf /var/www/conf.orig
fi

if [ -d /var/www/htdocs ];then
mv /var/www/htdocs /var/www/htdocs.orig
fi

# 3) Compilar
# OpenSSL
```

```
cd /usr/src/$OPENSSL
./config no-threads no-asm
make
make test
make install

# Apply mod_ssl to Apache source tree
cd /usr/src/$MODSSL
./configure --with-apache=../$APACHE --with-ssl=../$OPENSSL
cd ..

# Retirar a Versao do Apache
cd /usr/src/$APACHE/src/include
cat httpd.h | sed s%$VERSAOAPACHE%'Apache [Not Available]'% > httpd.new
mv httpd.h httpd.orig
mv httpd.new httpd.h

# Pre-configure Apache for PHP4's configure step
cd /usr/src/$APACHE
./configure --with-layout=OpenBSD

# Configure PHP4 and apply it to the Apache source tree
cd /usr/src/$PHP
set env CFLAGS='-O2 -I/usr/src/$OPENSSL/include'
./configure --with-apache=../$APACHE --with-mysql --enable-memory-limit=yes --enable-
debug=no --with-gettext --with-imap --with-xml --enable-mime-magic --with-iconv --enable-
mbstring --with-imap-ssl=../$OPENSSL/
gmake || make
gmake install || make install

# Build/install Apache with mod_ssl and PHP4
cd /usr/src/$APACHE
set env SSL_BASE=/usr/src/$OPENSSL
```

```
./configure --with-layout=OpenBSD --enable-module=ssl --activate-
module=src/modules/php4/libphp4.a --enable-module=php4
make
make certificate
make install

# 4) Recuperando diretorio original

cd /usr/src/$APACHE/src/include
mv httpd.orig httpd.h

if [ -d /var/www/htdocs.orig ];then
rm -rf /var/www/htdocs
mv /var/www/htdocs.orig /var/www/htdocs
fi

if [ -d /var/www/conf.orig ];then
rm -rf /var/www/conf
mv /var/www/conf.orig /var/www/conf
fi

# 5) Reiniciar Serviço

cat << EOF

Se não apareceu mensagens de erro os programas foram atualizados corretamente.

Altere o arquivo /usr/local/lib/php.ini:

"safemode = off"

Apos isso inicie o serviço com o comando:

/usr/sbin/apachectl start
```

Para que o apache inicie automaticamente insira a linha acima no arquivo

/etc/rc.local

EOF

Apêndice D - Diff do httpd.conf para o httpd.conf.default

```
wstrike# diff -u httpd.conf.default httpd.conf
--- httpd.conf.default  Fri Dec  6 14:32:44 2002
+++ httpd.conf  Fri Dec  6 15:46:40 2002
@@ -50,6 +50,9 @@
 # Unix platforms.
 #
 ServerType standalone
+ServerTokens min
+
+AddType application/x-httpd-php .php

 #
 # ServerRoot: The top of the directory tree under which the server's
@@ -266,7 +269,7 @@
 # e-mailed.  This address appears on some server-generated pages, such
 # as error documents.
 #
-ServerAdmin root@wstrike.my.domain
+ServerAdmin carrion@ravel.ufrj.br

 #
 # ServerName allows you to set a host name which is sent back to clients for
@@ -284,7 +287,7 @@
 # machine always knows itself by this address. If you use Apache strictly for
 # local testing and development, you may use 127.0.0.1 as the server name.
 #
-#ServerName wstrike.my.domain
+ServerName www.wstrike.ravel.ufrj.br

 #
 # DocumentRoot: The directory out of which you will serve your
@@ -325,7 +328,7 @@
```

```
 # Note that "MultiViews" must be named *explicitly* --- "Options All"
 # doesn't give it to you.
 #
-    Options Indexes FollowSymLinks MultiViews
+    Options FollowSymLinks MultiViews

 #
 # This controls which options the .htaccess files in directories can
@@ -371,7 +374,7 @@
 # directory index.  Separate multiple entries with spaces.
 #
 <IfModule mod_dir.c>
-    DirectoryIndex index.html
+    DirectoryIndex index.html index.php index.htm
 </IfModule>

 #
@@ -1069,8 +1072,8 @@

 #  General setup for the virtual host
 DocumentRoot "/var/www/htdocs"
-ServerName wstrike.my.domain
-ServerAdmin root@wstrike.my.domain
+ServerName www.wstrike.ravel.ufrj.br
+ServerAdmin root@wstrike.ravel.ufrj.br
 ErrorLog /var/www/logs/error_log
 TransferLog /var/www/logs/access_log

@@ -1091,7 +1094,7 @@
 #   built time. Keep in mind that if you've both a RSA and a DSA
 #   certificate you can configure both in parallel (to also allow
 #   the use of DSA ciphers, etc.)
-SSLCertificateFile /var/www/conf/ssl.crt/server.crt
+SSLCertificateFile /var/www/conf/cert/server.crt
```

```
 #SSLCertificateFile /var/www/conf/ssl.crt/server-dsa.crt

 #   Server Private Key:
@@ -1099,7 +1102,7 @@
 #   directive to point at the key file.  Keep in mind that if
 #   you've both a RSA and a DSA private key you can configure
 #   both in parallel (to also allow the use of DSA ciphers, etc.)
-SSLCertificateKeyFile /var/www/conf/ssl.key/server.key
+SSLCertificateKeyFile /var/www/conf/cert/server.key
 #SSLCertificateKeyFile /var/www/conf/ssl.key/server-dsa.key

 #   Server Certificate Chain:
@@ -1118,7 +1121,7 @@
 #   Note: Inside SSLCACertificatePath you need hash symlinks
 #         to point to the certificate files. Use the provided
 #         Makefile to update the hash symlinks after changes.
-#SSLCACertificatePath /var/www/conf/ssl.crt
+SSLCACertificatePath /var/www/conf/cert
 #SSLCACertificateFile /var/www/conf/ssl.crt/ca-bundle.crt

 #   Certificate Revocation Lists (CRL):
@@ -1136,8 +1139,9 @@
 #   number which specifies how deeply to verify the certificate
 #   issuer chain before deciding the certificate is not valid.
-#SSLVerifyClient require
-#SSLVerifyDepth  10
+SSLVerifyClient require
+SSLVerifyDepth  1
+SSLCACertificateFile conf/cert/ca.crt

 #   Access Control:
 #   With SSLRequire you can do per-directory access control based
```

Apêndice E - Script para criação do BD StrikeIN

```
CREATE TABLE ipsValidos (
    senhaAtual        VARCHAR(20) NOT NULL,
    login             VARCHAR(20) NOT NULL,
    enderecoIP        VARCHAR(16) NOT NULL,
    TTL               int(2) NOT NULL

);


ALTER TABLE ipsValidos
    ADD PRIMARY KEY (login, enderecoIP);


CREATE TABLE passwd (
    senha             VARCHAR(20) NOT NULL,
    login             VARCHAR(20) NOT NULL,
    nome              VARCHAR(20) NOT NULL
);


ALTER TABLE passwd
    ADD PRIMARY KEY (login);


ALTER TABLE ipsValidos
    ADD FOREIGN KEY (login)
                REFERENCES passwd  (login)
                ON DELETE RESTRICT
                ON UPDATE RESTRICT;
```

Apêndice F - Importação de certificados pelo browser Internet Explorer 6.0

a) Certificado da CA:

- Abrir o browser
- Escolher a opção Tools >> Internet Options
- Clicar na guia Contents
- Escolher a opção Certificates
- Clicar na guia Trusted Root Certification Authorities
- Clicar em Import
- Clicar em Next
- Escolher o certificado da CA
- Clicar em Next 2 vezes

b) Certificado do usuário:

- Abrir o browser
- Escolher a opção Tools >> Internet Options
- Clicar na guia Contents
- Escolher a opção Certificates
- Clicar na guia Personal
- Clicar em Import
- Clicar em Next
- Escolher o certificado pessoal do usuário
- Clicar em Next
- Digitar a Private Key do usuário
- Escolher opções de segurança
- Clicar em Next 2 vezes

Apêndice G – Código fonte do isAlive cliente e servidor

a) Código fonte do cliente isAlive:

i. Header (isAliveStation.h)

```
---------------------------------- Início do Arquivo isAliveStation.h ---------------------------
// isAliveStation.h (v0.85) *** Para LINUX/UNIX e Windows98/2k/XP (com CYGWIN) ***
// Author:        Bruno Astuto Arouche Nunes
// Date:28-01-2002
// 2003/1
//-----------------------------------------------------------------------

#include <stdlib.h>
#include <fstream.h>
#include <sys/wait.h>
#include <signal.h>
#include <openssl/evp.h>
#include "tcp_lib.h"

typedef void SIGFUNC(int);

#define VERBOSE                                 1
#define KEY_SIZE                        16
#define PORTNUMBER                              56789
#define TOKEN                                   "="
#define ACK_MSG                                 "200 ACK"
#define CONF_FILE_NAME                  "isAlive.conf"
#define CONFIG_PARAMETER_PASSWD                 "PASSWD"
#define CONFIG_PARAMETER_STAADDR        "STAADDR"

// ****** PROTOTYPES ****** //
ulong answerProbe(unsigned char *firstKey, int new_fd);
ulong initializeSingleTcpServer(ushort port, int *socketDescriptor, int verbose);
```

```
static unsigned char key[DATA_SIZE];
void sigchld_handler(int signo);
SIGFUNC *signal(int signo, SIGFUNC *func);
---------------------------------- Fim do Arquivo isAliveStation.h ----------------------------
```

ii. cpp (isAliveClient.cpp)

```
------------------------------- Início do arquivo isAliveStation.cpp -----------------------------
// isAliveStation.cpp (v0.85) *** Para LINUX/UNIX e Windows98/2k/XP (com CYGWIN)
***
// Author:        Bruno Astuto Arouche Nunes & Demetrio Carrion
// Date:06-02-2003
// 2003/1
//
// g++ isAliveStation.cpp tcp_lib.cpp -o isAliveStation -lcrypto
//--------------------------------------------------------------------

#include "isAliveStation.h"

// APAGAR ESSAS LINHAS ======================
char xxx[3];
// APAGAR ESSAS LINHAS ======================

int main(int argc, char **argv)
{
        ulong retCode = OK;
        int fildes[2];
        int socketDescriptor = 0;
   int sin_size = 0;
        int new_fd = 0;
        pid_t childPid;
        struct sigaction sa;
   struct sockaddr_in client_addr; //Hold the destination addr
        char *p;
```

```
        char configParamName[DATA_SIZE] = "\0"; memset(configParamName, '\0',
DATA_SIZE);
        char configLine[DATA_SIZE]                  = "\0"; memset(configLine, '\0',
DATA_SIZE);

        memset(key, '\0', KEY_SIZE);

        // Prepair to read the configuration file.-------------------------
        ifstream file(CONF_FILE_NAME); // open file
        if( !file.is_open() )
        {
                cout << "Error while opening the configuration file: " << CONF_FILE_NAME
<< endl;
                cout << "Could not open the file: " << CONF_FILE_NAME << endl;
                exit(-1);
        }
        while( !file.eof() && !file.fail() ) // beginning while2
        {
                file >> configLine;
                /* For each configuration parameters in the
                  configuration file, there must be a test (ifs)
                  to read the value of the parameter.*/
                // PASSWD if = test the string from file to see if
                // it is the information about the passwd.
                if( strstr(configLine, CONFIG_PARAMETER_PASSWD) )
                {//if xxx
                        /*strtok places a NULL terminator
                        in front of the token "=", if found.*/
                        p = strtok(configLine, TOKEN);
                        if(p) strcpy(configParamName,p);

                        /*A second call to strtok using a NULL
                        as the first parameter, returns a pointer
                        the character following the token.*/
```

```
                    p = strtok(NULL, TOKEN);
                    if(p) strcpy((char*)key, p);

                    cout << "Station FirstKey -> " << key << " <- " << endl;
                    cout << endl;
                    //break;
                }//end of if xxx
        }// end of the while2

        file.close(); // Close configuration file.

