ANDERSON DE AGUIAR KOPKE

Avisa - FERRAMENTA PARA EMISSÃO DE RELATÓRIO DE SERVIDORES DE COMPARTILHAMENTO DE ACESSO A INTERNET

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras, como parte das exigências do curso de Pós-Graduação *Lato Sensu* Administração em Redes Linux para obtenção do título de Especialista em Administração em Redes Linux.

Orientador

Prof. Herlon Ayres Camargo

LAVRAS

MINAS GERIAS – BRASIL

2005

ANDERSON DE AGUIAR KOPKE

Avisa - FERRAMENTA PARA EMISSÃO DE RELATÓRIO DE SERVIDORES DE COMPARTILHAMENTO DE ACESSO A INTERNET.

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras, como parte das exigências do curso de Pós-Graduação *Lato Sensu* Administração em Redes Linux para obtenção do título de Especialista em Administração em Redes Linux.

Aprovada em _____.

Prof. _____.

Prof. _____.

_____

Prof. Herlon Ayres Camargo (Orientador)

LAVRAS

MINAS GERIAS – BRASIL

Dedicatória

Dedico este trabalho à minha mãe Lídia, minha futura esposa Evelyn e meus pais por consideração José Luiz e Neíla. Pessoas que jamais serão esquecidas.

Agradecimentos

A Deus, por sua infinita misericórdia que alcança a todos nós, por sua ajuda, zelo e atenção constantes, mesmo quando negligenciamos sua presença para cuidar de nossos interesses.

Sumário

Lista de Figuras

Resumo

Este trabalho apresenta a elaboração de uma solução capaz de auxiliar no processo de acompanhamento de servidores de compartilhamento de acesso à internet construídos sobre *hardware* de baixo custo, instalados por uma empresa prestadora de serviços de informática em seus clientes. A proposta é a construção de um ferramenta desenvolvida em *Shell-Script* capaz de emitir relatórios diários com informações sobre o *hardware*, sistema, rede e *proxy*, mantendo a empresa informada sobre acontecimentos nestes servidores.

Capítulo 1 - Introdução

O serviço de acesso a internet conhecido como banda larga faz parte do cotidiano de um grande número de cidades do país. Uma das características deste tipo de conexão é seu fornecimento de forma contínua, podendo ficar conectado 24 horas por dia. A possibilidade de compartilhar o acesso à internet, por exemplo, para uma pequena rede através deste tipo de conexão parece sempre muito benéfica.

A facilidade de fazer um compartilhamento de conexão banda larga em um sistema operacional como o Windows XP, pode acabar trazendo sérios problemas para dentro do ambiente de rede local. Associado a um sistema operacional despreparado e muito visado por desenvolvedores de pragas cibernéticas como vírus, cavalo de tróia, *worms* e *spywares*, este serviço se transforma rapidamente de benefício em problema.

Este tipo de problema atinge principalmente lojas, escritórios e consultórios que não possuem funcionários preparados para trabalhar nesta área e acabam por prejudicar o funcionamento do ambiente computacional da empresa.

Com a finalidade de garantir o compartilhamento do acesso à internet de seus clientes de forma eficiente, uma empresa prestadora de serviços passou a oferecer a instalação de servidores Linux preparados para cumprir esta tarefa. Aproveitando a capacidade do sistema Linux de trabalhar com micro computadores antigos, o serviço era oferecido para ser instalado em um micro do próprio cliente que estivesse desativado ou disponível.

Esta solução traz benefícios para o cliente, como o compartilhamento de sua conexão de forma segura e controle do acesso à internet de seus funcionários. Como o *hardware* necessário existia dentro da própria empresa do cliente, o custo deste serviço diminuia, se tornando ainda mais atraente. Os servidores instalados eram em sua maioria equipados com 16 ou 32 *megabytes* de memória *RAM*, disco rígido de poucos *gigabytes* e com processador Intel Pentium ou AMD K6.

Os servidores eram instalados com os serviços essenciais para o

compartilhamento, incluindo apenas a ativação do *firewall* Netfilter[1], do servidor *proxy* Squid[2] e do serviço de acesso remoto OpenSSH[3].

Após a instalação na empresa do cliente, estes servidores eram testados, porém nenhum acompanhamento de seu funcionamento era realizado periodicamente. Este problema se agravava quando eram clientes que contratavam exclusivamente o serviço, não mantendo nenhum outro vínculo com a empresa contratada para realizar o serviço.

Assim, com esta total falta de informação, não era sabido se o cliente estava utilizando o servidor, se o servidor estava atualizado ou mesmo se outra empresa havia sido chamada para resolver algum problema de conexão. Além disso, se o servidor apresentasse alguma anormalidade que pudesse indicar um futuro problema, esta informação era de conhecimento apenas do próprio sistema.

Após a instalação, a senha do superusuário era divulgada aos responsáveis de cada empresa, caso acontecesse algum tipo de acesso e o sistema fosse danificado, provar o acontecido se tornava uma tarefa muito complicada.

Este trabalho apresenta uma proposta de solução para resgatar o acompanhamento destes servidores junto às empresas as quais se prestou o serviço. Apesar do cliente ganhar um acompanhamento (mesmo que não esteja ciente disso), o maior beneficiado será a empresa que prestou o serviço, já que evitará transtornos e diminuirá a possibilidade de ocorrer problemas que possam desgastar o relacionamento com seus clientes.

O desafio é encontrar uma forma de obter informações sobre estes servidores, onde dados como especificações do *hardware*, sistema, rede e serviços instalados possam ser computados e analisados. O serviço prestado não deve ser interrompido para estas alterações, visto que ele foi implementado e concluído. Para isso deve-se contornar as limitações do *hardware* envolvido através de uma solução eficiente.

1 Netfilter – www.netfilter.org
2 Squid – www.squid-cache.org
3 OpenSSH – www.openssh.com

Capítulo 2 - Justificativa

2.1 - Considerações iniciais

Este capítulo apresenta uma justificativa para a escolha dos recursos utilizados na implementação da solução proposta neste trabalho. As possibilidades avaliadas são discutidas de forma objetiva, mas sem deixar de mostrar que outras soluções podem ser criadas.

2.2 - Motivação

Segundo (RESENDE, 2002), antes de definir objetivos, precisa-se conhecer bem o problema em questão. Por isto, esta seção apresenta as várias etapas da análise e definição do problema que motivou a elaboração deste trabalho. Por se tratar de um problema vivido no cotidiano do autor, as informações serão detalhadas da melhor forma possível.

De acordo com as informações fornecidas no Capítulo 1, onde é tratada a contextualização do problema, o desafio é restabelecer o contato entre empresa e servidor instalado no cliente, fazendo com que inúmeros problemas sejam evitados neste relacionamento.

Sem informações sobre o servidor instalado, não é possível saber se ele está sendo utilizado. Nenhum tipo de prevenção ou atualização pode ser oferecida ao cliente. O abandono do servidor trás problemas para ambos os lados envolvidos, por um lado o prestador não tem garantia de ter realizado um bom serviço e, conseqüentemente, pode estar colocando sua imagem em risco. Do outro lado, pode haver um cliente insatisfeito por não ter o produto funcionando de acordo com o idealizado. Este tipo de impasse pode afastar o cliente da empresa e gerar mal testemunho para possíveis futuros clientes.

Esta falta de informação por parte da empresa prestadora de serviços impossibilita o acompanhamento do serviço realizado e coloca em risco a relação entre empresa e cliente. Uma solução precisa ser instaurada, fornecendo informações relevantes à manutenção do serviço prestado, lembrando-se que a limitação do *hardware* deve ser observada como fator decisivo para a escolha dos procedimentos a serem tomados. O serviço deve ser prestado de forma transparente ao cliente, sem onerá-lo ou interromper a atividade de seu servidor.

De acordo com as definições anteriores é possível definir a motivação em sua forma mais concreta:

"Restabelecer o contato entre prestadora de serviço e o servidor instalado no cliente, (de forma que esse acompanhamento gere benefícios para a relação empresa/cliente) minimizando o impacto da instalação de novos serviços, respeitando as limitações do *hardware* envolvido, o não interrompimento do serviço realizado, além de não onerar o cliente em qualquer instância."

2.3 - Objetivos

2.3.1 - Definição do propósito

Prevenir futuros problemas que possam atrapalhar o relacionamento entre a prestadora de serviço e o cliente e acompanhar os acontecimentos que interfiram direta ou indiretamente no bom funcionamento do servidor.

2.3.2 - Proposta para solução

Pode-se definir formalmente a proposta de solução como:

"Encontrar ou desenvolver uma ferramenta capaz de emitir relatórios sobre o servidor instalado no cliente. "

Com base no enunciado do problema, o primeiro passo é questionar quais informações serão relevantes para este relatório e relacionar os objetivos a serem alcançados.

"Restabelecer o contato entre prestadora de serviço e o servidor instalado no cliente,..."

- Os clientes utilizam conexões *adsl* velox, assim é necessário um acesso remoto garantido mesmo com a troca periódica do endereço IP, característica deste tipo de serviço. Para tal, o primeiro procedimento, antes mesmo de qualquer implementação, é criar um cadastro em um serviço de DNS dinâmico para cada servidor instalado. Esta tarefa poderá ser realizada na

fase de instalação da ferramenta que será criada ou escolhida para a emissão dos relatórios.

"...(de forma que esse acompanhamento gere benefícios para a relação empresa/cliente)..."

- Para acompanhar o serviço prestado e saber, por exemplo, se o cliente fez alguma alteração no *hardware* do servidor solicitando a intervenção de outra empresa, é interessante que esta ferramenta possa registrar informações sobre o *hardware* do servidor;

- Para manter o servidor com o *software* atualizado, informações sobre versões dos pacotes instalados também deverão ser capturados;

- A fim de evitar que o serviço pare por algum problema no servidor, detalhes sobre a utilização da memória, módulos e processos ativos ou espaço em disco também são relevantes para o relatório;

- Após a instalação do servidor, a senha do *root* é de responsabilidade do cliente, que cuida do desligamento do servidor, além do cadastro de *sites* liberados ou bloqueados. Assim, dados como acesso ao servidor, comandos executados, reinicializações e alterações no sistema de arquivo se tornam imprescindíveis.