        if( (retCode = initializeSingleTcpServer(PORTNUMBER, &socketDescriptor, 1)) < 0)
        {
                cout << "ERRO_1!" << endl;
                cout << tcpipErrorMsg [retCode] << endl;
                return(retCode);
        }

        signal(SIGCHLD, sigchld_handler);//sigchld_handler  SIG_IGN);

        while(1)
        {
                //sleep(1);
                cout << "1::::::::::::::::::::::::Begin of the while!!!" << endl;
                fflush(stdout);
                sin_size = sizeof(client_addr);
      if ((new_fd = accept(socketDescriptor, (struct sockaddr *) &client_addr, &sin_size)) <
0)
      {
                    continue;       /* back of for() */
                    /*if(errno == EINTR)
                    {
                            cout << "=> Error1 in ACCEPT function!" << endl;
                            cout << "=> errno = " << EINTR << endl;
```

```
                        }
                        else
                        {
                                cout << "=> Error2 in ACCEPT function!" << endl;
                                continue;
                                //close(socketDescriptor);
                                //err_sys("accept error");
                                //return(ERROR_ACCEPT);
                        }*/

        }

        printf("TCP/IP server      : Connection accepted from %s\n",
inet_ntoa(client_addr.sin_addr));
                        fflush(stdout);

                        if( pipe(fildes) )
                        {
                                cout << "=> Could not open PIPE!" << retCode << endl;
                                break;
                        }

//              cout << ":999999!!!AAAAAAAAAAAAAAAAAA!!!:" << endl;
//              gets(xxx);

                        if( !(childPid = fork()) )
                        {

//              cout << ":000000!!!AAAAAAAAAAAAAAAAAA!!!:" << endl;
//              gets(xxx);

                                // this won't be used any more ****
                                close(socketDescriptor);
```

```
                cout << "PID = " << childPid << endl;
                fflush(stdout);

                retCode = answerProbe(key, new_fd);
                write(fildes[1], key, DATA_SIZE);

//              cout << "::::::::::::::::::::::::::::::::::::::::" << sin_size << endl;
//              cout << "::::::::::::::::::::::::::::::::::::::::" << sizeof(new_fd) << endl;
        //      gets(xxx);

                if(retCode)
                {
                        cout << "Prober closed with error:" << retCode << endl;
                        close(new_fd);
                        exit(-1);
                        //break;
                }

//              cout << "2::::::::::::::::::::::::::::::::::::::::" << endl;
                close(new_fd);
                fflush(stdout);

                // Terminate child ********
                exit(0);

        }//end of if fork()

//              cout << "3::::::::::::::::::::::::::::::::::::::::" << endl;
        fflush(stdout);
        read(fildes[0], key, DATA_SIZE);
        close(new_fd);

//              cout << "4::::::::::::::::::::::::::::::::::::::::" << endl;
```

```
        }// END OF WHILE(1)...

//      cout << "::::::::::::::::::::::::::::::::::::::::" << endl;
//      gets(xxx);

        cout << "\n\n *** THE IS_ALIVE_STATION IS NOW DEAD!!! *** \n" << endl;
        cout << "\n\n *** HAVE A NICE DAY!!! *** \n" << endl;
        return (retCode);

}// END OF main()
//=================================================================
=================

//=================================================================
=================
// This function cllas the POSIX sigaction function(needed to avoid zombies)
/*SIGFUNC *signal(int signo, SIGFUNC *func)
{
        struct sigaction act, oact;
        act.sa_handler = func;          // reap all dead processes
        sigemptyset(&act.sa_mask);
        act.sa_flags = 0;

        if(signo == SIGALRM)
        {
                #ifdef SA_INTERRUPT
                act.sa_flags = SA_INTERRUPT;      /* SunOS 4.x /
                #endif
        }
        else
        {
                #ifdef SA_RESTART
                act.sa_flags = SA_RESTART;                /* SVR4, 4.4BSD /
```

```
		#endif
	}

	if( sigaction(signo, &act, &oact) < 0 )
	{
		cout << "=> Could not reap dead processes!" << endl;
		return(SIG_ERR);
	}

	return(oact.sa_handler);
}*/
//=================================================================
=================

//=================================================================
=================
// This function handles SIGCHLD signal (needed to avoid zombies)
void sigchld_handler(int signo)
{
	pid_t pid;
	int stat;

	while( (pid = waitpid(-1, &stat, WNOHANG)) > 0)
	{
		cout << "===> Child " << pid << " terminated!" << endl;
		//cout << "===> childPid " << childPid << " terminated!" << endl;
	}
	return;
}
//=================================================================
=================

//=================================================================
=================
```

```
// This function probes the station
ulong answerProbe(unsigned char *firstKey, int new_fd)
{

//        cout << ":11111!!!AAAAAAAAAAAAAAAAA!!!:" << endl;
//        gets(xxx);

/*  CRYPT  ----------------------------------------------------*/
        int outlen = 0;
        int tmplen = 0;
        //unsigned char key[] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
        unsigned char iv[] = {1,2,3,4,5,6,7,8};
        EVP_CIPHER_CTX ctx;
/*  CRYPT  ----------------------------------------------------*/
        ulong retCode = OK;
        int bytes_recved = 0;
        int bytes_sent = 0;             // how many bytes we´ve sent at that moment

        byte *txmsg = 0;
        txmsg = new byte[DATA_SIZE];
        memset(txmsg, '\0', DATA_SIZE);

        byte *rxmsgDecrypted = 0;
        rxmsgDecrypted = new byte[DATA_SIZE];
        memset(rxmsgDecrypted, '\0', DATA_SIZE);

        byte *txmsgCrypted;
        txmsgCrypted = new byte[DATA_SIZE];
        memset(txmsgCrypted, '\0', DATA_SIZE);

        byte *rxmsgCrypted = 0;
        rxmsgCrypted = new byte[DATA_SIZE];
        memset(rxmsgCrypted, '\0', DATA_SIZE);
```

```
        cout << "FirstKey ->:" << firstKey << ":<-" << endl;
        cout << "NEW_FD   ->:" << new_fd   << ":<-" << endl;

        fflush(stdout);
        cout << "==========================================" << endl;

        // Recive first msg from the isAliveDaemon
        if( ( bytes_recved = recv(new_fd, rxmsgCrypted, DATA_SIZE, 0) ) < 0 )
                return(ERROR_REVC);


//      cout << ":333333!!!AAAAAAAAAAAAAAAAA!!!:" << endl;
//      gets(xxx);

//      cout << "XXX:" << rxmsgCrypted << ":" << endl;

// Decrypt message recived from isAliveDaemon --------------------------------

        cout << "==>bytes_recved:" << bytes_recved << endl;

        EVP_CIPHER_CTX_init(&ctx);
        EVP_DecryptInit_ex(&ctx, (EVP_CIPHER*)EVP_bf_cbc(), NULL, firstKey, iv);
        if( !EVP_DecryptUpdate(&ctx, rxmsgDecrypted, &outlen, rxmsgCrypted, 24) )
        {
                /* Error */
                cout << "Error in EVP_DecryptUpdate()!" << endl;
                return 1;
        }
        /* Buffer passed to EVP_EncryptFinal() must be after data just
         * encrypted to avoid overwriting it.*/
        EVP_CIPHER_CTX_set_padding(&ctx, 0);
        if(!EVP_DecryptFinal_ex(&ctx, rxmsgDecrypted + outlen, &tmplen))
        {
```

```
            /* Error */
            cout << "Error in EVP_DecryptFinal_ex()!" << endl;
            return 1;
        }
        outlen += tmplen;
        EVP_CIPHER_CTX_cleanup(&ctx);
        /* Need binary mode for fopen because encrypted data is
         * binary data. Also cannot use strlen() on it because
         * it wont be null terminated and may contain embedded
         * nulls.
         */
// END OF Decrypt message recived from isAliveDaemon -------------------------

        fflush(stdout);
        cout << "strlen(rxmsgDecrypted): " << strlen((char*)rxmsgDecrypted) << endl;
        fflush(stdout);
        cout << "outlen            : " <<  outlen  << endl;
        fflush(stdout);
        cout << "\nrxmsgCrypted       :"  << rxmsgCrypted  << ":" << endl;
        fflush(stdout);
        cout << "rxmsgDecrypted     :"  << rxmsgDecrypted << ":" << endl;
        fflush(stdout);
        cout << "Decrypted message  :" << rxmsgDecrypted << endl;
        fflush(stdout);
        cout << "OLD KEY            :" << key << endl;
        fflush(stdout);
        memmove(key, rxmsgDecrypted, KEY_SIZE);
        cout << "rxmsgDecrypted     :" << rxmsgDecrypted << endl;
        fflush(stdout);
        cout << "NEW KEY            :" << key << endl;

        memset(txmsg, '\0', DATA_SIZE);
        memset(txmsgCrypted, '\0', DATA_SIZE);
        strcpy((char*)txmsg, ACK_MSG);
```

```
        strcat((char*)txmsg, "\0");  //0x00;  //EOS

// Encrypt message to be sent to isAliveStation ----------------------------------------
        EVP_CIPHER_CTX_init(&ctx);
        EVP_CIPHER_CTX_set_padding(&ctx, 1);
        EVP_EncryptInit_ex(&ctx, (EVP_CIPHER*)EVP_bf_cbc(), NULL, key, iv);
        if(!EVP_EncryptUpdate(&ctx, txmsgCrypted, &outlen, (unsigned char*)txmsg,
strlen((char*)txmsg)))
        {
                /* Error */
                return 1;
        }
        /* Buffer passed to EVP_EncryptFinal() must be after data just
         * encrypted to avoid overwriting it.
         */
        if(!EVP_EncryptFinal_ex(&ctx, txmsgCrypted + outlen, &tmplen))
        {
                /* Error */
                return 1;
        }
        outlen += tmplen;
        EVP_CIPHER_CTX_cleanup(&ctx);
        /* Need binary mode for fopen because encrypted data is
         * binary data. Also cannot use strlen() on it because
         * it wont be null terminated and may contain embedded
         * nulls.
         */
// END OF Encrypt message to be sent to isAliveStation ------------------

        cout << "==>txmsg=ACK_MSG:" << txmsg << ":" << endl;
        cout << "==>txmsgCrypted:" << txmsgCrypted << ":" << endl;


/*      cout << "Decrypted message  :" << newKey << endl;
```

```
        cout << "USED KEY          :" << key << endl;
        cout << "txmsgCrypted      :" << txmsgCrypted << endl;
        cout << "outlen            :" << outlen << endl;
*/

        // sending encrypted ACK message to the client
        bytes_sent = send(new_fd, txmsgCrypted, outlen, 0);
        if (bytes_sent < 0)
        {
                cout << tcpipErrorMsg [retCode] << endl;
                return(ERROR_SEND);
        }

        // printing message sent to isAliveDaemon
        cout << "Message sent to prober:" << txmsgCrypted << endl;//myServer.getTXmsg(0)
<< endl;

        cout << "========================================" << endl;

        fflush(stdout);
        delete [] txmsg;
        delete [] rxmsgDecrypted;
        delete [] txmsgCrypted;
        delete [] rxmsgCrypted;

//      cout << ":88888!!!AAAAAAAAAAAAAAAAA!!!:" << endl;
//      gets(xxx);

        return(OK);

}// END OF answerProbe()
//================================================================
=================
```

```
//=================================================================
==================
// This function initializes a tcp server calling socket, bind, listen
ulong initializeSingleTcpServer(ushort port, int *socketDescriptor, int verbose)
{
	struct  sockaddr_in server_addr; //Hold the destination addr
	struct  hostent *h; //Hold the server information
	char    serverAddress[SIZE_DATA];
	int     sock_fd;
	int     yes = 1;

	#if DEBUG
		printf("DEBUG: IN initializeSingleTcpServer FUNCTION \n");
	#endif

	if((gethostname(serverAddress, sizeof(serverAddress))) < 0) // get the host name
		return(ERROR_GETHOSTNAME);

	if((h = gethostbyname(serverAddress)) == NULL) // get the host info
		return(ERROR_GETHOSTBYNAME);

	// Setting structure parameters ************************
	server_addr.sin_family = SOCK_FAMILY;
	server_addr.sin_port   = htons(port);
	server_addr.sin_addr   = *((struct in_addr *)h->h_addr);
	memset(&(server_addr.sin_zero), '\0', 8); // zero to the rest of the structure
	// Setting structure parameters ********END***************