"... minimizando o impacto da instalação de novos serviços, respeitando as limitações do hardware envolvido..."

- Devido ao *hardware* utilizado nestes servidores, a solução deverá ser implementada de forma que a instalação de novos serviços, principalmente na forma de *daemons*, seja cautelosa. Por isso, a utilização de recursos disponíveis para coletar as informações para o relatório é uma boa escolha.

"...o não interrompimento do serviço realizado, além de não onerar o cliente em qualquer instância."

- A melhor maneira de atender esta necessidade será resolver o problema através da configuração remota das novas atribuições do servidor.

2.4 – Metodologia

Após a definição das atribuições da solução a ser implantada, pesquisas na internet foram realizadas a fim de encontrar alguma ferramenta que pudesse atender às necessidades em questão. O resultado das pesquisas não apontou uma ferramenta específica, mas mostrou uma alternativa que seria a utilização de algumas ferramentas (que suprem parcialmente as necessidades) em conjunto com saídas de comandos presentes na própria distribuição. Para unir as saídas destas ferramentas e dos comandos em um relatório unificado capaz de fornecer todas as informações solicitadas, optou-se por usar a programação em *Shell script*.

Para facilitar o acesso aos relatórios emitidos por cada servidor, o ideal seria tê-los em um lugar centralizado, onde o administrador pudesse acessá-los sem a necessidade de consultar cada servidor para isso. Foi resolvido então que cada servidor enviaria os relatórios através de uma cópia remota utilizando-se das opções de cadastro de chaves públicas e privadas do serviço *OpenSSH*.

Para melhorar o acesso aos relatórios, estes seriam visualizados remotamente através de um *browser*, o que possibilitaria consultas por meio de qualquer micro ligado à internet. Como resultado destas tarefas, o administrador poderia então fazer o acompanhamento destes servidores, garantindo a qualidade do serviço prestado.

2.4.1 - Em busca de uma ferramenta

Na fase de definição do problema, ficou claro que todos aqueles detalhes particulares, tornariam a pesquisa por uma ferramenta capaz de resolver todas as exigências muito difícil. Após várias pesquisas no *Google*[4], *SourceForge*[5] e *Freshmeat*[6], foram separados apenas dois projetos que se identificavam com a questão, ambos desenvolvidos através de *Shell script*.

O primeiro (BARBOSA, 2005) apresentava poucos recursos. É um

4 Google - www.google.com
5 SourceForge- www.sourceforge.net
6 Freshmeat- www.freshmeat.net

script simples de estrutura linear utilizando uma seqüência de comandos que redireciona suas saídas para um arquivo de registros. Apesar da simplicidade, ele captura vários dados do servidor.

O segundo (KANDASAMY, 2005) que foi analisado demonstrou mais propriedade. Apesar de não atender diretamente como uma solução, mostrou alguns conceitos como a organização do *script* em funções e a utilização do diretório *proc* para retirar informações necessárias sobre *hardware* e rede.

Como observado em (BARBOSA, 2005) e (KANDASAMY, 2005), a linguagem *Shell* é uma boa opção por ser simples de programar e, ao mesmo tempo, oferecer recursos para criar estruturas complexas de programação. Além disso, por ser amplamente utilizada em tarefas cotidianas de automação em servidores, é uma linguagem bastante familiar aos administradores de redes Linux. Sendo assim, o *Shell* padrão do Linux, conhecido como *Bash,* passa a ser uma boa opção para servir de base para a construção da solução deste problema.

2.4.2 - A opção pelo desenvolvimento

Com uma pesquisa rápida em (NEVES, 2003), (CAMARGO, 2005) e (SICA, UCHÔA, 2004), pôde-se concluir que a utilização conjunta entre *Bash* e comandos básicos de administração de sistemas seriam suficientes para a elaboração de uma ferramenta capaz de atender os requisitos do problema. Seu relacionamento com expressões regulares através de comandos como `sed`, `grep` e sua interoperabilidade com a linguagem `awk` permite elaborar relatórios formatados sem dificuldade.

Para fazer a verificação da integridade dos arquivos do sistema, foi escolhida a ferramenta Tripwire (MELO; TRIGO, 2004) que havia sido utilizada pelo autor deste trabalho em outras ocasiões. Desta forma, será possível fazer o acompanhamento das alterações que porventura venham a ocorrer no sistema de arquivo.

Durante pesquisa, foi encontrada uma ferramenta capaz de realizar testes no disco rígido utilizando a tecnologia SMART inseridas nos discos. Apesar de não ser compatível com discos muito antigos, seria possível testá-la e usá-la nos discos compatíveis. Esta ferramenta se chama Smartmontools (ALLEN, 2004) e será muito útil neste trabalho.

2.5 - Considerações finais

A análise do problema mostrou que existem uma série de fatores que precisam ser levados em consideração, como recursos do *hardware*, custos e acesso remoto. Desta forma a solução precisa ser abrangente para atender todas as necessidades e minuciosa para trabalhar as particularidades do problema. As etapas que se seguiram à contextualização do problema relatado no Capítulo 1, ajudaram a dimensionar o tamanho exato do problema e idealizar uma solução que se encaixe perfeitamente nos moldes da questão.

Foi decidido por implementar uma ferramenta baseada na linguagem *Shell.* Ela utilizará o *Bash*, (*Shell* padrão do Linux), juntamente com comandos básicos do sistema, algumas ferramentas externas e a estrutura do diretório especial proc para a coleta de informação. Bem elaborada, esta ferramenta será uma solução capaz de atender todos os requisitos do problema.

A Figura 2.1 apresenta o modelo gerado com as informações obtidas nos Capítulos 1 e 2. O quadro central representa o arquivo principal dividido em funções. Esta característica auxilia a manutenabilidade da ferramenta. O quadro inferior representa a utilização dos recursos disponíveis no sistema para a geração do arquivo de saída (quadro superior esquerdo). A direita pode-se observar a representação do arquivo de configuração que concentra todas as alterações necessárias para personalização durante e após a instalação. Este modelo foi utilizado como base para a primeira fase do desenvolvimento da ferramenta Avisa.

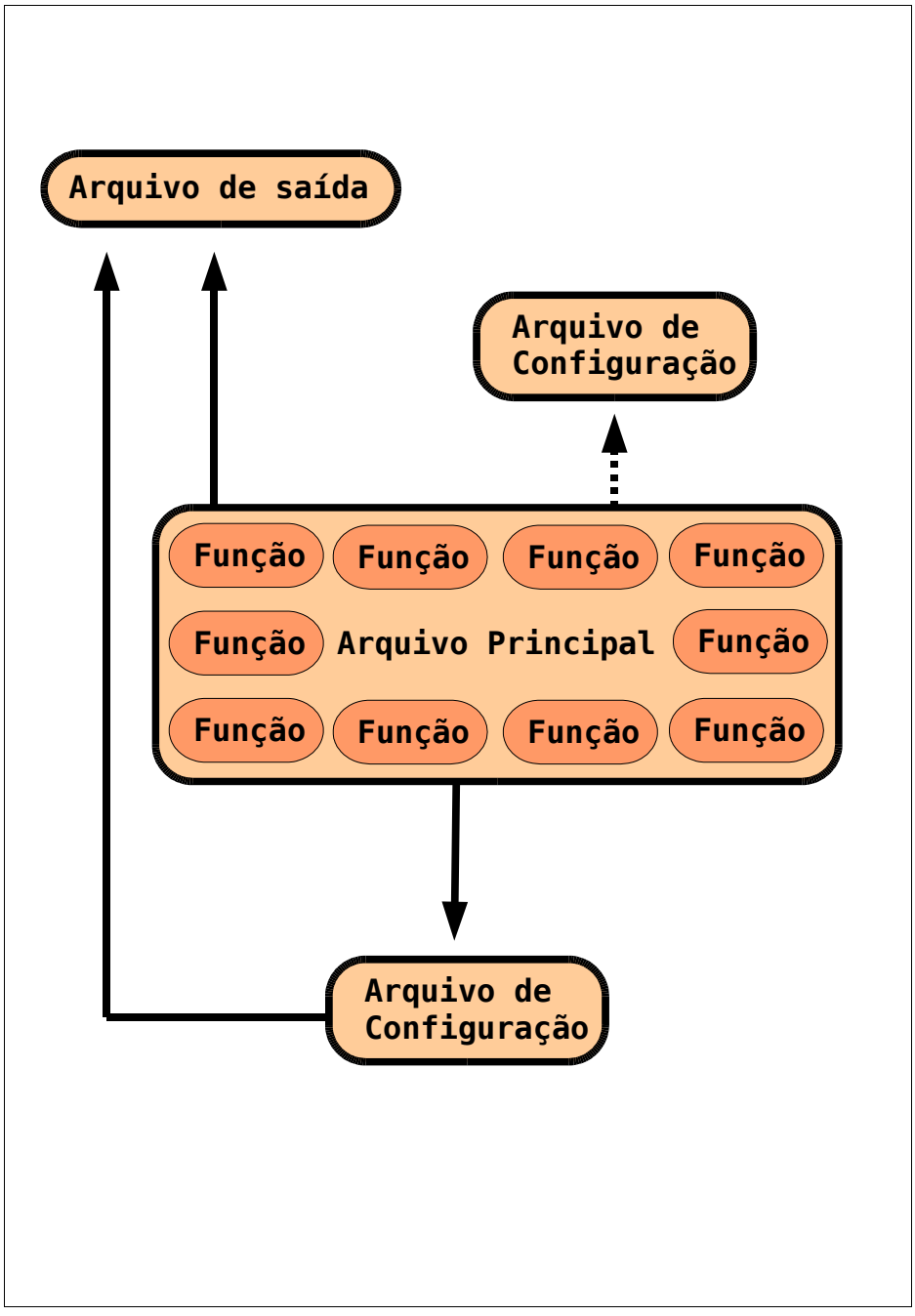


Figura 2.1: Modelo da ferramenta Avisa

Capítulo 3 - Revisão Bibliográfica

3.1 - Considerações iniciais

Este capítulo apresenta conceitos básicos utilizados no desenvolvimento da solução do problema relatado neste trabalho. Antes de começar a implementar a solução, é necessário reunir informações que possam ajudar a preparar uma base sólida de conhecimento sobre o assunto discutido, a fim de auxiliar a aplicação correta de conceitos e ferramentas, utilizando os recursos de forma precisa e eficiente na construção da solução. Esta etapa demonstra a seleção destes recursos e a indicação das fontes de pesquisa utilizadas.