	// Opening socket ************************
	if ((sock_fd = socket(SOCK_FAMILY, SOCK_TYPE, SOCK_PROTOCOL)) < 0)
		return(ERROR_OPEN_SOCKET);

	printf("TCP/IP server      : Socket Opened ... \n");
```

```
	// Forcing the port to be released ********
	// Lose the pesky "Address alredy in use" error message **********
	setsockopt(sock_fd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int));

	// Binding to port ********************
	if ((bind(sock_fd, (struct sockaddr *) &server_addr, sizeof(server_addr))) < 0)
	{
		close(sock_fd);
		return(ERROR_BIND);
	}

	printf("TCP/IP server	: Bind to port %d ...\n", port);

	// Listenig to port ********************
	if ((listen(sock_fd, BACKLOGSINGLE)) < 0)
	{
		close(sock_fd);
		return(ERROR_LISTEN);
	}

	printf("TCP/IP server	: Listening port %d ...\n", port);
	printf("Server Name (IP)   : %s (%s)\n\n", h->h_name, inet_ntoa(*((struct in_addr *)h-
>h_addr)));

			*socketDescriptor = sock_fd;

	return(OK);
}// END OF initializeSingleTcpServer

------------------------------- Fim do arquivo isAliveStation.cpp --------------------------------
```

b) Código fonte do servidor isAlive:

i. Header (isAliveDaemon.cpp)

```
------------------------------- Início do arquivo isAliveDaemon.cpp ----------------------------
// isAliveDaemon.cpp (v0.8) *** Apenas para SOs UNIXlike ***
// Author:      Bruno Astuto Arouche Nunes & Demetrio Carrion
// Date:14-01-2002
// 2002/2
//
// Compilando
// g++ -I/usr/local/include/ -I/usr/local/include/mysql -O2 -c isAliveDaemon.cpp tcp_lib.cpp
//
// Linkando
// g++ -g -O2 -L/usr/local/lib/mysql -o isAliveDaemon isAliveDaemon.o -L/usr/local/lib/
//                  -lsqlplus -lz -lmysqlclient -lz -lmysqlclient tcp_lib.cpp
//-------------------------------------------------------------------------------------

#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <fstream.h>
#include <iostream.h>
#include <iomanip.h>
#include <openssl/evp.h>
#include "sqlplus.hh"
#include "tcp_lib.h"

#define VERBOSE                 1
#define PROBE_ERROR             13
#define UPDATE_ERROR    17
#define TIME_TO_WAIT    3
#define PORTNUMBER              56789
#define KEY_SIZE                16
#define LINE_SIZE               4096
#define IN_FILE_NAME    "/var/www/htdocs/regras.conf"
```

```
#define OUT_FILE_NAME   "/var/www/htdocs/pf.conf"
//#define IN_FILE_NAME    "regras.conf"
//#define OUT_FILE_NAME"pf.conf"
#define ACK_MSG                 "200 ACK"
#define DB_NAME                 "strikein"
#define DB_HOST                 "10.10.0.15"
#define DB_USER                 "wstrike"
#define DB_PASSWD               "wstrike"

// This function is responsable for the connection with the STAs.
ulong probeStation(char *serverAddr, int port, unsigned char *key, char *tableName);

// This function updates the Packet Filter's conf string.
ulong updatePfConfString(char *badStation);

// This function updates the Packet Filter's conf file.
ulong updatePfFile(char *updateString);

// Reload Packet Filter's conf file.
ulong reloadPfConf();

// Key generator : this is not the best way to do this.
// it would be nice to find another way of generating keys.
//-------------------------------------------------------
void seedGen();
char* getKey();
//-------------------------------------------------------

int main (int argc, char **argv)
{
        ulong size = 0;
        char tableName[33];
        strcpy(tableName,"ipsValidos");
```

```
	// Generate the seed for srand to generate the key.
	seedGen();

	while(1) // beginning main while
	{
		// Prepair to read the stations addrs from the data base.
		try{
			// Connection(cchar *db, cchar *host="",
			//		cchar *user="", cchar *passwd="")
			Connection con(DB_NAME, DB_HOST, DB_USER, DB_PASSWD);
			Query query = con.query();
			// This creates a query object that is bound to con.
			query << "SELECT * FROM " << tableName;
			// You can write to the query object like you would any other ostrem
			cout << "Query: " << query.preview() << endl;
			// Query::preview() simply returns a string with the current query
			// string in it.
			Result res = query.store();
			// Query::store() executes the query and returns the results
			//cout << "Records Found: " << res.size() << endl << endl;

			Row row;

			Result::iterator i;
			// The Result class has a read-only Random Access Iterator
			for (i = res.begin(); i != res.end(); i++)
			{
				row = *i;
				cout << "Probing station -> " << row[2].c_str() << endl;
				if( !probeStation((char *)row[2].c_str(),
					PORTNUMBER, (unsigned char *)row[0].c_str(),
tableName) == 0 )
				{
					// Probe NOT successfull!
```

```
                                cout << "PROBE_ERROR:" << PROBE_ERROR <<
endl;
                                cout << "Unable to probe station:" << row[2].c_str() <<
endl;


        /*************************************************************
                                TO DO: Something must be done here in case the
update fail!

        *************************************************************/
                                updatePfConfString((char *)row[2].c_str());
                                reloadPfConf();

                                // Remove the Station from the data base.
                                query << "DELETE FROM " << tableName
                                        << " WHERE enderecoIP='" << row[2].c_str()
<< "'";
                                cout << "Query: " << query.preview() << endl;
                                Result res = query.store();

                                cout << "Station " << row[2].c_str() << " have been
dumped!\n" << endl;
                        }
                }
                cout << endl;
        }catch(BadQuery &er) { // handle any connection or
                                                        // query errors that may come up
        #ifdef USE_STANDARD_EXCEPTION
                cerr << "Error: " << er.what() << endl;
        #else
                cerr << "Error: " << er.error << endl;
        #endif
                return -1;
```

```
                } catch (BadConversion &er) { // handle bad conversions
                #ifdef USE_STANDARD_EXCEPTION
                        cerr << "Error: " << er.what() << "\"." << endl
                                << "retrieved data size: " << er.retrieved
                                << " actual data size: " << er.actual_size << endl;
                #else
                        cerr << "Error: Tried to convert \"" << er.data << "\" to a \""
                                << er.type_name << "\"." << endl;
                #endif
                        return -1;
                #ifdef USE_STANDARD_EXCEPTION
                } catch (exception &er) {
                        cerr << "Error: " << er.what() << endl;
                        return -1;
                #endif
                }
                //-----------------------------------------------------------------

                sleep(TIME_TO_WAIT);
        }// end of the main while

        cout << "\n\n *** THE IS_ALIVE_DAEMON IS NOW DEAD!!! *** \n" << endl;
        cout << "\n\n *** HAVE A NICE DAY!!! *** \n" << endl;
        return(OK);

}// END OF main()


//-------------------------------------------------------------
// Key generator : this is not the best way to do this.
// it would be nice to find another way of generating keys.
void seedGen()
{
        time_t now;
```

```
        time(&now);
        srand((unsigned int)now);
}
// end of the keyGen function
//------------------------------------------------------------

//------------------------------------------------------------
char* getKey()
{
        static char keyToGet[KEY_SIZE + 1] = "\0"; memset(keyToGet, '\0', KEY_SIZE);
        static char keyTemp[KEY_SIZE + 1] = "\0"; memset(keyTemp, '\0', KEY_SIZE);

        sprintf(keyToGet,"%d",rand());

        while( strlen(keyToGet) < KEY_SIZE )
        {
//              cout<< "*** < 16 ***";
                sprintf(keyTemp,"%d",rand());
                strncat(keyToGet, keyTemp, KEY_SIZE-strlen(keyToGet));
        }

        return(keyToGet);
}
// end of the keyGen function
//------------------------------------------------------------

//------------------------------------------------------------
// This function updates the Packet Filter's conf file.
ulong updatePfConfString(char *badStation)
{
        char tableName[33];
        strcpy(tableName,"ipsValidos");

        char fileLine[LINE_SIZE];
```

```
memset(fileLine, '\0', LINE_SIZE);

// Prepair to read the stations addrs from the data base.
try{
        Connection con(DB_NAME, DB_HOST, DB_USER, DB_PASSWD);
        Query query = con.query();
        // read all IPs except the one to be excluded
        // from the PF conf file.
        query << "SELECT * FROM " << tableName
                << " WHERE enderecoIP<>'" << badStation << "'";
        cout << "Query: " << query.preview() << endl;
        Result res = query.store();
        Row row;

        Result::iterator i;
        i = res.begin();

        strcpy(fileLine,"IPS = '{ ");
        // The Result class has a read-only Random Access Iterator
        while( (i != res.end()) )
        {
                row = *i;
                strcat(fileLine, row[2].c_str());
                i++;
                strcat(fileLine, "/32, ");
        }

        strcat(fileLine, "127.0.0.1/32 }'\n\0");

        cout << "IPs LINE: " << fileLine << endl;

}catch(BadQuery &er) { // handle any connection or
                                        // query errors that may come up
#ifdef USE_STANDARD_EXCEPTION
```

```
            cerr << "Error: " << er.what() << endl;
        #else
            cerr << "Error: " << er.error << endl;
        #endif
            return (UPDATE_ERROR);
          } catch (BadConversion &er) { // handle bad conversions
        #ifdef USE_STANDARD_EXCEPTION
            cerr << "Error: " << er.what() << "\"." << endl
                 << "retrieved data size: " << er.retrieved
                 << " actual data size: " << er.actual_size << endl;
        #else
            cerr << "Error: Tried to convert \"" << er.data << "\" to a \""
                 << er.type_name << "\"." << endl;
        #endif
            return (UPDATE_ERROR);
        #ifdef USE_STANDARD_EXCEPTION
          } catch (exception &er) {
            cerr << "Error: " << er.what() << endl;
            return (UPDATE_ERROR);
        #endif
          }
            //-----------------------------------------------------------------

        /*************************************************************
        TO DO: Something must be done here in case the update fail!
        *************************************************************/
        if ( !updatePfFile(fileLine) )
            return(OK);
        else
            return(UPDATE_ERROR);
}
//-------------------------------------------------------------

//-------------------------------------------------------------
```

```
// Reload Packet Filter's conf file.
ulong reloadPfConf()
{
	char systemString[DATA_SIZE];
	memset(systemString, '\0', DATA_SIZE);

	strcpy(systemString, "pfctl -f ");
	strcat(systemString, OUT_FILE_NAME);
	system(systemString);

	cout << "Reloading PF's configuration file: " << systemString << endl;

}
//-------------------------------------------------------------

// This function updates the Packet Filter's conf file.
ulong updatePfFile(char *updateString)
{
	char fileLine[LINE_SIZE];
	memset(fileLine, '\0', LINE_SIZE);

	ifstream infile(IN_FILE_NAME, ios::nocreate); // open file
	if( !infile.is_open() )
	{
		cout << "\n*** ERROR while opening file " << IN_FILE_NAME
			<< " for reading! *** \n" << endl;
		return(UPDATE_ERROR);
	}

	ofstream outfile(OUT_FILE_NAME); // open file
	if( !outfile.is_open() )
	{
		cout << "\n*** ERROR while opening file " << OUT_FILE_NAME
			<< " for reading! *** \n" << endl;
```

```
                return(UPDATE_ERROR);
        }

        outfile << updateString;

        while(!infile.eof() && !infile.fail() && !outfile.fail())
        {
                memset(fileLine, '\0', LINE_SIZE);
      infile.getline(fileLine, LINE_SIZE);

                //cout << "AAAAAAAAAAAAAAAAAAAAAAAAA:" << strlen(fileLine)
<< endl;

                outfile << fileLine << "\n";

        }//end of the WHILE

        infile.close();
        outfile.close();

        return(OK);
}

//-------------------------------------------------------------
// This function is responsable for the connection with the STAs.
ulong probeStation(char *serverAddr, int port, unsigned char *key, char *tableName)
{
/*  CRYPT  ----------------------------------------------------*/
        EVP_CIPHER *alg;
        alg = (EVP_CIPHER*)EVP_bf_cbc();
        int outlen = 0;
        int tmplen = 0;
        //unsigned char key[] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
        unsigned char iv[] = {1,2,3,4,5,6,7,8};
```

```
        EVP_CIPHER_CTX ctx;
/*  CRYPT  ------------------------------------------------*/
        ulong retCode;
        int bytes_recved;
        char rxmsgDecrypted [DATA_SIZE + EVP_CIPHER_block_size(alg)];
        memset(rxmsgDecrypted, '\0', DATA_SIZE + EVP_CIPHER_block_size(alg));
        char txmsg[DATA_SIZE] = "\0"; memset(txmsg, '\0', DATA_SIZE);
        char rxmsg[DATA_SIZE] = "\0"; memset(rxmsg, '\0', DATA_SIZE);
        char newKey[DATA_SIZE] = "\0"; memset(newKey, '\0', DATA_SIZE);
        unsigned char txmsgCrypted[1024 + EVP_CIPHER_block_size(alg)];
        memset(txmsgCrypted, '\0', 1024 + EVP_CIPHER_block_size(alg));
        byte rxmsgCrypted [DATA_SIZE] = "\0";    memset(rxmsgCrypted, '\0',
DATA_SIZE);

        // ----- Write and read from database --------------------------------
        try{
                Connection con(DB_NAME, DB_HOST, DB_USER, DB_PASSWD);
                Query query = con.query();

                // Change the key on the data base for the next probe.
                query << "UPDATE " << tableName << " SET senhaAtual='" << getKey()
                        << "' WHERE enderecoIP='" << serverAddr << "'";
                cout << "Query: " << query.preview() << endl;
                Result res = query.store();

                // Read the new key from the data base to send it to the station.
                query << "SELECT senhaAtual FROM " << tableName << " WHERE
enderecoIP='" << serverAddr << "'";
                cout << "Query: " << query.preview() << endl;
                res = query.store();
                Row row;

                Result::iterator i;
                i = res.begin();
```

```
            row = *i;
            cout << "THE NEW_KEY >> " << row[0].c_str() << endl;
            cout << "THE KEY     >> " << key << endl;
            strcpy(newKey, row[0].c_str());


            fflush(stdout);

        }catch(BadQuery &er) { // handle any connection or
        // query errors that may come up
        #ifdef USE_STANDARD_EXCEPTION
            cerr << "Error: " << er.what() << endl;
        #else
            cerr << "Error: " << er.error << endl;
        #endif
            return (PROBE_ERROR);
        } catch (BadConversion &er) { // handle bad conversions
        #ifdef USE_STANDARD_EXCEPTION
            cerr << "Error: " << er.what() << "\"." << endl
                    << "retrieved data size: " << er.retrieved
                    << " actual data size: " << er.actual_size << endl;
        #else
            cerr << "Error: Tried to convert \"" << er.data << "\" to a \""
                    << er.type_name << "\"." << endl;
        #endif
            return (PROBE_ERROR);
        #ifdef USE_STANDARD_EXCEPTION
        } catch (exception &er) {
            cerr << "Error: " << er.what() << endl;
            return (PROBE_ERROR);
        #endif
        }
// ----- Write and read from database ----------END OF----------
```

```
        if(newKey == NULL) { return(PROBE_ERROR); }

// Initialize the connection with the Station ----------------------------
        tcpClient myClient(serverAddr, port, VERBOSE);

        if ( (retCode = myClient.getConstructorError()) )
        {
                if(VERBOSE)
                {
                        cout << "ERRO_1!" << endl;
                        cout << tcpipErrorMsg [retCode] << endl;
                }
                return(retCode);
        }

        if ( (retCode = myClient.initializeClientConnection()) )
        {
                if(VERBOSE)
                {
                        cout << "ERRO_2!" << endl;
                        cout << tcpipErrorMsg [retCode] << endl;
                }
                return(retCode);
        }

        cout << "Station ID: "<< myClient.getNameOfTheHost(0) << " ("
                << myClient.getIpOfTheHost(0) << ")" << endl;
// END OF Initialize the connection with the Station ------------------------

        cout << "\n=========================================" << endl;

        #ifdef WIN32
                int i = 0;
                while(i < DATA_SIZE)
```

```
                {
                        txmsg[i] = '\0';
                        i++;
                }
        #else
                bzero(txmsg, DATA_SIZE);
        #endif

        /*strcpy(txmsg,"200 sending newKey(");
        strcat(txmsg, newKey);
        strcat(txmsg, ") encrypted with key(");
        strcat(txmsg, key);
        strcat(txmsg, ").");*/
                        //cout << "AAAAAAAAAAAAAAAAAAAAAAAAAA" << endl;
// Encrypt message to be sent to isAliveStation  -------------------------
        EVP_CIPHER_CTX_init(&ctx);
        EVP_CIPHER_CTX_set_padding(&ctx, 1);
        EVP_EncryptInit_ex(&ctx, alg, NULL, key, iv);
        if(!EVP_EncryptUpdate(&ctx, txmsgCrypted, &outlen, (unsigned char*)newKey,
strlen(newKey)))
        {
                /* Error */
                return 1;
        }
        /* Buffer passed to EVP_EncryptFinal() must be after data just
         * encrypted to avoid overwriting it.
         */
        if(!EVP_EncryptFinal_ex(&ctx, txmsgCrypted + outlen, &tmplen))
        {
                /* Error */
                return 1;
        }
        outlen += tmplen;
        EVP_CIPHER_CTX_cleanup(&ctx);
```

```
        /* Need binary mode for fopen because encrypted data is
         * binary data. Also cannot use strlen() on it because
         * it wont be null terminated and may contain embedded
         * nulls.
         */
/*
        cout << "\nnewKey:" << newKey << ":" << endl;
        cout << "\ntxmsgCrypted:" << txmsgCrypted << ":" << endl;
*/

        cout << "200 sending newKey(" << newKey << ") encrypted with key(" << key <<
")." << endl;
/*      cout << "Decrypted message  :" << newKey << endl;
        cout << "USED KEY          :" << key << endl;
        cout << "txmsgCrypted      :" << txmsgCrypted << endl;
        cout << "outlen            :" << outlen << endl;
*/

// END OF Encrypt message to be sent to isAliveStation ----------

        if ( (retCode = myClient.sendData(txmsgCrypted, 24)) )
        {
                if(VERBOSE)
                {
                        cout << tcpipErrorMsg [retCode] << endl;
                }
                return(retCode);
        }
        // printing message sent to server
        cout << "Message sent to " << myClient.getNameOfTheHost(0)
                << ": " << myClient.getTXmsg(0) << endl;

//cout << "CCCCCCCCCCCCCCCCCCCCCCCCc" << endl;
```

```
        // Recive first msg from server
        if( ( bytes_recved = recv(myClient.socketDescriptor, rxmsgCrypted, DATA_SIZE, 0)
) < 0 )
                return(ERROR_REVC);

// Decrypt message recived from isAliveDaemon --------------------------------

        cout << "==>bytes_recved:" << bytes_recved << endl;

        EVP_CIPHER_CTX_init(&ctx);
        EVP_DecryptInit_ex(&ctx, (EVP_CIPHER*)EVP_bf_cbc(), NULL, (byte*)newKey,
iv);
        if( !EVP_DecryptUpdate(&ctx, (byte*)rxmsgDecrypted, &outlen, rxmsgCrypted,
bytes_recved) )
        {
                /* Error */
                cout << "Error in EVP_DecryptUpdate()!" << endl;
                return 1;
        }
        /* Buffer passed to EVP_EncryptFinal() must be after data just
         * encrypted to avoid overwriting it.*/
        EVP_CIPHER_CTX_set_padding(&ctx, 0);
        if(!EVP_DecryptFinal_ex(&ctx, (byte*)rxmsgDecrypted + outlen, &tmplen))
        {
                /* Error */
                cout << "Error in EVP_DecryptFinal_ex()!" << endl;
                return 1;
        }
        outlen += tmplen;
        EVP_CIPHER_CTX_cleanup(&ctx);
        /* Need binary mode for fopen because encrypted data is
         * binary data. Also cannot use strlen() on it because
         * it wont be null terminated and may contain embedded
         * nulls.
```

```
        */
// END OF Decrypt message recived from isAliveDaemon -------------------------

                /*
if ( (retCode = myClient.receiveData(DATA_SIZE, &bytes_recved)) )
        {
                if(VERBOSE)
                {
                        cout << tcpipErrorMsg [retCode] << endl;
                }
                return(retCode);
        }*/

        if( !memcmp(ACK_MSG, rxmsgDecrypted, 7) )
        {
                // printing message recived from server
                cout << ">>>Message from " << myClient.getNameOfTheHost(0)
                << ":" << rxmsgDecrypted << endl;
        }
        else
        {
                // printing message recived from server
                cout << ">>>Message from " << myClient.getNameOfTheHost(0)
                << ":" << rxmsgDecrypted << endl;
                cout << ">>>**** ERROR WHILE TRYING TO DECRYPT MESSAGE!!!
****" << endl;
                return (PROBE_ERROR);
        }

        cout << "=========================================" << endl;

        return(OK);

}
```

---------- Fim do arquivo isAliveDaemon.cpp ----------

c) Biblioteca tcp_lib:

i. Header (tcp_lib.h)

---------- Início do arquivo tcp_lib.h ----------

```
// tcp_lib.h (v7.3)
// Bruno Astuto Arouche Nunes
// 2002/2
// 24-10-2002
//---------------------------------------------------------------------

#ifndef _TCPLIB_H_
#define _TCPLIB_H_

#include <iostream.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#ifdef WIN32
	// includes for windows
	#include <winsock2.h>
#else
	// includes for unix
	#include <sys/types.h>
	#include <sys/socket.h>
	#include <netinet/in.h>
```

```
        #include <arpa/inet.h>
        #include <unistd.h>
        #include <netdb.h>
        #include <errno.h>
#endif

#define OK 0
#define ERROR_GETHOSTBYNAME 1
#define ERROR_OPEN_SOCKET 2
#define ERROR_CONNECT 3
#define ERROR_CLOSE 4
#define ERROR_REVC 5
#define ERROR_SEND 6
#define ERROR_BIND 7
#define ERROR_LISTEN 8
#define ERROR_SETSOCKOPT 9  // not used
#define ERROR_ACCEPT 10
#define ERROR_GETHOSTNAME 11
#define ERROR_GETPEERNAME 12

#define SIZE_DATA 1024
#define DATA_SIZE 1024

#define SOCK_FAMILY             AF_INET // socket family
#define SOCK_TYPE          SOCK_STREAM // socket type
#define SOCK_PROTOCOL IPPROTO_TCP // used protocol

#define BACKLOG       5  // max number of clients
#define BACKLOGSINGLE 1  // max number of clients

#ifdef WIN32
        #define False false
        #define True true
#else
```

```
        #define FALSE false
        #define False false
        #define TRUE true
        #define True true
#endif


#ifndef byte
#define byte unsigned char
#endif
#ifndef ulong
#define ulong unsigned long
#endif
#ifndef ushort
#define ushort unsigned short
#endif

extern char *tcpipErrorMsg[];

/* ---------- PROTOTYPES ---------- */

//////////////////////////////////////
//                  tcpIp                  //
//////////////////////////////////////
class tcpIp
{
protected: // atributes
        char nameOfTheHost[DATA_SIZE];
        char ipOfTheHost[DATA_SIZE];
        byte rxmsg[DATA_SIZE];
        byte txmsg[DATA_SIZE];
        int retCode;
        int verbose; // flag to dispay messages on screen.
        int constructorError;
```

```
        ushort port;

#ifdef WIN32
protected:
        void callWSAStartup(); // method necessary to the use of WinSocks
#endif

public: // methods
        tcpIp(); // constructor
        ~tcpIp(); // destructor
        void setNameOfTheHost(struct hostent *h);
        void setIpOfTheHost(struct hostent *h);
        char *getNameOfTheHost(int print);
        char *getIpOfTheHost(int print);
        byte *getRXmsg(int print);
        byte *getTXmsg(int print);
        ulong simpleSendData(byte *buffer);
        ulong sendData(byte *buffer, int buflen);
        ulong simpleSendData(char *buffer);
        ulong sendData(char *buffer, int buflen);
        ulong receiveData(ulong maxlength, int *bytes_recved);
        ulong getConstructorError();

        int socketDescriptor;
};// end of tcpIp class


//////////////////////////////////////////
//                    tcpClient                    //
//////////////////////////////////////////
class tcpClient : public tcpIp
{
private: // atributes
        char serverAddress[DATA_SIZE];
```