3.2 - *Bourne Again Shell (Bash)*

O sistema operacional Linux é desenvolvido e organizado em múltiplas camadas. O *Shell* e o *Kernel* são camadas distintas nesta organização que se relacionam através de bibliotecas e chamadas do sistema (LOVE, 2004). Desta forma, o *Shell* é responsável pela interação entre usuário e sistema, fazendo o papel de interprete desta comunicação.

De acordo com os padrões *IEEE POSIX.2* e o *Open Group Shell*, a *Free Software Foundation*, desenvolveu o *Bash*, um *Shell* livre segundo os termos da *GNU GPL*. O *Bash* é o *Shell* padrão utilizado no sistema operacional Linux. Mais informações em (GNU,1998).

Muito mais que um interpretador de comandos, o *Bash* disponibiliza um conjunto de funções que permite o uso dos principais recursos de uma linguagem de programação, tornando possível criar *scripts* refinados que auxiliem principalmente na automação de tarefas realizadas pelo administrador.

3.2.1 - Redirecionamentos

O *Bash* permite a manipulação das entradas e das saídas de comandos. Basicamente, esta funcionalidade está ligada a três descritores de arquivos conhecidos como *standard input (stdin), standard output (stdout)* e *standard error (stderr).* No *Bash,* suas representações são respectivamente 0, 1 e 2.

Utilizando estes recursos, pode-se associar um arquivo para alimentar um comando, redirecionar a saída de um comando diretamente para um arquivo, eliminar informações sobre erros durante a execução de comandos, entre outras funções. Um exemplo pode ser visto na Figura 3.1.

Redireciona a saída do comando para o arquivo.
Comando 1> arquivo Ex.: `[and@park and]$ ls > lista`

Redireciona e inclui a saída do comando no final do arquivo.
Comando 1> >arquivo Ex.: `[and@park and]$ wc -l lista >> lista_com_total`

Redireciona a saída de erro do comando para o arquivo.
Comando 2> arquivo Ex.: `[and@park and]$ ls /root/ 2> sempermissao`

Redireciona e inclui a saída de erro do comando no final do arquivo.
Comando 2> >arquivo Ex.: `[and@park and]$ rm -f /root/* 2>> sempermissao`

Redireciona saída padrão e saída de erros para o arquivo.
Comando &> arquivo Ex.: `[and@park and]$ rm -rfv /tmp/* &> /dev/null`

Figura 3.1: Exemplo da utilização de redirecionamentos em *Bash*.

Um estudo mais detalhado sobre o tópico pode ser encontrado em (CAMARGO,2005) e (TLDP,2005).

3.2.2 - *Pipe*

O *pipe* " | " é um tipo de redirecionamento especial, ele executa um papel importante que é capturar a saída de um comando para servir como entrada de outro, tornando possível a associação de vários comandos em prol de um único resultado.

Por exemplo, para procurar uma palavra em uma arquivo texto pode-se utilizar *cat* <arquivo> | *grep -i* <palavra>. Desta forma a saída do comando *cat,* utilizado para ler o arquivo, foi redirecionada como entrada para o comando *grep*, responsável por encontrar a semelhança com a palavra solicitada.

3.2.3 - Operadores do *Bash*

Nesta seção, são apresentados os operadores mais utilizados para auxiliar na criação de *scripts*. A Figura 3.2 apresenta estes operadores.

Operadores Aritméticos	
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo
**	Exponenciação

Operadores Relacionais	
==	Igual
!=	Diferente
>	Maior
>=	Maior ou Igual
<	Menor
<=	Menor ou Igual

Operadores de Atribuição	
=	Atribui valor a uma variável
+=	Incrementa a variável por uma constante
-=	Decrementa a variável por uma constante
*=	Multiplica a variável por uma constante
/=	Divide a variável por uma constante
%=	Resto da divisão por uma constante
++	Incrementa em 1 o valor da variável
--	Decrementa em 1 o valor da variável

Operadores de BIT	
<<	Deslocamento à esquerda
>>	Deslocamento à direita
&	E de bit (AND)
|	OU de bit (OR)
^	OU exclusivo de bit (XOR)
~	Negação de bit
!	NÃO de bit (NOT)

Operadores Lógicos	
&&	E lógico (AND)
||	OU lógico (OR)

Operadores de BIT	
<<=	Deslocamento à esquerda
>>=	Deslocamento à direita
&=	E de bit
|=	OU de bit
^=	OU exclusivo de bit

Figura 3.2: Operadores

Compreender corretamente o funcionamento destes operadores é fundamental para fazer com que eles retornem os resultados esperados durante sua utilização. Exemplos interessantes podem ser obtidos em (NEVES,2003).

3.2.4 - Variáveis

Como qualquer linguagem de programação, o *Shell* pode trabalhar com variáveis. Além da utilização normal, o *Shell* disponibiliza variáveis especiais que podem armazenar dados sobre os comandos digitados, o que auxilia principalmente a criação de *scripts* interativos, onde parâmetros na linha de comando são fundamentais para as decisões da programação feita no *script.*

O retorno do código do último comando executado pode ser facilmente testado através da linha de comando. Para isso, basta digitar `ls -lha /root/; echo $?` , se não acontecer um erro, o valor retornado por `echo $?` será 0 (zero), acontecendo um erro, por exemplo, por falta de permissão de acesso, será retornado 1 (um). Na Figura 3.3, estas variáveis especiais são apresentadas.

Variável	***Parâmetros Posicionais***
`$0`	Parâmetro número 0 (nome do comando ou função)
`$1`	Parâmetro número 1 (da linha de comando ou função)
`...`	Parâmetro número N ...
`$9`	Parâmetro número 9 (da linha de comando ou função)
`${10}`	Parâmetro número 10 (da linha de comando ou função)
`...`	Parâmetro número NN ...
`$#`	Número total de parâmetros da linha de comando ou função
`$*`	Todos os parâmetros, como uma string única
`$@`	Todos os parâmetros, como várias strings protegidas
Variável	***Miscelânea***
`$$`	Número PID do processo atual (do próprio *script*)
`$!`	Número PID do último job em segundo plano
`$_`	Último argumento do último comando executado
`$?`	Código de retorno do último comando executado

Figura 3.3: Variáveis

3.2.5 - Blocos e agrupamentos

Um conceito que pode evitar problemas durante a criação de *Shell scripts* é saber empregar corretamente os símbolos: aspas simples, aspas duplas, apóstrofos, colchetes e parênteses. A utilização equivocada de um destes símbolos pode fazer com que as saídas de comparações de valores ou atribuições de variáveis retornem resultados errados ou simplesmente um erro abortando o *script*. Na Figura 3.4, são apresentadas as formas mais comuns de utilização destes recursos.

Sintaxe	Descrição	Exemplo
"..."	Protege uma string, mas reconhece $, \ e ` como especiais	"abc"
'...'	Protege uma string completamente (nenhum caractere é especial)	'abc'
$'...'	Protege uma string completamente, mas interpreta \n, \t, \a, etc	'abc\n'
`...`	Executa comandos numa subshell	`ls`
{...}	Agrupa comandos em um bloco	{ ls ; }
(...)	Executa comandos numa subshell	(ls)
$(...)	Executa comandos numa subshell, retornando o resultado	$(ls)
((...))	Testa uma operação aritmética, retornando 0 ou 1	((5 > 3))
$((...))	Retorna o resultado de uma operação aritmética	$((5+3))
[...]	Testa uma expressão, retornando 0 ou 1 (alias do comando 'test')	[5 -gt 3]
[[...]]	Testa uma expressão, retornando 0 ou 1 (podendo usar && e ll)	[[5 > 3]]

Figura 3.4: Blocos e agrupamentos

3.2.6 - Escapes reconhecidos pelo comando echo

Na elaboração de relatórios através de *Shell scripts* o comando `echo` é muito importante. Ele é responsável por imprimir as saídas que irão compor os cabeçalhos, as notas informativas ou resultados armazenados em variáveis. Para facilitar este trabalho, existe um conjunto de códigos que podem auxiliar na formatação destes resultados. A Figura 3.5 apresenta estes códigos com suas respectivas descrições.

Escape	*Mnemônico*	*Descrição*
\a	*Alerta*	Alerta (bipe)
\b	*Backspace*	Caractere Backspace
\c	*EOS*	Termina a string
\e	*Escape*	Caractere Esc
\f	*Form feed*	Alimentação
\n	*Newline*	Linha nova
\r	*Return*	Retorno de carro
\t	*Tab*	Tabulação horizontal
\v	*Vtab*	Tabulação vertical
\\	*Backslash*	Barra invertida \ literal
\nnn	*Octal*	Caractere cujo octal é nnn
\xnn	*Hexa*	Caractere cujo hexadecimal é nn

Figura 3.5: Escapes especiais

3.3 - Comandos úteis para manipulação de arquivos

3.3.1 - *cat*

O comando `cat` é um utilitário para manipulação de arquivos. Sua principal função é mostrar o conteúdo de um arquivo binário ou texto sem a necessidade de editá-lo. A sintaxe e um exemplo podem ser visto na Figura 3.6.

Sintaxe: `cat` [opções] [*diretório/arquivo*] [*diretório1/arquivo1*]

Exemplo: `[and@park and]$ cat -n lista`

Opções mais comuns:

-n, numera as linhas enquanto exibe o conteúdo do arquivo.

Figura 3.6: Exemplo da utilização do comando `cat`.

3.3.2 - *cp*

Comando responsável por realizar os procedimentos de cópia de arquivos e diretórios. A Figura 3.7 apresenta sua sintaxe.