```
        struct sockaddr_in dest_addr; //Hold the destination addr.

private: // methods
        ulong setTcpClient(char *serverAddr, ushort p, int v = 0);

public: // methods
        tcpClient(char *serverAddr, ushort p, int v); // constructor
        tcpClient(char *serverAddr, char *p, int v); // constructor
        ~tcpClient(); // destructor
        ulong initializeClientConnection();
        ulong endClient();

};// end of tcp_client class


////////////////////////////////////
//                tcpServer                //
////////////////////////////////////
class tcpServer : public tcpIp
{
private: // atributes
        char ipOfThePeer[DATA_SIZE]; //Hold connector's IP address
        char nameOfThePeer[DATA_SIZE]; //Hold connector's name
        int portOfThePeer; //Hold connector's connection port

private: // methods
        ulong setTcpServer(ushort p, int v = 0);

public: // methods
        tcpServer(ushort p, int v); // constructor
        tcpServer(char *p, int v); // constructor
        ~tcpServer(); // destructor
        ulong initializeSingleServer();
        ulong initializeServer();
```

```
        ulong disconectClient();
        ulong setPeerInfo();
        ulong setPeerInfo(struct sockaddr_in peer_addr);
        char *getPeerAddr(int print);
        char *getPeerName(int print);
        int getPeerPort(int print);

        struct sockaddr_in server_addr; //Hold server's addr (my addr)
        struct sockaddr_in client_addr;  //Hold connector's information
        int sock_fd;

};// end of tcp_server class

/* ----------- END OF PROTOTYPES ------------ */

#endif
```

---------- Fim do arquivo tcp_lib.h. ----------

ii. Código fonte (tcp_lib.cpp)

---------- Início do arquivo tcp_lib.cpp ----------

```
// tcp_lib.cpp (v7.3)
// Bruno Astuto Arouche Nunes
// 2002/2
// 28-11-2002
//---------------------------------------------------------------------

#include "tcp_lib.h"

#define DEBUG                   0
#define BACKLOG                 5  // max number of clients
#define BACKLOGSINGLE 1  // max number of clients
```

```
char *tcpipErrorMsg[] =
{
        "OK",
        "TCP/IP    : ERROR IN gethostbyname FUNCTION ...\n",
        "TCP/IP    : ERROR IN socket FUNCTION ...\n",
        "TCP/IP    : ERROR IN connect FUNCTION ...\n",
        "TCP/IP    : ERROR IN close FUNCTION ...\n",
        "TCP/IP    : ERROR IN recv FUNCTION ...\n",
        "TCP/IP    : ERROR IN send FUNCTION ...\n",
        "TCP/IP    : ERROR IN bind FUNCTION ...\n",
        "TCP/IP    : ERROR IN listen FUNCTION ...\n",
        "TCP/IP    : ERROR IN setsockopt FUNCTION ...\n",
        "TCP/IP    : ERROR IN accept FUNCTION ...\n",
        "TCP/IP    : ERROR IN gethostname FUNCTION ...\n",
        "TCP/IP    : ERROR IN getpeername FUNCTION ...\n"
};

/*****************************************************/
/*                    tcpIp class functions                    */
/*****************************************************/

// constructor of the tcpIp class
tcpIp::tcpIp()
{
        #if DEBUG
                cout << "DEBUG: IN THE CONSTRUCTOR OF tcpIp::tcpIp()!" << endl;
        #endif

        constructorError = OK;
//      nameOfTheHost = 0;
//      ipOfTheHost = 0;
//      rxmsg = 0;
//      txmsg = 0;
}
```

```
// destructor of the tcpIp class
tcpIp::~tcpIp()
{
	#if DEBUG
		cout << "DEBUG: IN THE DESTRUCTOR OF tcpIp::tcpIp()!" << endl;
	#endif
	/*
	nameOfTheHost;
	ipOfTheHost;
	rxmsg;
	txmsg;
	*/
//	if (nameOfTheHost)   delete (nameOfTheHost);
//	if (ipOfTheHost)     delete (ipOfTheHost);
//	if (rxmsg)                 delete (rxmsg);
//	if (txmsg)                 delete (txmsg);
}

void tcpIp::setNameOfTheHost(struct hostent *h)
{
	// initialize the nameOfTheHost atribute
//	nameOfTheHost = new char[strlen(h->h_name) + 1];
	strcpy(nameOfTheHost, h->h_name);
}

char *tcpIp::getNameOfTheHost(int print = 0)
{
	// return the nameOfTheHost atribute
	if (print)
	{
		cout << nameOfTheHost << endl;
	}
	return(nameOfTheHost);
```

```
}

void tcpIp::setIpOfTheHost(struct hostent *h)
{
	// initialize the ipOfTheHost atribute
//	ipOfTheHost = new char[strlen( inet_ntoa(*((struct in_addr *)h->h_addr)) ) + 1];
	strcpy(ipOfTheHost, inet_ntoa(*((struct in_addr *)h->h_addr)) );
}

char *tcpIp::getIpOfTheHost(int print = 0)
{
	// return the ipOfTheHost atribute
	if (print)
	{
		cout << ipOfTheHost << endl;
	}
	return(ipOfTheHost);
}

ulong tcpIp::simpleSendData(char *buffer)
{
	ulong retcode = OK;
	retcode = simpleSendData((byte*)buffer);

	return(retcode);
}
ulong tcpIp::sendData(char *buffer , int buflen)
{

	//cout << "CCCCCCCCCCCCCCCCCCCCCCCCCCCC" << endl;
	ulong retcode = OK;
	retcode = sendData((byte*)buffer, buflen);

	return(retcode);
```

```
}

ulong tcpIp::simpleSendData(byte *buffer)
{
	int bytes_sent = 0;		// how many bytes we´ve sent at that moment
	ulong buffer_length = 0;

	buffer_length = strlen((char*)buffer)+1;

	#if DEBUG
		cout << "DEBUG: IN simpleSendData FUNCTION 0 \n" << endl;
	#endif

//	txmsg = 0;
	/*
	if(txmsg) //safe delete
	{
		delete(txmsg);
		txmsg = 0;
	}

	txmsg = new byte[buffer_length];
*/
	#if DEBUG
		cout << "DEBUG: IN simpleSendData FUNCTION 1 \n" << endl;
	#endif

	#ifdef WIN32
		/*ulong i = 0;
		while( i < strlen(buffer) )
		{
			txmsg[i] = '\0';
			i++;
		}*/
```

```
            memset(txmsg, '\0', DATA_SIZE);
        #else
            bzero(txmsg, DATA_SIZE);
        #endif

        //bcopy(txmsg, buffer, buffer_length);
        memcpy(txmsg, buffer, buffer_length);
        //memccpy(txmsg, buffer, '\0', buffer_length);
        //strcpy(txmsg, buffer);

        #if DEBUG
            cout << "DEBUG: txmsg = " << txmsg << endl;
        #endif

        #if DEBUG
            cout << "DEBUG: IN simpleSendData FUNCTION 2 \n" << endl;
            //cout << "DEBUG: total_length = "<< total_length << " \n" << endl;
        #endif

#ifdef WIN32
        bytes_sent = send(socketDescriptor, (char*)txmsg, strlen((char*)txmsg), 0);
            if (bytes_sent < 0) { return(ERROR_SEND); }
#else
        bytes_sent = send(socketDescriptor, txmsg, strlen((char*)txmsg), 0);
            if (bytes_sent < 0) { return(ERROR_SEND); }
#endif


        #if DEBUG
            cout << "txmsg ->" << txmsg << "<-" << endl;
        #endif

        return(OK);
}
```

```
ulong tcpIp::sendData(byte *buffer, int buflen)
{
	//	cout << "DEBUG: AAAAAAAAAAAAAAAAAAAAAAAAA \n" <<
endl;
	ulong total = 0;			// how many bytes we´ve sent
	ulong bytes_left = 0;	// how many we have left to send
	ulong total_length = 0;
	int bytes_sent = 0;			// how many bytes we´ve sent at that moment
	ulong buffer_length = 0;

	buffer_length = buflen;

	#if DEBUG
		cout << "DEBUG: IN sendData FUNCTION 0 \n" << endl;
	#endif

//	txmsg = 0;
	/*
	if(txmsg) //safe delete
	{
		delete (txmsg);
		txmsg = 0;
	}

	txmsg = new byte[buffer_length];
*/
	#if DEBUG
		cout << "DEBUG: IN sendData FUNCTION 1 \n" << endl;
	#endif

		//cout << "DEBUG1: txmsg = " << txmsg << endl;

	//	cout << "DEBUG: BUFFER = " << buffer << endl;
```

```
//        cout << "DEBUG: buffer LEN = " << strlen((char*)buffer) << endl;
#ifdef WIN32
        /*ulong i = 0;
        while( i < strlen(buffer) )
        {
                txmsg[i] = '\0';
                i++;
        }*/
        memset(txmsg, '\0', DATA_SIZE);
#else
        bzero(txmsg, DATA_SIZE);
#endif

//        cout << "DEBUG2: txmsg = " << txmsg << endl;
//bcopy(txmsg, buffer, buffer_length);
memcpy(txmsg, buffer, buffer_length);
//memccpy(txmsg, buffer, '\0', buffer_length);
//strcpy(txmsg, buffer);
//        cout << "DEBUG: txmsg = " << txmsg << endl;
//        cout << "DEBUG: txmsg LEN = " << strlen((char*)txmsg) << endl;

#ifdef WIN32
        // Put CR, LF or EOS at the end of the rxmsg to avoid
        // writing garbage at cout or printf.
        //strcat(txmsg, "\r");  //0x0d;  //CR \r
        //strcat(txmsg, "\n");  //0x0a;  //LF \n
        //strcat(txmsg, "\0");  //0x00;  //EOS
#endif

#if DEBUG
        cout << "DEBUG: txmsg = " << txmsg << endl;
#endif

bytes_left = buflen;
```

```
	total_length = buflen;//strlen((char*)txmsg);

	#if DEBUG
		cout << "DEBUG: IN sendData FUNCTION 2 \n" << endl;
		cout << "DEBUG: total_length = "<< total_length << " \n" << endl;
	#endif

	while(total < total_length)
	{
		bytes_sent = send(socketDescriptor, (char*)(txmsg+total), bytes_left, 0);
		if (bytes_sent < 0) { return(ERROR_SEND); }
		total += bytes_sent;
		bytes_left -= bytes_sent;
	}

	#if DEBUG
		cout << "txmsg ->" << txmsg << "<-" << endl;
	#endif

	return(OK);
}

ulong tcpIp::receiveData(ulong maxlength, int *bytes_recved)
{
	#if DEBUG
		cout << "DEBUG: IN reciveTcpData FUNCTION \n" << endl ;
	#endif

	bytes_recved = 0;

//	rxmsg = 0;
/*
	if(rxmsg) //safe delete
	{
```

```
            delete(rxmsg);
            rxmsg = 0;
        }

        rxmsg = new byte[maxlength];
*/
        #ifdef WIN32
            /*ulong i = 0;
            while(i < maxlength)
            {
                rxmsg[i] = '\0';
                i++;
            }*/
            memset(rxmsg, '\0', DATA_SIZE);
        #else
            bzero(rxmsg, DATA_SIZE);
        #endif

        #if DEBUG
            cout << "DEBUG: IN recv FUNCTION \n" << endl;
            cout << &rxmsg << endl;
        #endif

#ifdef WIN32
        if( ( *bytes_recved = recv(socketDescriptor, (char*)rxmsg, DATA_SIZE, 0) ) < 0 )
            return(ERROR_REVC);
#else
        if( ( *bytes_recved = recv(socketDescriptor, rxmsg, DATA_SIZE, 0) ) < 0 )
            return(ERROR_REVC);
#endif

        #if DEBUG
            cout << "DEBUG: IN recv FUNCTION 2 \n" << endl;
            cout << "DEBUG: bytes_recved = "<< bytes_recved << " \n" << endl;
```