Sintaxe: cp [*opções*] [*origem*] [*destino*]

Exemplo: `[and@park and]$ cp lista lista_de_comandos`

Opções mais comuns:

-f, Não pede confirmação, substitui todos os arquivos caso já existam.
-R, Copia arquivos, sub-diretórios , arquivos especiais FIFO e dispositivos.
-v, Mostra os arquivos enquanto estão sendo copiados.
-p, Preserva atributos do arquivo, quando possível.
-u, Copia somente os arquivos de origem mais novos que os de destino.

Figura 3.7: Exemplo da utilização do comando *cp*.

3.3.3 - *cut*

O comando *cut* pode ser utilizado para extrair trechos das linhas de um arquivo. De acordo com as opções solicitadas na sua execução, pode-se escolher um caracter como delimitador, dividindo as linhas em campos que podem ser facilmente manipulados. Um exemplo pode ser visto na Figura 3.8.

Sintaxe: cut [*opções*] [*arquivo*]

Exemplo: `[root@park and]# cat /etc/passwd|cut -d ":" -f1`

Opções mais comuns:

-f [campos], Mostra somente a lista de [campos].
-d [delimitador], Divide a linha em campos marcados pelo valor do delimitador.

Figura 3.8: Exemplo da utilização do comando *cut*.

Na segunda linha da Figura 3.8, o comando *cat* faz a leitura do arquivo *passwd* e em seguida sua saída é redirecionada para a entrada do comando *cut*. As linhas do arquivo *passwd* tem o seguinte formato, *usuário:senha:uid:gid:identificação:diretório:shell*. Assim, com as opções *-d ":"* e *-f1* completando a linha de comando, o resultado será a exibição apenas do primeiro campo do arquivo, que retornará o usuário.

3.3.4 - *tr*

Sua finalidade é alterar, apagar ou traduzir caracteres. Na Figura 3.9 pode-se observar dois exemplos práticos, no primeiro é apresentado como fazer uma transformação de caracteres minúsculos para maiúsculos em um arquivo. No segundo exemplo, o comando *tr* retira os espaços duplicados da saída proveniente do *ls* para que o *cut* possa ser utilizado para exibir os campos 2 e 6.

Sintaxe: tr [opções] [string]

Exemplo (1): `[and@park and]$ cat arquivo|tr a-z A-Z`

Exemplo (2): `[and@park and]$ ls -lha|tr -s " "|cut -d " " -f2,6`

Opções mais comuns:

-d, deleta os caracteres em *string1* da saída
-s, Remove caracteres repetidos contidos em *string1* da saída

Figura 3.9: Exemplo da utilização do comando *tr*.

3.3.5 - *grep*

Este é um comando muito útil para trabalhar em conjunto com outros comandos de manipulação de arquivos. Através dele pode-se procurar por um texto dentro de um arquivo.

No exemplo da Figura 3.10, o *grep* foi utilizado para procurar um usuário chamado anderson no arquivo `passwd`.

Sintaxe: `grep` [*expressão*] [*arquivo*] [*opções*]

Ex.: `[root@park and]# cat /etc/passwd|cut -d ":" -f1 |grep -i anderson`

Opções mais comuns:

-i, Ignora diferença entre maiúsculas e minúsculas no texto procurado e arquivo.

-v, Exibe as linhas que não contenham o padrão de coincidência.

-w, Exibe linhas que tenham somente palavras inteiras coincidentes com o padrão.

Figura 3.10: Exemplo da utilização do comando *grep*.

3.3.6 - *sed*

O comando *sed* (***S**tream **ED**itor*) é utilizado para aplicar várias transformações em um fluxo de texto que normalmente parte de um *pipe* que liga a saída de um outro comando ou de um arquivo ao *sed*. Ele faz a leitura da entrada linha por linha, posteriormente a edição de acordo com as regras pré selecionadas, e exibe o resultado na saída padrão. O *sed* possui inúmeros filtros que podem ser utilizados na manipulação de textos. No exemplo da Figura 3.11, o *sed* foi utilizado para transformar todos os espaços encontrados no arquivo texto em quebras de linhas. Mais informações em (THOBIAS, 2003).

Sintaxe: sed [opções] regras [arquivo]

Exemplo: `[root@park and]# sed 's/ /\n/g' arquivo_texto`

Opções mais comuns:
-n, envia para a saída padrão somente as linhas que atendam o critério da pesquisa.

Figura 3.11: Exemplo da utilização do comando sed.

3.3.7 - *awk*

Segundo (CAMARGO, 2005) e (NEVES, 2003), *awk* pode ser considerado uma linguagem de programação direcionada a processamento de textos. O comando *awk* tem uma ligação estreita com a linguagem *Shell* e é amplamente utilizado em *scripts* para resolver problemas de pesquisa de dados em arquivos texto e emissão de relatórios formatados. Como é apresentado na Figura 3.12, existem duas formas básicas de usar o *awk*, passando os parâmetros diretamente na linha de comando ou utilizando o arquivo com os comandos *awk*.

Sintaxe (1): awk '[padrão] [{procedimentos}]' arquivo(s)
Sintaxe (2): awk -f scritp arquivo(s)

Exemplo (1): `awk -F"IN=" '{ print $2 }' /var/log/messages`
Exemplo (2): `awk '$1 ~ /INICIO/,/FIM/ { print }' arquivo_texto`

Opções mais comuns:
-f [arquivo], realiza as instruções contidas em [arquivo]
-F [delimitador], utilizado para criar um delimitador e separar as linhas em campos

Figura 3.12: Exemplo da utilização do comando awk.

3.4 - Expressões Regulares

Expressão regular é uma composição de símbolos e caracteres especiais, que, agrupados com fragmentos de texto, formam uma expressão que é utilizada na pesquisa por palavras (ou grupo delas) que se adequem ao padrão definido. Por exemplo, se o conjunto de palavras for {casa, pássaro, casco, passo} e a expressão regular buscar por um padrão ss, obterá as palavras pássaro e passo.

3.4.1 - Metacaracteres

Metacaracteres são símbolos que têm funções específicas na pesquisa por expressões regulares, suas aplicações podem mudar dependendo do contexto no qual estão inseridos. Agrupados, eles podem combinar suas funções fazendo construções mais complexas. Segundo (NEVES,2003), os metacaracteres podem ser comparados a pequenas ferramentas no processo de pesquisa.

Importante ressaltar que existem diferenças na utilização dos metacaracteres através das linguagens e aplicativos. Assim, é necessário procurar informações sobre como o aplicativo ou a linguagem irá tratar aquele metacaracter. Por exemplo, com o comando *sed* é necessário usar "\" para o uso das chaves "{ }". Com *awk,* não existe esta necessidade. Mais detalhes sobre estas diferenças podem ser encontrados em (JARGAS,2001).

A Figura 3.13 apresenta um quadro com o símbolo do metacaracter, o nome e sua descrição utilizado pelo *awk*.

Meta	*Nome*	*Descrição*
.	ponto	Coringa de um caractere
[]	lista	Casa qualquer um dos caracteres listados
[^]	lista negada	Casa qualquer caractere, exceto os listados
?	opcional	A entidade anterior pode aparecer ou não (opcional)
*	asterisco	A entidade anterior pode aparecer em qualquer quantidade
+	mais	A entidade anterior deve aparecer no mínimo uma vez
{,}	chaves	A entidade anterior deve aparecer na quantidade indicada
^	circunflexo	Casa o começo da linha
$	cifrão	Casa o fim da linha
\b	borda	Limita uma palavra (letras, números e sublinhado)
\	escape	Escapa um meta, tirando seu poder
|	ou	Indica alternativas (usar com o grupo)
()	grupo	Agrupa partes da expressão, é quantificável e multinível
\1	retrovisor	Recupera o conteúdo do grupo 1
\2	retrovisor	Recupera o conteúdo do grupo 2 (segue até o \9)
.*	coringa	Casa qualquer coisa, é o tudo e o nada
??	opcional NG	Idem ao opcional comum, mas casa o mínimo possível
*?	asterisco NG	Idem ao asterisco comum, mas casa o mínimo possível
+?	mais NG	Idem ao mais comum, mas casa o mínimo possível
{}?	chaves NG	Idem às chaves comuns, mas casa o mínimo possível

Figura 3.13: Metacaracteres compatíveis com *awk*

A maneira mais fácil de entender como funciona e como se aplica a pesquisa através de expressões regulares é praticando. Inúmeras dicas e instruções (das mais simples às mais complexas) podem ser encontradas em (JARGAS,2001).

3.5 - *Smartmontools*

Smartmontools (SMT, 2004) é um pacote que contém um aplicativo chamado `smarctl` e um *daemon* chamado `smartd`, ambos para verificação de erros no disco rígido. Este pacote foi desenvolvido por um professor de física da Universidade de Wisconsin chamado Bruce Allen. Seu interesse em desenvolver este projeto estava diretamente ligado a um outro projeto, a utilização de *clusters* instalados com Linux para realizar estudos sobre ondas gravitacionais no início do universo.

A fim de diminuir a possibilidade de paradas no *cluster* por defeitos em discos rígidos, ele estudou o funcionamento da tecnologia SMART (*Self-Monitoring, Analysis and Reporting Technology*) disponível em discos mais recentes (como os de padrão ATA-3 em diante) e criou uma ferramenta capaz de verificar os atributos fornecidos pelo SMART e, assim, avaliar o estado do disco rígido.

A ferramenta pode ser utilizada para verificações periódicas (`smartctl`) ou para um acompanhamento constante (`smartd`). Os testes podem ser realizados sem o perigo de corrompimento nos dados e existem vários tipos de parâmetros que podem ser ajustados conforme a vontade do administrador. Antes da utilização, é importante observar se o modelo do disco rígido está na lista de compatíveis. Caso contrário, as informações geradas podem não ser reais. A Figura 3.14 mostra como verificar se o disco é compatível.

```
prompt# smartctl -i /dev/hda

 Device Model: IC35L120AVV207-0
 Serial Number: VNVD02G4G3R72G
 Firmware Version: V24OA63A
 Device is: In smartctl database [for details use: -P show]
 ATA Version is: 6
 ATA Standard is: ATA/ATAPI-6 T13 1410D revision 3a
 SMART support is: Available - device has SMART capability.
 SMART support is: Enabled
```

Figura 3.14: Verificação da compatibilidade do disco.