```
        #endif

        #ifdef WIN32
                // Put CR, LF or EOS at the end of the rxmsg to avoid
                // writing garbage at cout or printf.
                //strcat(rxmsg, "\r");  //0x0d;  //CR \r
                //strcat(rxmsg, "\n");  //0x0a;  //LF \n
                //strcat(rxmsg, "\0");  //0x00;  //EOS
        #endif

        #if DEBUG
                cout << "rxmsg ->" << rxmsg << "<-" << endl;
        #endif

   return(OK);
}

byte *tcpIp::getRXmsg(int print = 0)
{
        // return the rxmsg atribute
        if (print)
        {
                cout << rxmsg << endl;
        }
        return(rxmsg);
}

byte *tcpIp::getTXmsg(int print = 0)
{
        // return the txmsg atribute
        if (print)
        {
                cout << txmsg << endl;
        }
```

```
        return(txmsg);
}

ulong tcpIp::getConstructorError()
{
        return(constructorError);
}

#ifdef WIN32
void tcpIp::callWSAStartup()
{
        // WSAStartup call -- Needed for windows sockets
        WORD wVersionRequested;
        WSADATA wsaData;
        int err;

        wVersionRequested = MAKEWORD( 2, 2 );

        err = WSAStartup( wVersionRequested, &wsaData );
        if ( err != 0 )
        {
                /* Tell the user that we could not find a usable */
                /* WinSock DLL.                                  */
                return;
        }

        /* Confirm that the WinSock DLL supports 2.2.            */
        /* Note that if the DLL supports versions greater    */
        /* than 2.2 in addition to 2.2, it will still return */
        /* 2.2 in wVersion since that is the version we      */
        /* requested.                                        */

        if ( LOBYTE( wsaData.wVersion ) != 2 ||
                        HIBYTE( wsaData.wVersion ) != 2 )
```

```
        {
                /* Tell the user that we could not find a usable */
                /* WinSock DLL.                                  */
                WSACleanup( );
                return;
        }
        /* The WinSock DLL is acceptable. Proceed. */
}
#endif

// END OF tcp_client class functions ================================

/*********************************************************************/
/*                                    tcpClient class functions
                  */
/*********************************************************************/

// constructor
tcpClient::tcpClient(char *serverAddr, char *p, int v = 0)
{
        int myport = atoi(p);
        int retcode = OK;

        if( (retcode = setTcpClient(serverAddr, myport, v)) )
        {
                constructorError = retcode;
        }
}

// constructor
tcpClient::tcpClient(char *serverAddr, ushort p, int v = 0)
{
        int retcode = OK;
```

```
        if( (retcode = setTcpClient(serverAddr, p, v)) )
        {
                constructorError = retcode;
        }
}

// Called by the constructor to initialize parameters for the client.
ulong tcpClient::setTcpClient(char *serverAddr, ushort p, int v)
{

        #ifdef WIN32
                callWSAStartup();
        #endif

        #if DEBUG
                cout << "DEBUG: IN tcpClient CONSTRUCTOR \n" << endl;
        #endif

        struct hostent *h; //Hold the server information.
        constructorError = OK;
//      serverAddress = 0;

        // initialize the serverAddress atribute
//      serverAddress = new char[strlen(serverAddr) + 1];
        strcpy(serverAddress, serverAddr);
        // initialize the port atribute
        port = p;
        // initialize the verbose atribute
        verbose = v;

        #if DEBUG
                cout << "DEBUG: serverAddress = "<< serverAddress << " \n" << endl;
        #endif
```

```
        if( ( h = gethostbyname(serverAddress) ) == NULL ) // get the host info
        {
                if (verbose)
                {
                        cout << "DEBUG: constructorError = " << constructorError << endl;
                }
                return(ERROR_GETHOSTBYNAME);
        }
        else
        {
                // initialize the nameOfTheHost atribute
                setNameOfTheHost(h);
                // initialize the ipOfTheHost atribute
                setIpOfTheHost(h);

                // Setting structure parameters ************************
                dest_addr.sin_family = SOCK_FAMILY;
                dest_addr.sin_port   = htons(port);
                dest_addr.sin_addr   = *((struct in_addr *)h->h_addr);
                memset(&(dest_addr.sin_zero), '\0', 8); // zero to the rest of the structure
                // Setting structure parameters *******END***************
        }

        #if DEBUG
                cout << "DEBUG: LEAVING tcpClient CONSTRUCTOR! \n" << endl;
        #endif

        return(OK);
}

ulong tcpClient::initializeClientConnection()
{
        #if DEBUG
                cout << "DEBUG: IN initializeClientConnection FUNCTION \n" << endl;
```

```
        #endif

   // Opening socket ************************
        if ( ( socketDescriptor = socket(SOCK_FAMILY, SOCK_TYPE,
SOCK_PROTOCOL) ) < 0 )
                return(ERROR_OPEN_SOCKET);

        if (verbose)
        {
                cout << "TCP/IP client    : Socket Opened ... " << endl;
        }

   // Connecting to server *********************
   if ( connect(socketDescriptor, (struct sockaddr *) &dest_addr, sizeof(dest_addr)) < 0 )
                return(ERROR_CONNECT);

        if (verbose)
        {
                cout << "TCP/IP client    : Connected ...\n";
                cout << "Host Name (IP)   : " << nameOfTheHost << " (" << ipOfTheHost <<
")" << endl;
        }

        return(OK);
}

ulong tcpClient::endClient()
{
        int i = 0;

        #if DEBUG
                cout << "DEBUG: IN endClient FUNCTION \n" << endl;
        #endif
```

```
	#ifdef WIN32
		// close socket for Win32
		WSACleanup();
		return(OK);
	#else
		// close socket for unix
		if( (i = close(socketDescriptor)) )
		{
			return(ERROR_CLOSE);
		}
	#endif

	return(OK);
}

// destructor
tcpClient::~tcpClient()
{
	#if DEBUG
		cout << "DEBUG: IN DESTRUCTOR ~tcpClient() \n" << endl;
	#endif

//	if(serverAddress)	delete(serverAddress);

	// END tcp client
	if ( (retCode = endClient()) )
	{
		cout << tcpipErrorMsg [retCode] << endl;
	}
}
// END OF tcp_client class functions ================================


/*****************************************************/
```

```
/*                          tcpServer class functions                          */
/*************************************************************/

// constructor
tcpServer::tcpServer(char *p, int v = 0)
{
        int myport = atoi(p);
        int retcode = OK;

        if( (retcode = setTcpServer(myport, v)) )
        {
                constructorError = retcode;
        }
}

// constructor
tcpServer::tcpServer(ushort p, int v)
{
        int retcode = OK;

        if( (retcode = setTcpServer(p, v)) )
        {
                constructorError = retcode;
        }
}

ulong tcpServer::setTcpServer(ushort p, int v)
{
        #ifdef WIN32
                callWSAStartup();
        #endif

        #if DEBUG
                cout << "DEBUG: IN tcpServer CONSTRUCTOR \n" << endl;
```

```
        #endif

        struct hostent *h;        //Hold the server information.
        constructorError = OK;
//      ipOfThePeer = 0; //Hold connector's IP address
//      nameOfThePeer = 0; //Hold connector's name
        portOfThePeer = 0; //Hold connector's connection port

        // initialize the serverAddress atribute
        char serverAddress[DATA_SIZE];
        if( ( gethostname(serverAddress, sizeof(serverAddress)) ) < 0 ) // get the host name
        {
                if (verbose)
                {
                        constructorError = ERROR_GETHOSTNAME;
                        cout << "1: tcpServer Constructor Error = " << constructorError <<
endl;
                }
                return(ERROR_GETHOSTNAME);
        }

        // initialize the port atribute
        port = p;

        // initialize the verbose atribute
        verbose = v;

        if( ( h = gethostbyname(serverAddress) ) == NULL ) // get the host info
        {
                if (verbose)
                {
                        constructorError = ERROR_GETHOSTBYNAME;
                        cout << "2: tcpServer Constructor Error = " << constructorError <<
endl;
```

```
            }
            return(ERROR_GETHOSTBYNAME);
        }
        else
        {
            // initialize the nameOfTheHost atribute
            setNameOfTheHost(h);
            // initialize the ipOfTheHost atribute
            setIpOfTheHost(h);

            // Setting structure parameters ************************
            server_addr.sin_family = SOCK_FAMILY;
            server_addr.sin_port   = htons(port);
            server_addr.sin_addr   = *((struct in_addr *)h->h_addr);
            memset(&(server_addr.sin_zero), '\0', 8); // zero to the rest of the structure
            // Setting structure parameters *******END**************
        }

        #if DEBUG
            cout << "DEBUG: LEAVING tcpServer CONSTRUCTOR! \n" << endl;
        #endif

        return(OK);
}

ulong tcpServer::initializeSingleServer()
{
        #ifdef WIN32
            int sin_size;
        #else
            socklen_t sin_size;
        #endif

        retCode = initializeServer();
```

```
        sin_size = sizeof(struct sockaddr_in);
        if ( ( socketDescriptor = accept(sock_fd, (struct sockaddr *) &client_addr, &sin_size)
) < 0 )
        {
                #ifdef WIN32
                        // close socket for Win32
                        WSACleanup();
                        return(ERROR_ACCEPT);
                #else
                        // close socket for unix
                        close(sock_fd);
                        return(ERROR_ACCEPT);
                #endif
        }

        if (verbose)
        {
                cout << "TCP/IP server      : Connection accepted from " <<
inet_ntoa(client_addr.sin_addr) << endl;
        }

#ifdef WIN32
#else
        if ( close(sock_fd) )
        {
                return(ERROR_CLOSE);
        }
#endif

        return(retCode);
}

ulong tcpServer::initializeServer()
```

```
{
        #ifdef WIN32
                char yes = 1;
        #else
                int  yes = 1;
        #endif

        // Opening socket ************************
        if ((sock_fd = socket(SOCK_FAMILY, SOCK_TYPE, SOCK_PROTOCOL)) < 0)
                        return(ERROR_OPEN_SOCKET);

        if (verbose)
        {
                cout << "TCP/IP server      : Socket Opened ..." << endl;
        }

        // Forcing the port to be released ********
        // Lose the pesky "Address alredy in use" error message *********
        setsockopt(sock_fd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int));

        // Binding to port *********************
        if ( ( bind( sock_fd, (struct sockaddr *) &server_addr, sizeof(server_addr) ) ) < 0 )
        {
                #ifdef WIN32
                        // close socket for Win32
                        WSACleanup();
                        return(ERROR_BIND);
                #else
                        // close socket for unix
                        close(sock_fd);
                        return(ERROR_BIND);
                #endif
        }
```

```
	if (verbose)
	{
		cout << "TCP/IP server      : Bind to port " << port << " ..." << endl;
	}

	// Listenig to port ****************************************************
	// this server accept one single connection. (BACKLOGSINGLE = 1)
	if ((listen(sock_fd, BACKLOGSINGLE)) < 0)
	{
		#ifdef WIN32
			// close socket for Win32
			WSACleanup();
			return(ERROR_LISTEN);
		#else
			// close socket for unix
			close(sock_fd);
			return(ERROR_LISTEN);
		#endif
	}

	if (verbose)
	{
		cout << "TCP/IP server      : Listening port " << port << " ..." << endl;
		cout << "Server Name (IP)   : " << nameOfTheHost << " (" << ipOfTheHost
<< ")" << endl;
	}

	return(OK);
}

ulong tcpServer::setPeerInfo()
{
	#if DEBUG
		cout << "DEBUG: IN setPeerInfo() METHOD!!!" << endl;
```

```
	#endif

	int retcode = OK;
	struct sockaddr_in peer_addr; //Hold connector's addr

	#ifdef WIN32
		int peer_addr_len;
	#else
		socklen_t peer_addr_len;
	#endif

	if( ( getpeername(socketDescriptor, (struct sockaddr *) &peer_addr, &peer_addr_len)
) < 0 )
	{
		return(ERROR_GETPEERNAME);
	}

	if ( (retcode = setPeerInfo(peer_addr)) )
	{
		return(retcode);
	}

	return(OK);
}

ulong tcpServer::setPeerInfo(struct sockaddr_in peer_addr)
{
	struct hostent *h;

	if( ( h = gethostbyname( inet_ntoa(peer_addr.sin_addr) ) ) == NULL ) // get the
conectors info
	{
		return(ERROR_GETHOSTBYNAME);
	}
```