Informações sobre instalação e um fórum de discussão podem ser encontrados em (SMT, 2004).

3.6 - *Tripwire*

O *Tripwire* é uma ferramenta desenvolvida para verificação de modificações ocorridas em um sistema de arquivos. Suas características o definem como um *IDS* (*Intrusion Detection System*) de *host,* que diferentemente do *IDS* de rede, não trabalha como *daemon* e não realiza bloqueios, apenas registra as ocorrências. Outra diferença é o *IDS* de *host* ter a possibilidade de registrar erros de pessoas autorizadas ao acessar o sistema, isto quer dizer que nem sempre a detecção do erro será proveniente de um intruso.

Após instalado, o Tripwire cria um banco de dados com informações dos arquivos que devem ser monitorados e, a partir deste banco, poderão ser feitas checagens com a finalidade de identificar qualquer alteração ocorrida nestes arquivos. O banco de dados é armazenado em forma criptografada, garantindo que não possa ser alterado sem que seja usada a chave criptográfica e fornecida a senha gerada na instalação.

O Tripwire permite que a base de seus registros seja atualizada, assim uma alteração proposital no sistema de arquivos, não aparecerá em todos os relatórios que forem emitidos após aquela alteração. Porém, é importante observar que esta tarefa deve ser feita somente pelo administrador e com muita cautela, pois, uma vez feita a atualização, a ferramenta assumirá que o fato ocorrido era esperado e que aquelas novas características do sistema de arquivos são corretas.

A ferramenta é de fácil utilização e seus relatórios permitem a visualização de um quadro geral com as informações divididas por categorias e caso ocorram alterações, detalhes sobre cada uma delas. Mais informações podem ser encontradas em (MELO; TRIGO, 2004).

3.7 – Considerações finais

Esta etapa foi dedicada a reunir material para servir de base na construção da solução. Neste capítulo, foram apresentados alguns conceitos básicos retirados do material selecionado, e, apesar da forma superficial com que estes conceitos foram abordados, as referências citadas em cada um dos tópicos contém informações mais detalhadas que podem ser consultadas.

Capítulo 4 - Construção do script avisa.sh

4.1 – Considerações iniciais

Este capítulo apresenta os procedimentos tomados durante o planejamento e implementação do *script* `avisa.sh`. As partes mais relevantes de seu código são demonstradas e comentadas, a fim de facilitar a compreensão de seu funcionamento. O *script* `avisa.sh` é a parte da ferramenta responsável por coletar os dados do sistema e transformá-los em um relatório único. Para isso, ele trabalhará executando os comandos e ferramentas externas (apresentadas no capítulo anterior) que terão suas saídas direcionadas (quando necessário formatadas) para um arquivo chamado `registro.log`. Um arquivo de configuração chamado `avisa.conf`, idealizado para auxiliar na configuração da ferramenta, também é apresentado.

4.2 - Planejamento inicial

Partindo do princípio que o objetivo havia sido definido e estava de acordo com as solicitações do enunciado do problema, o próximo passo foi definir quais comandos e ferramentas seriam utilizados na construção do *script*. Para tal, foram discriminadas todas as informações que o relatório deveria registrar e os comandos equivalentes necessários. Como pode ser visto nas Figuras 4.1 e 4.2.

Hardware	***comandos***
processador	`cat /proc/cpuinfo | head -n8`
Disco rígido	`cat /proc/ide/hda/model`
Unidade de cdrom	`cat /proc/ide/hdc/model`
memória	`livre=`cat /proc/meminfo | head -n2 |tail -n1 \` `| tr -s " " | cut -d " " -f2` `usada=`cat /proc/meminfo |head -n1 |tr -s " " \` `| cut -d " "  -f2` `total=`expr $livre + $usada` `echo "$total KB"`
placa de rede	`lspci | grep -i eth`
placa de vídeo	`lspci | grep -i vga`

Figura 4.1: Hardware - Solicitações do problema e comandos referentes

Informações do sistema	*Comandos/ferramentas*
distribuição	`cat /etc/conectiva-release`
versão do kernel	`uname -r`
versão do iptables	`iptables -V | cut -d " " -f2` `rpm -qf `which iptables``
versão do squid	`squid -v | cut -d " " -f4 | head -n1` `rpm -qf `which squid``
versão do ssh	`rpm -qf `which sshd` | cut -d "-" -f 3` `rpm -qf `which sshd``
versão do openssl	`rpm -qa | grep -i openssl | cut -d "-" -f 1`
árvore de processos	`ps ufax`
módulos carregados	`lsmod`
utilização da memória	`cat /proc/meminfo`
utilização do HardDisk	`df -Th`
última reinicialização	`last |grep -i reboot |head -n1 |tr -s " " |\` `cut -d " " -f5,6,7,8`
tempo que o servidor está ligado	`tempo=`uptime | grep -i day`` `if [ "$tempo" == "" ]` `then` `horas=`uptime | tr -s " " | cut -d " " -f4 \` `|cut -d ":" -f1` `minutos=`uptime |tr -s " " |cut -d " " -f4 \` `| cut -d ":" -f2 |  tr -d ","` `echo "Servidor esta ativo a $horas hora(s) e \` `$minutos minuto(s)"` `else` `dias=`uptime |tr -s " " |cut -d " " -f4` `horas=`uptime |tr -s " " |cut -d " " -f6 \` `|cut -d ":" -f1` `minutos=`uptime |tr -s " " |cut -d " " -f6 | \` `cut -d ":" -f2 | tr -d ","` `echo "O servidor esta ativo a $dias dia(s), \` `$horas hora(s) e $minutos minuto(s)"` `fi`
informações sobre login	`last | head -n3`
últimos comandos	`tail -n5 /root/.bash_history`
Alteração no sistema de arquivos.	Para esta tarefa será necessário a utilização de um *IDS* de *host*, que será definido mais adiante.

Figura 4.2: Sistema - Solicitações do problema e comandos referentes

Durante os testes com os comandos referentes às solicitações, algumas questões foram observadas:

- Para um informação exata do disco rígido e da unidade de cdrom, o *script* precisa verificar em qual *interface IDE* o disco rígido está localizado, e se existe ou não uma unidade de cdrom instalada. Prevendo situações como esta durante o andamento do projeto, foi decidido criar um arquivo de configuração, onde o administrador pudesse configurar parâmetros sem precisar alterar as linhas diretamente no *script*. Este arquivo de configuração será apresentado posteriormente;

- Para registrar corretamente o tempo que o servidor está ativo, é necessário verificar se ele está ligado a mais de um dia ou não, isto porque a saída do comando é diferente em cada uma das situações. Assim, o comando cut poderia pegar o campo errado, interferindo na saída para o relatório. Por isso, foi utilizada uma estrutura condicional nesta solução para garantir a saída correta da informação desejada;

- Durante a programação, acontecem erros de sintaxe ou de digitação que certamente trazem algum tipo de dificuldade na resolução. Por isso, o *script* foi dividido em funções, o que facilita a criação de novos módulos e a realização de testes isolados.

A estrutura do *script* foi idealizada como demonstrado na Figura 4.3:

```
#!/bin/bash

Principal () {
       função_1
       função_2
       ...
             função_n

}
função_1 () { }
função_2 () { }
...
função_n () { }
Principal
```

Figura 4.3: Estrutura idealizada para o script `avisa.sh`

Após a finalização das duas funções que haviam sido criadas de acordo com as informações exibidas nas Figuras 4.1 e 4.2 (estas funções foram nomeadas como HW e Sistema), mais duas funções foram elaboradas. A função Envio, que será responsável pela emissão do arquivo gerado pelo *script* para o destino, que será escolhido durante a configuração do arquivo `avisa.conf`. A função Backup responsável por fazer e rotacionar as cópias de segurança desta ferramenta.

De acordo com o planejamento, uma forma de registrar alterações no sistema de arquivo seria interessante para este *script*. A ferramenta escolhida para esta questão foi o *Tripwire*, um sistema de detecção de intrusos de *host*, que tem sua instalação e utilização explicados no anexo.

Fornecendo um relatório completo de alterações na estrutura de diretórios do sistema, o Tripwire atende à necessidade de registro de alteração, criação ou remoção de arquivos. Esta tarefa, primeiramente definida como parte da função Sistema, passará a compor uma nova função chamada Tripwire.

Durante a elaboração das funções citadas, surgiu uma nova questão que não fazia parte do problema inicial para qual este *script* estava sendo escrito. A nova idéia seria a inclusão de registro de informações sobre o *firewall* e sobre o servidor *proxy*, agregando valor ao *script*.

Dentro do domínio das alternativas de solução para estas duas novas atribuições do *script*, cogitou-se a possibilidade da utilização de um *IDS* de rede, por exemplo o *Snort*[7], e um analisador de *log* do *Squid* como o *Sarg*[8]. Porém, pesquisando sobre estas ferramentas, foi constatado que estas soluções entrariam em conflito com a proposta inicial do projeto, uma vez que além de exigir recursos extras de processamento e memória, seria necessário a instalação de novos serviços como o *MySQL*[9] e o *Apache*[10] no servidores.

Sem interferir ou alterar o propósito da solução, foi realizada uma nova listagem de tarefas que o *script* poderia agregar, bem como os comandos equivalentes necessários, como descrito na Figura 4.4 e 4.5.

7 Snort – www.snort.org
8 Sarg – sarg.sourceforge.net
9 MySQL – www.mysql.com
10 Apache – www.apache.org

Informações sobre a rede	*comandos*
Total de pacotes enviados Total de pacotes recebidos Colisões	`ifconfig <interface>`
Conexões estabelecidas	netstat
Bloqueios do *firewall*	Devido a complexidade da tarefa, ela formará uma nova função e será explicada posteriormente.

Figura 4.4: Rede - Solicitações do problema e comandos referentes

Informações sobre o proxy	*comandos*
Tentativas de acesso a *sites* bloqueados.	Devido a complexidade da tarefa, ela será explicada posteriormente.
Pesquisa por palavras chaves nos endereços acessados.	Devido a complexidade da tarefa, ela será explicada posteriormente.