```
        #if DEBUG
                cout << "ipOfThePeer::::::" << endl;
        #endif
        // initialize the ipOfThePeer atribute
//      ipOfThePeer = new char[strlen( inet_ntoa(*((struct in_addr *)h->h_addr)) ) + 1];
        strcpy(ipOfThePeer, inet_ntoa(*((struct in_addr *)h->h_addr)) );

        #if DEBUG
                cout << "nameOfThePeer::::::" << endl;
        #endif
        // initialize the nameOfThePeer atribute
//      nameOfThePeer = new char[strlen(h->h_name) + 1];
        strcpy(nameOfThePeer, h->h_name);

        #if DEBUG
                cout << "portOfThePeer::::::" << endl;
        #endif
        // initialize the portOfThePeer atribute
        portOfThePeer = ntohs(peer_addr.sin_port);

        return(OK);
}

char *tcpServer::getPeerAddr(int print = 0)
{
        // return the ipOfThePeer atribute
        if (print)
        {
                cout << ipOfThePeer << endl;
        }
        return(ipOfThePeer);
}
```

```
char *tcpServer::getPeerName(int print = 0)
{
	// return the nameOfThePeer atribute
	if (print)
	{
		cout << nameOfThePeer << endl;
	}
	return(nameOfThePeer);
}

int tcpServer::getPeerPort(int print = 0)
{
	// return the portOfThePeer atribute
	if (print)
	{
		cout << portOfThePeer << endl;
	}
	return(portOfThePeer);
}

ulong tcpServer::disconectClient()
{
	int i = 0;

	#if DEBUG
		cout << "DEBUG: IN endServer FUNCTION \n" << endl;
	#endif

	#ifdef WIN32
		// close socket for Win32
		WSACleanup();
		return(OK);
	#else
		// close socket for unix
```

```
                if( (i = close(socketDescriptor)) )
                {
                        return(ERROR_CLOSE);
                }
        #endif

        return(OK);
}

// destructor
tcpServer::~tcpServer()
{
        #if DEBUG
                cout << "DEBUG: IN DESTRUCTOR ~tcpServer() \n" << endl;
        #endif

//      if (nameOfThePeer)    delete(nameOfThePeer);
//      if (ipOfThePeer)      delete(ipOfThePeer);

        #ifdef WIN32
                // close socket for Win32
                WSACleanup();
        #endif
}
// END OF tcp_server class functions ================================
```

---------- Fim do arquivo tcp_lib.h.cpp ----------

e) Shell para compilar o isAliveDaemon:

```
echo "Compilando..."
g++ -I/usr/local/include/ -I/usr/local/include/mysql -O2 -c isAliveDaemon.cpp tcp_lib.cpp
```

```
echo "Linkando..."
g++ -g -O2 -L/usr/local/lib/mysql -o isAliveDaemon isAliveDaemon.o -L/usr/local/lib/ -
lsqlplus -lz -lmysqlclient -lz -lmysqlclient tcp_lib.cpp -lmcrypt -lltdl libcrypto.a

echo "Terminando..."
rm *.o
```

Apêndice H - Scripts PHP

a) index.php:

```
<html>

<head>
  <title>Bemvindo ao ponto de acesso WStrike - RAVEL</title>
</head>



<body bgcolor="#FFFFFF" text="#000000">



 <center><h2> Autenticação StrikeIN </h2> </center>

 <form name="login_frm" method="post" action="efetuaLogin.php">

  <table align=center border=0 cellspacing=5 cellpadding=5>
   <tr>
    <td bgcolor="#666666" width="141" height="113" align="left" >
     <p align="right"><font color="#FFFFFF"> Login </font></p>
     <p align="right"><font color="#FFFFFF">Senha </font></p>
    </td>

      <td bgcolor="#CCCCCC" width="296" height="113" align="left" >
     <p class=t3 align='center'> <a href='insere_respon_form.php'><span class=t3>
      </span></a>
      <input type="text"  tabindex="1" name="usr" size="30" maxlength="30">
     <p class=t3 align='center'>
      <input type="password"  tabindex="2" name="pass_usr" size="30" maxlength="16">
    </td>
```

```
    </tr>
   </table>


   <center><input type="submit" name="OK" value="  OK  "> </center>

  </form>

<center>
   <p><a href="isAliveStation/isAlive.conf">isAlive.conf</a>

   <p><a href="isAliveStation/isAliveStation.exe">isAliveStation.exe</a>
</center>


</body>

</html>
```

b) efetuaLogin.php

```
<?php

include "conectaInc.php";

$table_name = "passwd";

//$Pass_usr=crypt($pass_usr,"er");
//$pass_usr=substr($pass_usr,2);

$sql="SELECT senha from $table_name WHERE login='$usr' AND senha='$pass_usr'";

$resultado=mysql_query($sql,$con);
```

```
if(mysql_num_rows($resultado)>0){

  echo "Login OK\n";

  $ip = getenv( "REMOTE_ADDR" );

  $sqlInsere = "INSERT INTO ipsValidos values('$pass_usr', '$usr', '$ip', '$pass_usr','0')";

  $statusInsere = mysql_query($sqlInsere, $con) or exit("Nao inseriu o IP\n");


  // Pega todos os ips validos do bd
  $sqlIP = "SELECT enderecoIP from ipsValidos";

  $resultado = mysql_query($sqlIP, $con) or exit("Nao foi possivel consultar tabelas de ips");

  if ($row = mysql_fetch_array($resultado) )
    $string = "$row[enderecoIP]/32" or exit("Nenhum IP encontrado");

  while($row = mysql_fetch_array($resultado) ) {
    $string =  "$string, $row[enderecoIP]/32";
  }

  echo $string;

  $ipsValidos = "IPS = '\{ $string }'\n";
  // Functions
  function file_get_contents($filename, $use_include_path = 0) {
    $fd = fopen ($filename, "rb", $use_include_path);
    $contents = fread($fd, filesize($filename));
    fclose($fd);
    return $contents;
  }
```

```
$filename = "pf.conf";

// Delete the file with the old IPs
if (file_exists($filename)){
   unlink($filename);
}

// Create an empty file which will contain de new valid IPs
touch ($filename);

// Merge the new IPs file and the rules file into pf.conf

if (!$fp = fopen($filename, 'a')) {
   echo "O arquivo não pode ser aberto\n";
   exit;
}

// Write $somecontent to our opened file.
if (!fwrite($fp, $ipsValidos) ) {
   echo "Não foi possível escrever no arquivo";
   exit;
}

if (!fwrite($fp, file_get_contents("regras.conf", 0))) {
    echo " Não foi possível escrever no arquivo";
    exit;
}

print "Success, wrote ($ipsValidos) to file ($filename)";


fclose($fp);

// Reload pf rules
```

```
  exec('sudo pfctl -f /var/www/htdocs/pf.conf');

} else {

  echo "Login incorreto";

}
?>
```

c) conectaInc.php

```
<?php

      //realiza a conexão no Banco MySql
      $con=mysql_connect("strikein","wstrike","wstrike");
      if (!$con)
      {
      echo("<body bgcolor='#FFFFFF' text='#000000'>
            <p align='center'> </p>
            <table width='101%' border='0' height='60'>
              <tr>
                <td bgcolor='#FFFFFF'>
                  <p align='center'><img src='cabecalho_ravel.gif' width='768'
height='77'></p>
                  <p class=bodymaior2_titulo align='center'><font color='#000000'>  Cadastro
                    de Relat&oacute;rios</font></p>
                </td>
              </tr>
            </table>
            <p> </p>");
      echo("<p class=t3> Erro no Banco de Dados </p>");
      exit;
      }
      mysql_select_db("strikein",$con);
```

```
    $endl="<br>";

?>
```

Apêndice I – Regras do Firewall e NAT

IPS = ips inseridos pelo script PHP

```
# Regras de NAT
nat on rl0 from 192.168.0.0/24 to any -> 10.10.0.16

# Regras do firewall

# Definicoes
semfio     = "192.168.0.0/24"
ap         = "192.168.0.1/32"
ap_externa = "10.10.0.16/32"

# Nega tudo por padrao
block in log on {enc0, wi0} all
block out log on {enc0, wi0} all

# Deixa passar o que for encriptado
pass in proto esp from $semfio to $ap
pass out proto esp from $ap to $semfio

# Deixa passar informacoes de controle UDP do isakmpd
pass in  on wi0 proto udp from $semfio port = 500 to $ap port = 500
pass out on wi0 proto udp from $ap port = 500 to $semfio port = 500

# Deixa passar DHCP
pass in on wi0 proto udp from $semfio to $ap port = 67
pass in on wi0 proto udp from $ap port = 67 to $semfio

# Tirar header devido ao encapsulamento
pass in on enc0 proto ipencap all

# Deixa acessar a pagina de autenticacao
```

```
pass in on enc0 proto tcp from $semfio to $ap port {22, 443}
pass out on enc0 proto tcp from $ap port {22, 443} to $semfio

# Deixa passar isAlive
pass in on enc0 proto tcp from $IPS to $ap flags S/SA keep state
pass out on enc0 proto tcp from $ap to $IPS port 56789 flags S/SA keep state

# Deixa passar SAMBA
pass in  on enc0 proto {udp, tcp} from $semfio to 10.10.0.76 port {137,138,139}
pass out on enc0 proto {udp, tcp} from 10.10.0.76 port {137,138,139} to $semfio

# Libera acesso para IPS autenticados
pass in  on enc0 proto tcp from $IPS to any port {ssh, http, https}
pass out on enc0 proto tcp from any port {ssh, http, https} to $IPS

# Libera consulta DNS para IPS autenticados
pass in  on enc0 proto { tcp, udp } from $IPS to 10.10.0.76 port = 53
pass out on enc0 proto { tcp, udp } from 10.10.0.76 port 53 to $IPS
```

Apêndice J – Arquivos de configuração da VPN no AP

a) isakmpd.conf:

---------- Início do arquivo isakmpd.conf ----------

```
[Phase 1]
Default=ISAKMP-clients

[Phase 2]
passive-Connections=IPsec-clients

[ISAKMP-clients]
Phase=1
Configuration=Sentinel-main-mode
Authentication= mekmitasdigoat

[IPsec-clients]
Phase=2
Configuration=Sentinel-quick-mode
Local-ID=Local-Net
Remote-ID=Remote-host

[Local-Net]
ID-type=IPV4_ADDR_SUBNET
Network=0.0.0.0
Netmask=0.0.0.0

[Remote-host]
ID-type=IPV4_ADDR
Address=0.0.0.0

[Sentinel-main-mode]
EXCHANGE_TYPE=ID_PROT
Transforms=ISES_MAIN
```

```
[Sentinel-quick-mode]
EXCHANGE_TYPE=QUICK_MODE
Suites=ISES_QUICK

[ISES_MAIN]
ENCRYPTION_ALGORITHM=   3DES_CBC
HASH_ALGORITHM=         MD5
AUTHENTICATION_METHOD=  PRE_SHARED
GROUP_DESCRIPTION=      MODP_1024
LIFE=                   ANY

[ISES_QUICK]
ENCRYPTION_ALGORITHM=   3DES_CBC
HASH_ALGORITHM=         HMAC_MD5
AUTHENTICATION_METHOD=  PRE_SHARED
GROUP_DESCRIPTION=      MODP_1024
LIFE=                   ANY
wstrike#
```

---------- Fim do arquivo isakmpd.conf ----------

b) isakmpd.policy

---------- Início do arquivo isakmpd.policy ----------