Figura 4.5: *Proxy* - Solicitações do problema e comandos referentes

Mesmo não utilizando novas ferramentas externas, o autor irá trabalhar com os recursos existentes para gerar novos dados sobre o servidor no relatório. Estes dados serão uma lista de bloqueios proveniente do *firewall* existente e um relatório sobre o *Squid* como descrito na Figura 4.5.

Após avaliação sobre a viabilidade das novas atribuições, o *script* passou a ter a estrutura como demonstrada na Figura 4.6:

```
#!/bin/bash

Principal () {
       HW
       Rede
       Sistema
       Smart
       Tripwire
       Firewall
       Squid
       Envio
       Backup
 }

HW () {
}

Rede () {
}

Sistema () {
}

Smart () {
}

Tripwire () {
}

Firewall () {
}

Squid () {
}

Envio () {
}

Backup () {
}

Principal
```

Figura 4.6: Representação da estrutura do script `avisa.sh`

4.3 - O arquivo de configuração avisa.conf

Com o aumento das atividades da ferramenta, muitas configurações deveriam ser alteradas no código do *script* principal, devido a esta dificuldade foi idealizado um arquivo de configuração que concentrasse estas alterações. A estrutura do arquivo de configuração do *script* foi planejada de uma forma que cada função pudesse acessar uma seção referente a ela no arquivo `avisa.conf`, coletando informações necessárias para seu funcionamento.

Como o comando *awk* tem a propriedade de ler determinado trecho de um arquivo baseado em delimitadores, sua utilização é pertinente nesta situação. Um exemplo das seções é demonstrada na Figura 4.7:

```
#Hardware
#
# Lista dos Hardwares a serem detectados, descomente as linhas
de acordo com sua #necessidade
# Por exemplo, para gerar o relatório sobre o processador,
deixe a entrada PROCESSADOR #descomentada.

PROCESSADOR
HD_Primary_Master
#HD_Primary_Slave
#HD_Secondary_Master
#HD_Secondary_Slave
#CDROM_Primary_Master
#CDROM_Primary_Slave
#CDROM_Secondary_Master
#CDROM_Secondary_Slave
MEMORIA
PLACADEREDE
PLACADEVIDEO

#/Hardware
```

Figura 4.7: Representação da estrutura do arquivo de configuração `avisa.conf`.

Usando as linhas de comentário `#Hardware` e `#/Hardware`, a seção estará delimitada, o administrador então definirá qual *hardware* deve ser registrado pela função HW descomentando as linhas dentro dos limites da seção.

Para registrar, por exemplo, a existência de uma unidade de cdrom na

IDE secundária posição *Master*, basta descomentar a linha `#CDROM_Secondary_Master`.

Resumindo, para cada função do *script* que necessite de alguma configuração especial, existe uma seção no arquivo `avisa.conf` que será acessada pelo *script* através do comando *awk* que usará as linhas `#<nome_da_função>` e `#/<nome_da_função>` como delimitadores.

4.4 - Detalhes sobre atividades das funções

4.4.1 - Função HW

Esta função tem por finalidade registrar as informações sobre o *hardware* do servidor. Como foi mencionado, a função se baseia nas linhas descomentadas do arquivo de configuração para então decidir sobre qual periférico irá registrar informações. A Figura 4.8 mostra a linha onde o comando *awk* solicita a leitura do arquivo `avisa.conf`.

Uma observação, a "\" na primeira linha indica que não foi possível continuar o comando na mesma linha devido a formatação da página, então, sempre que aparecer este símbolo no final de uma linha de comando, deve ser considerado que ambas as linhas formam uma só linha de comando.

```
for hardware in `awk '$1 ~ /#Hardware/,/#\/Hardware/ \
{ print }' $CAMINHO/avisa.conf |  grep -iv '#'`
```

Figura 4.8: Leitura da sessão Hardware no arquivo de configuração `avisa.conf`

Logo após esta linha, é utilizado, a estrutura de programação *case* e comparado o valor da variável *hardware*. Desta forma a função HW encaminha suas saídas para o relatório.

4.4.2 - Função Rede

O comando `ifconfig` trás informações valiosas para avaliação do comportamento da rede. Por isso não foi utilizada a estrutura do diretório *proc*

para a coleta destas informações nesta função.

O princípio para definição das *interfaces* é o mesmo utilizado pela função HW. É feito uma leitura do arquivo de configuração `avisa.conf` e de acordo com as *interfaces* registradas lá, o *script* executa o comando *`ifconfig`* em cada uma delas. Detalhes podem ser vistos na Figura 4.9.

```
 for interface in `awk '$1 ~ /#Interfaces/,/#\/Interfaces/  \
{ print }' $CAMINHO/avisa.conf |  grep -iv '#'`
  do
    echo "Relatório da Interface $interface :" >> $SAIDA
    echo >> $SAIDA
     ifconfig $interface >> $SAIDA
    echo
"------------------------------------------------------------ \
----------------------" >> $SAIDA
    echo >> $SAIDA
  done
```

Figura 4.9: Captura das informações das interfaces no arquivo `avisa.conf`.

4.4.3 - Função Sistema

A função Sistema é responsável por coletar a maior parte das informações deste *script*. Para cumprir a exigência de minimizar o impacto de novas instalações nos servidores e deixar o *script* mais simples possível, foram usados comandos como demonstrados na Figura 4.10.

```
echo -n "Versão do Iptables: " >> $SAIDA
iptables -V | cut -d " " -f2 >> $SAIDA
echo -n "Referente ao pacote " >> $SAIDA
rpm -qf `which iptables` >> $SAIDA
echo "------------------------------------------------------"
>> $SAIDA
echo >> $SAIDA
```

Figura 4.10: Trecho responsável por coletar informações sobre o iptables.

Por não utilizar recursos complexos, a única parte desta função que

merece destaque pode ser visualizada na Figura 4.11.

Durante os testes na fase da definição dos comandos que atenderiam cada funcionalidade da lista estabelecida, como demonstrado na Figura 4.2, verificou-se a necessidade de um código mais refinado para a definição correta do registro de dias ou horas do estado ativo do servidor .

Para isso, a primeira tarefa desta parte do *script* é saber se o sistema está ligado ou não a mais de um dia. Para tomar esta decisão, uma variável denominada *tempo* recebe a saída do comando *uptime | grep -i day*. Se o resultado da comparação *if ["$tempo" == ""]* for verdadeiro, pode-se afirmar que o servidor não está ligado a mais de um dia, justamente porque a palavra *day* só aparece quando se completa o primeiro dia de atividade. Caso o resultado retorne falso, indica que o servidor está ligado a mais de um dia.

```
tempo=`uptime | grep -i day`
if [ "$tempo" == "" ]
   then
   horas=`uptime|tr -s " "|cut -d " " -f4|cut -d ":" -f1`
   minutos=`uptime|tr -s " "|cut -d " " -f4|cut -d ":" -f2|tr -d ","`
   echo "Este servidor esta ativo a $horas hora(s) e $minutos \
minuto(s)" >> $SAIDA
   else
   dias=`uptime|tr -s " "|cut -d " " -f4`
   horas=`uptime|tr -s " "|cut -d " " -f6|cut -d ":" -f1`
   minutos=`uptime|tr -s " "|cut -d " " -f6|cut -d ":" -f2|tr -d ","`
   echo "Este servidor esta ativo a $dias dia(s), $horas hora(s) e \
$minutos minuto(s)"  >> $SAIDA
fi
```

Figura 4.11: Parte responsável por registrar o tempo de atividade do servidor.

4.4.4 - Função Smart

A função Smart é composta pelo pacote Smartmontools que após ser instalado, oferece uma verificação do disco rígido informando se está ou não confiável. Apesar de não haverem dados importantes armazenados no servidor, um problema no disco rígido pode acarretar no interrompimento do serviço, logo sua utilização nesta ferramenta é bem vinda. Os procedimentos para a instalação da ferramenta *smartmontools* pode ser encontrada no anexo.

4.4.5 - Função Tripwire

Assim como a função Smart, esta função é composta por uma ferramenta externa chamada *Tripwire*, responsável por registrar as alterações no sistema de arquivos do servidor. A atividade desta função é apenas direcionar a saída desta ferramenta para o relatório final. Informações sobre sua instalação e configuração podem ser encontradas no anexo.

4.4.6 - Função Firewall

A título de acompanhar as atividades do *firewall* configurado no servidor e com a finalidade de registrar material para um estudo futuro sobre os bloqueios efetuados pelo *firewall* de um servidor em produção, esta função foi idealizada. Basicamente sua tarefa é separar os registros de *log* sobre os bloqueios realizados, filtrar endereços IP e portas, comparar estes endereços capturados com uma base de registro de reincidências e contabiliza-los. Além disto, deve registrar tentativas de acesso via porta 22 (*ssh*) que falharem durante o fornecimento da senha.

Seu funcionamento é baseado na utilização de arquivos temporários gerados com os dados do `syslog`, onde são aplicados filtros para capturar as informações desejadas. Estes filtros são comandos como `cut`, `tr` e `awk` que permitem a retirada das informações destes arquivos de uma forma precisa. A linha de comando que captura os registros do *log* pode ser visto na Figura 4.12.

```
cat /var/log/messages | grep -i INPUT-Bloqueado | tr -s " " |\
> $CAMINHO/tmp_messages
```

Figura 4.12: Captura dos registros do *log*

Na linha de comando mostrada, são registradas todas as entradas com a marcação INPUT-Bloqueado (marcação definida através da opção `–log` do `iptables`), que são justamente as incidências de bloqueios realizados pelo *firewall*. Sabe-se que o registro dos *logs* sofre um processo de rotação para que o disco não fique lotado com estas informações, mas este procedimento não é diário, então, no arquivo `tmp_messages` (Figura 4.12) com certeza existirá registros de dias anteriores ao que se está executando o *script*.

A ferramenta Avisa foi desenvolvida para execução diária, logo os

registros devem ser apenas referentes a data do dia corrente. Para que o *script* não contabilize dados incorretamente, uma nova função para cuidar do registro das datas foi criada e será explicada mais adiante (Função Data).

Por enquanto, basta saber que uma variável chamada mes_dia é responsável por fornecer a data corretamente. De posse do arquivo `tmp_messages`, é aplicado um filtro, através do comando *awk* para a retirada dos endereços de origem dos bloqueios realizados pelo *firewall*. Um novo arquivo chamado de `tmp_enderecos` é criado com todos os endereços registrados desde o início do dia até aquele exato momento. (Figura 4.13)

```
awk -F"SRC=" '{ print $2 }' $CAMINHO/tmp_messages|cut -d " " \
-f1|sort -n|uniq > $CAMINHO/tmp_enderecos
```

Figura 4.13: Utilização do awk para gerar o arquivo `tmp_enderecos`

O arquivo `tmp_enderecos` é comparado com um arquivo chamado `reincidentes` que é um arquivo auxiliar deste *script*. Ele é criado vazio durante a instalação e, depois, passará a guardar os endereços IP que são bloqueados pelo *firewall.* Desta forma, um cadastro com todos os endereços IP bloqueados será criado.

Sua entrada é composta por "endereço-número_de_incidências_diárias" por exemplo, na primeira utilização do *script*, o arquivo `tmp_enderecos` será totalmente reescrito para o arquivo `reincidentes`, seguindo o padrão "endereço-número_de_incidências_diárias" onde "número_de_incidências" receberá "1", pois este será o primeiro registro do endereço. Um exemplo de como será o conteúdo do arquivo pode ser visto na Figura 4.14.

```
200.123.233.14-1
81.100.2.10-1
69.101.87.3-1
```

Figura 4.14: Registros do arquivo `reincidentes`.

No segundo dia de uso, caso sejam registrados novos bloqueios com algum dos endereços registrados no arquivo `reincidentes`, sua entrada passará de "endereço-1" para "endereço-2", informando que é a segunda vez que ele aparece nos registros de *log* do *firewall*. O resultado é exibido na Figura 4.15.

```
200.123.233.14-1
81.100.2.10-2
69.101.87.3-1
```

Figura 4.15: Registros do arquivo `reincidentes` no segundo dia de utilização.

Para que a utilização do cadastro de incidências seja compreendido, faz-se necessário algumas explicações. Os servidores de compartilhamento de acesso a internet que a ferramenta Avisa atenderá, a princípio, não ficariam ligados 24 horas por dia. O procedimento normal seria ligá-los no início do expediente e desligá-los no final. Porém, foi verificado que em alguns casos os clientes deixavam o servidor ligado de um dia para o outro. Por este motivo, foi definido um tipo de funcionamento para cada caso.

Foi adicionado outro arquivo auxiliar chamado `contador` que será utilizado para definir algumas ações durante a execução do script `avisa.sh`. Além do arquivo `contador`, uma seção chamada Contador foi criada no arquivo `avisa.conf`. Os valores possíveis para esta seção são "1" e "4". O valor "1" indica que o *script* `avisa.sh` está instalado em um servidor que funciona entre o início e o final do expediente do cliente. O valor "4" indica que o *script* esta instalado em um servidor que trabalha 24 horas por dia.

Foi definido que o *script* emita um relatório no fim do expediente para os servidores que funcionam por período de expediente, e quatro relatórios durante o dia para os servidores que ficam ligados ininterruptamente.

Na função Firewall, o arquivo `contador` tem papel fundamental para que os endereços registrados no arquivo `reincidentes` sejam consistentes.

No funcionamento por expediente, não ocorre inconsistência nas informações do registro da função Firewall, pois o *script* `avisa.sh` será executado apenas uma vez por dia. Isto significa que todas as vezes que houver uma comparação entre o arquivo `tmp_enderecos` e o arquivo `reincidentes`, os endereços IP coincidentes deverão ter seus índices atualizados.

No funcionamento do *script* em servidores ligados 24 horas por dia, onde ele será executado por 4 vezes neste período, a função Firewall não pode simplesmente contabilizar o endereço a cada execução. Isto porque se um bloqueio foi feito, por exemplo, no endereço 69.101.2.10 na primeira execução do *script* às oito horas da manhã, ela será novamente constatada durante as

outras três execuções, o que resultaria no final do dia, na seguinte entrada no arquivo `reincidentes`, 69.101.2.10-4, onde o valor correto deveria ser 69.101.2.104-1.

Para resolver este problema, o *script* `avisa.sh` fará uma leitura do arquivo `contador` durante a chamada à função Principal e armazenará o valor em uma variável de mesmo nome. (contador). Assim, durante a execução, o *script* poderá tomar decisões baseadas no conteúdo desta variável.

Como pode ser visto na Figura 4.16, a variável i recebe, um por vez, todos os endereços IP registrados no arquivo `tmp_enderecos` e a variável j recebe a primeira parte da entrada registrada no arquivo `reincidentes`, ou seja, o endereço IP. (Lembrando que a entrada neste arquivo é composta por "endereço-número_de_incidências ").

Acontece a comparação entre as variáveis i e j. Sendo verdadeira a igualdade, significa que o endereço IP é reincidente e seu índice de ocorrências deve ser atualizado no arquivo `reincidentes`. Antes que isto aconteça, a variável y recebe o número de incidências diárias do endereço que está sendo checado. A variável contador é checada e, caso seu valor seja igual a "1", a variável y é incrementada. Se a variável contador for diferente de "1", é feita uma comparação verificando se ela é menor ou igual ao valor da seção Contador no arquivo de configuração `avisa.conf`. Isto significa que se a variável contador contiver o valor 2, 3 ou 4, a variável y não será incrementada, preservando a consistência do relatório de reincidências diárias dos endereços IP bloqueados.

```
for i in `cat $CAMINHO/tmp_enderecos`
  do
    j=`cat $CAMINHO/reincidentes|cut -d "-" -f1|grep -iw $i`
    if [ "$i" == "$j" ]
     then
      y=`cat $CAMINHO/reincidentes|grep -iw $i|cut -d "-" -f2`
      if [ $contador -eq 1 ]
       then
        y=`expr $y + 1`
       else
        if
        [ $contador -le `awk '$1 ~ /#Contador/,/#\/Contador/ \
{ print }' $CAMINHO/avisa.conf  | grep -iv '#'` ]
        then
         y=$y
        fi
      fi
```

Figura 4.16: Detalhes sobre a função Firewall

A continuação da estrutura *for* da Figura 4.16 pode ser observado na Figura 4.17. Para que a atualização do arquivo `reincidentes` aconteça corretamente, é gerado um arquivo `tmp_reincidentes` e repassado para dentro dele todos os endereços que não foram encontrados no arquivo `tmp_enderecos` naquela execução. Em seguida, são adicionados os endereços reincidentes com seus índices atualizados. Caso seja a primeira vez que o endereço esteja sendo registrado, ele recebe o índice com valor "1". Juntamente com a atualização do arquivo `reincidentes`, é gerado um arquivo chamado `tmp_end4report` que será utilizado na construção do relatório de saída do *script* `avisa.sh`.

```
   > $CAMINHO/tmp_reincidentes
   cat $CAMINHO/reincidentes|grep -iv $i \
   >> $CAMINHO/tmp_reincidentes
   cat $CAMINHO/tmp_reincidentes > $CAMINHO/reincidentes
   echo $i-$y >> $CAMINHO/reincidentes
   echo $i-$y >> $CAMINHO/tmp_end4report
    else
     echo $i-1 >> $CAMINHO/reincidentes
     echo $i-1 >> $CAMINHO/tmp_end4report
    fi
done
```

Figura 4.17: Continuação da estrutura *for* exibida na figura 4.16.

O próximo passo é utilizar o arquivo `tmp_end4report` e associar a cada entrada as portas que estes endereços IP tentaram alcançar no servidor. Em seguida, gerar o relatório no formato como demonstrado na Figura 4.18.

```
    Endereços IP        |  Incidências   | Portas
========================|================|====================
                        |                |
------------------------|----------------|--------------------
         1.138.124.20   |      1         |  1025 1026
------------------------|----------------|--------------------
       14.154.136.211   |      1         |  1025
------------------------|----------------|--------------------
        15.232.22.196   |      1         |  1025 1026
------------------------|----------------|--------------------
```

Figura 4.18: Saída gerada pela função Firewall

Como pode ser visto na Figura 4.19, para obter o resultado final, uma série de medidas devem ser realizadas. Para cada entrada do arquivo `tmp_end4report`, primeiro é feito uma leitura para que as portas daquele endereço sejam registradas no arquivo `tmp_ep`.

```
for i in `cat $CAMINHO/tmp_end4report|cut -d "-" -f1`
  do
   cat $CAMINHO/tmp_messages|grep -iw $i|awk -F"DPT=" \
   '{ print $2 }'|cut -d " " -f1 > $CAMINHO/tmp_ep
   echo "> `cat $CAMINHO/tmp_end4report | grep -i $i`" \
   >> $CAMINHO/tmp_epf
   echo "-" >> $CAMINHO/tmp_epf
   sort -n $CAMINHO/tmp_ep | uniq >> $CAMINHO/tmp_epf
 done
```

Figura 4.19: Procedimentos para gerar a saída apresentada na Figura 4.18

Em seguida, são registrados no arquivo `tmp_epf` o sinal de maior (>), o endereço, o sinal de menos (-) e as portas referentes. Um exemplo pode ser visto na Figura 4.20.

```
>
1.138.124.20
-
1025
1026
>
14.154.136.211
-
1025
>
15.232.22.196
-
1032
```

Figura 4.20: Conteúdo do arquivo tmp_epf

Com o arquivo `tmp_epf` construído, fora utilizados os comandos *`sed`* e *`awk`* para terminar a tarefa. O código pode ser visto na Figura 4.21.

Na primeira utilização, o comando *`sed`* lê o arquivo `tmp_epf` e sobrescreve-o sobre `tmp_ep` trocando cada quebra de linha por um espaço, gerando uma única linha com todas as entradas antes registradas.