```
Comment: This policy accepts ESP SAs from a remote that uses the rigth password.
Authorizer: "POLICY"
Conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" &&
            esp_enc_alg != "null" -> "true";
```

---------- Fim do arquivo isakmpd.policy ----------

Apêndice K – Configuração do SSH Sentinel

- Deve-se adicionar uma VPN clicando-se no botão *Add no Folder* VPN *Connections*

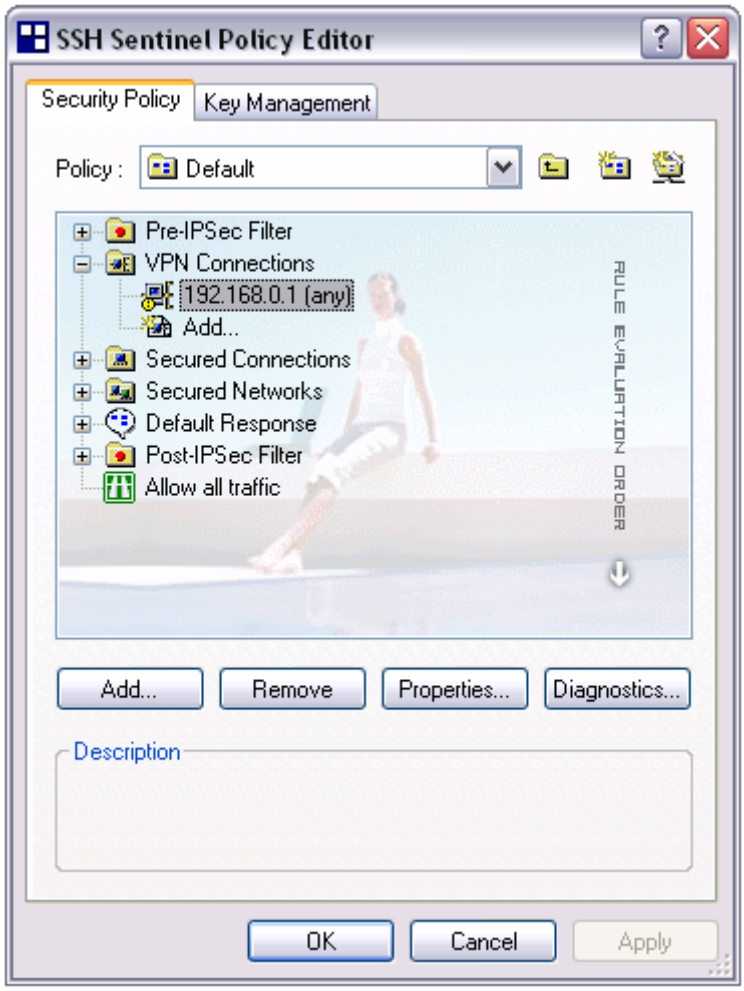

Figura 8 - Adicionando uma VPN no SSH Sentinel

- Na próxima tela devem ser incluídas as informações sobre as regrais gerais da VPN

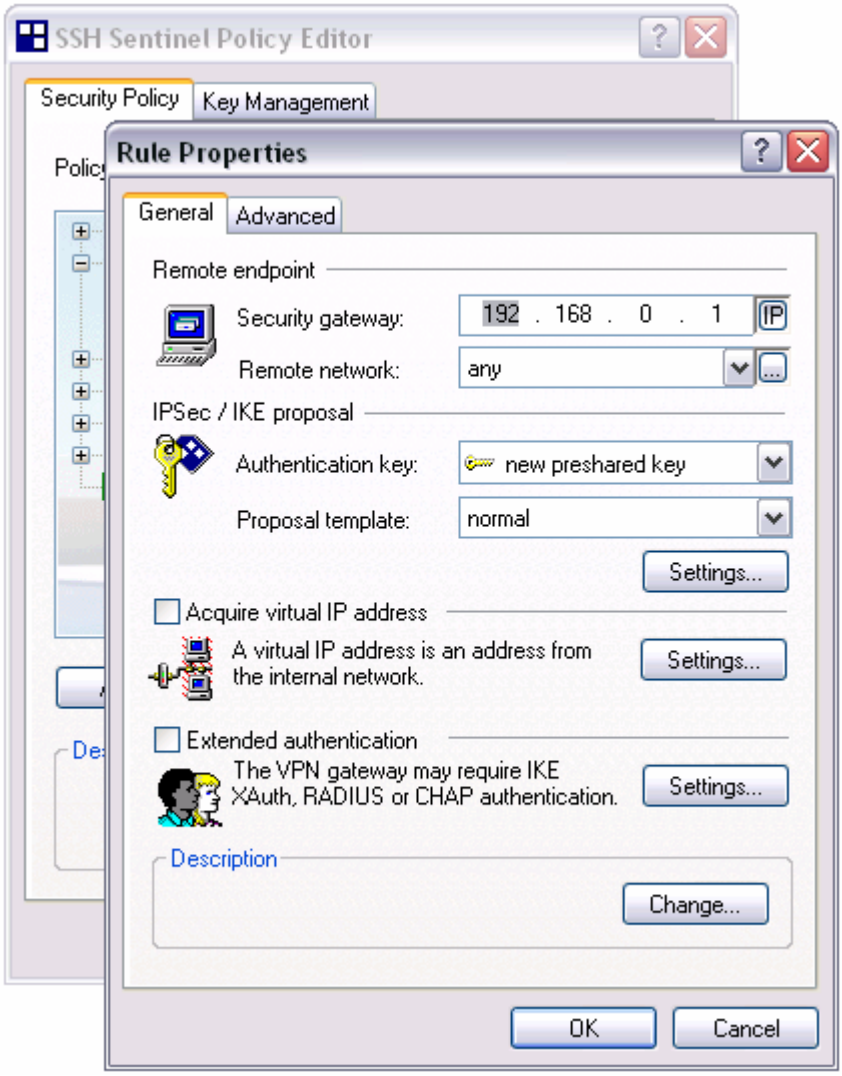

Figura 9 - Configurações gerais da VPN no SSH Sentinel

- Acessa-se a próxima tela clicando-se no botão *Settings*, logo abaixo de *IPSec / IKE proposal*

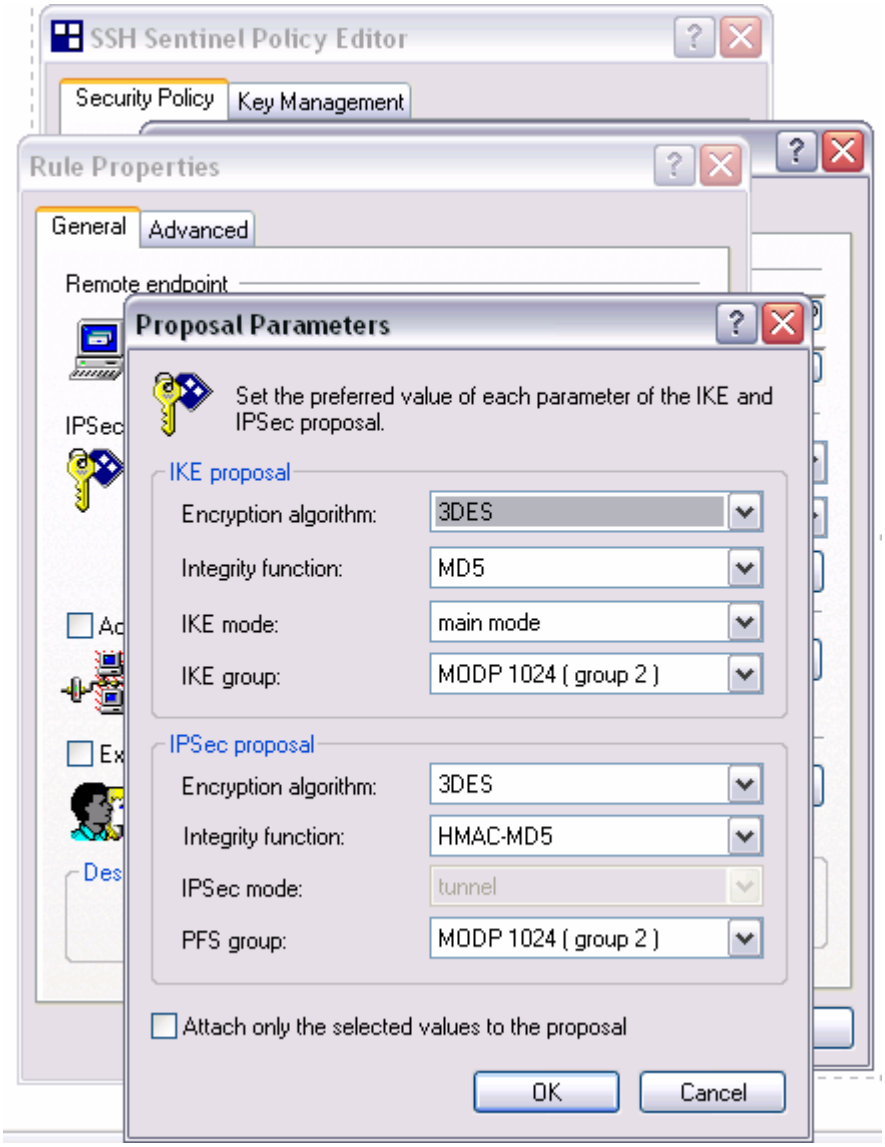

Figura 10 - Configurações de algoritmos de encriptação e hash para o SSH Sentinel

- Retornando a tela anterior, configura-se os parâmetros presentes na guia *Advanced*.

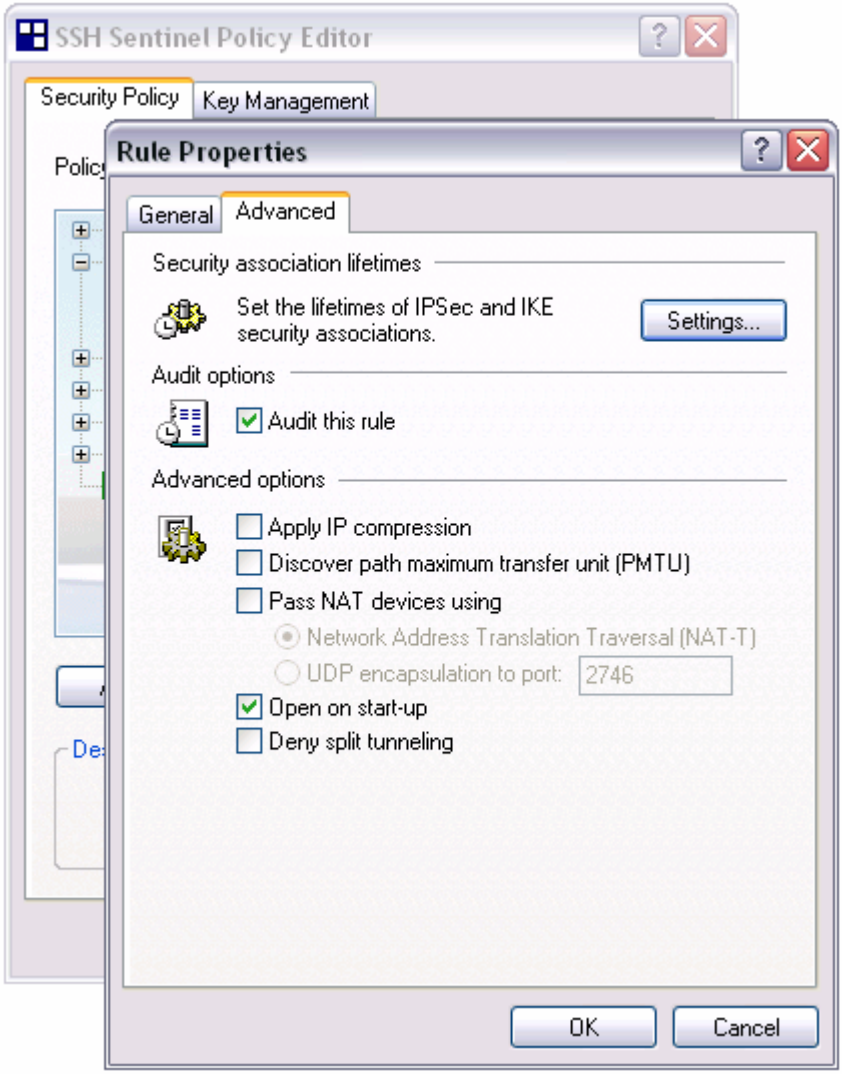

Figura 11 - Configurações avançadas da VPN no SSH Sentinel

Ao reiniciar o computador o SSH Sentinel estará ativo e pronto para formar uma VPN.