Na segunda utilização, o comando `sed` lê o arquivo `tmp_ep` e troca todos os sinais de maior (>) por uma quebra de linha, redirecionando sua saída para um novo arquivo chamado `tmp_ep4awk`.

```
sed ':a;$!N;s/\n/ /;ta;' $CAMINHO/tmp_epf > $CAMINHO/tmp_ep
sed 's/>/\n /g' $CAMINHO/tmp_ep > $CAMINHO/tmp_ep4awk
```

Figura 4.21: Utilização do sed para filtrar os dados nos arquivos temporários.

Neste momento do desenvolvimento, para que os dados pudessem ser enviados para o relatório final com um formato adequado, foi necessária a criação de mais um arquivo auxiliar, o `formata.awk`. Este arquivo contém os parâmetros para formatar a saída e exibi-la com mostrado na Figura 4.18. Seu conteúdo pode ser observado na Figura 4.22

```
BEGIN { FS = "-"
          printf  "\n%20s %2s %4s %1s %1s\n", "Endereços
IP","|", "Incidências","|", "Portas"
          print "===========|===========|==================="
}

        { printf  "%20s %2s %6s %6s %1s\n", $1, "|", $2, "|",
$3
          print "--------------|-----------|--------------" }


END  { print
"\n=========================================================" }
```

Figura 4.22: Arquivo responsável por dar forma à saída da função Firewall

Como pode ser observado na Figura 4.23, o arquivo `tmp_ep4awk` é lido pelo comando *`cat`* e, em seguida, o *`awk`* aplica o arquivo `formata.awk` (que contém os parâmetros para a formatação da saída do relatório) utilizando o sinal de menos (-) como delimitador dos campos que serão exibidos como foi demonstrado na Figura 4.18.

```
cat $CAMINHO/tmp_ep4awk|awk -f $CAMINHO/formata.awk >> $SAIDA
```

Figura 4.23: Aplicação do arquivo `formata.awk`

Antes de terminar, a função Firewall ainda registra tentativas de acesso via *ssh* (porta 22) que tenham falhado na autenticação da senha. O código pode ser visto na Figura 4.24.

O arquivo `tmp_ssh_failed`, recebe os registros do arquivo *`messages`* (localizado em */var/log/*), depois o *`sed`* é utilizado para pegar a informação que está entre os campos *from* e *port*, que é exatamente o endereço IP.

Devido a um espaço no início do arquivo `tmp_ssh_failed` gerado pela saída do `sed`, é feito um ajuste na contagem através de uma estrutura condicional *if* para que o total exibido seja compatível com os registros encontrados.

```
echo "Tentativas de acesso ilegal via SSH" >> $SAIDA
 echo "=========================================" >> $SAIDA
 echo >> $SAIDA
 cat /var/log/messages|grep -i sshd|grep -i failed| \
grep -i "$mes_dia"|sed 's/ /\n/g' > $CAMINHO/tmp_ssh_failed
 sed -n '/from/{/port/tc;:a;/port/!{N;ba;};: \
c;s/.*from//;s/port.*$//;p;}' $CAMINHO/tmp_ssh_failed|sort| \
uniq > $CAMINHO/tmp_ssh_failed_count
 cat $CAMINHO/tmp_ssh_failed_count >> $SAIDA
 echo >> $SAIDA
 qdt=`wc -l $CAMINHO/tmp_ssh_failed_count | cut -d " " -f1`
 if [ $qdt -ge 2 ]
   then
       qdt=`expr $qdt - 1`
 fi
 echo "$qdt endereços IP registrados." >> $SAIDA
 echo "-----------------------------------------" >> $SAIDA
```

Figura 4.24: Registro de tentativas de acesso via *ssh*

4.4.7 - Função Squid

Esta função tem duas atividades à realizar. A primeira é o registro de tentativas de acesso a *sites* bloqueados, informando o IP da estação que tentou o acesso. A segunda é o registro de acessos a *sites* que contenham determinadas palavras na sua *url*. Para esta segunda atividade, existe uma seção chamada Squid no arquivo `avisa.conf`, onde é possível colocar as palavras que serão localizadas.

O registro de datas do arquivo *`access.log`* não corresponde a um padrão normal de visualização como nos registros de *log* do *`iptables`*, por exemplo. Por isso, foi necessário utilizar uma função da linguagem *Perl* conhecida como *localtime*. Assim, mais um arquivo auxiliar foi construído, o `localtime.pl`.

Como na Função Firewall, a data será fornecida por uma função à parte denominada Data, que será explicada mais adiante. Por enquanto, basta saber que a variável mes_dia retorna a data corretamente.

Na Figura 4.25, pode ser observado que o primeiro passo é registrar todas as ocorrências dos endereços bloqueados no arquivo `tmp_squid`. Os comandos *`tr`* e *`cut`* foram utilizados, respectivamente, para eliminar espaços duplicados e especificar quais campos seriam capturados.

A próxima etapa é registrar todos os endereços IP das estações no arquivo `tmp_squidhosts` e em seguida é feito uma filtragem para relacionar os endereços aos *sites* bloqueados.

```
perl $CAMINHO/localtime.pl /var/log/squid/access.log|\
tr -s " "|cut -d " " -f2,3,5,7,8,11|grep -i DENIED|\
grep -i "$mes_dia"\ > $CAMINHO/tmp_squid
 cat $CAMINHO/tmp_squid | cut -d " " -f4 | sort -n |\
uniq > $CAMINHO/tmp_squidhosts
 for i in `cat $CAMINHO/tmp_squidhosts`
  do
     echo >> $SAIDA
     echo $i >> $SAIDA
     echo "================" >> $SAIDA
     cat $CAMINHO/tmp_squid | cut -d " " -f4,6 | grep -i $i |\
cut -d " " -f2|cut -d "/" -f1,2,3,4|sort -n |uniq >> $SAIDA
  done
```

Figura 4.25: Detalhes sobre a primeira tarefa da função Squid

Para realizar a pesquisa por palavras chaves, foram utilizados os recursos exibidos na Figura 4.26. Para cada palavra registrada no arquivo `avisa.conf`, é realizada uma pesquisa e o resultado vai para o relatório.

```
for palavrachave in `awk '$1 ~ /#Squid/,/#\/Squid/ { print }'\
$CAMINHO/avisa.conf |grep -iv '#'`
  do
   perl $CAMINHO/localtime.pl /var/log/squid/access.log |\
tr -s " " | cut -d " " -f2,3,4,5,7,8,11 | grep -v DENIED |\
grep -iw $palavrachave | grep -i "$mes_dia" >> $SAIDA
 done
```

Figura 4.26: Detalhes sobre a segunda tarefa da função Squid

4.4.8 - Função Data

A função Data foi criada para cuidar dos registros de datas feitos pelas funções Firewall e Squid. Sua tarefa é checar a data, fazer as adaptações necessárias e armazená-las na variável mes_dia. A princípio, a simplicidade de obter uma data pode encobrir problemas que só se manifestarão em determinadas situações.

Por exemplo, quando o *script* for executado uma vez por dia (no caso da execução por expediente) não existirão problemas com a data, mas o *script* deixa de registrar dados sobre aquele dia a partir do momento que o relatório for tirado e algum acesso acontecer após sua execução.

Para os servidores que trabalham 24 horas por dia, a melhor maneira de resolver isto é adiando a última execução do *script* para, por exemplo às zero horas e cinco minutos do dia seguinte. Desta forma, fica garantido que, de 0 hora do dia anterior até as 23 horas 59 minutos e 59 segundos, todas as ações das funções Firewall e Squid foram registradas.

O problema acontece justamente porque se não houver uma orientação naquele momento, o *script* fará o registro do dia atual, o que não é o resultado correto. A solução pode ser vista na Figura 4.27. Se o contador for menor do que o valor encontrado na sessão Contador ou igual a "1", ele irá registrar a data do dia corrente. Se o valor for diferente destas comparações, indica que esta é a última execução do dia, e apesar de estar acontecendo no dia seguinte, a data será de um dia atrás ao que se está sendo executado o *script*.

```
 if
[ $contador -lt `awk '$1 ~ /#Contador/,/#\/Contador/ \
{ print }' $CAMINHO/avisa.conf | grep -iv '#'` ] || \
[ $contador -eq 1 ]
 then
  dia=`date +%-d`
  mes=`date +%b`
  ano=`date +%Y`
 else
  dia=`date +%-d --date "1 day ago"`
  mes=`date +%b --date "1 day ago"`
  ano=`date +%Y --date "1 day ago"`
fi
```

Figura 4.27: Função Data

Para atender ao padrão de saída do *log* do *firewall* e do arquivo *access.log*, é feita a troca da abreviatura dos meses que tem diferença entre os idiomas português e inglês, caso contrário erros poderão acontecer durante a coleta dos dados no arquivo *messages*.

```
         case $mes in

        Fev)
                mes=Feb;;
        Abr)
                mes=Apr;;
        Mai)
                mes=May;;
        Ago)
                mes=Aug;;
        Set)
                mes=Sep;;
        Out)
                mes=Oct;;
        Dez)
                mes=Dec;;
        esac

mes_dia=`echo $mes $dia`
```

Figura 4.28: Troca das abreviaturas para atender aos registros das funções Firewall e Squid

4.4.9 - Função Envio

Nada mais que a cópia da saída para o destino. Sendo estas, duas variáveis que serão definidas na função Principal.

4.4.10 - Função Backup

A fim de evitar que o relatório se perca por não ter conseguido ser enviado ao destino devido a algum problema de conexão (isto porque, por padrão, o envio é feito por cópia remota para um servidor que armazena todos os relatórios dos servidores) esta função se encarrega de registrar o relatório em um arquivo denominado `backup_<data>.log`, onde `<data>` é o dia corrente.

Quando são completados oito dias de *backup*, a função passa a eliminar os arquivos mais antigos, um por dia, como pode ser visto na Figura 4.29

```
cp $SAIDA $CAMINHO/backup_$DATACERTA.log
howmany=`ls -lha backup_* |tr -s " " |cut -d " " -f8 |wc -l`
if [ $howmany -eq 8 ]
 then
 REMOVER=`ls -lhat $CAMINHO/backup_*|tail -n1|awk \
'{print $NF}'`
     rm -f $REMOVER
fi
```

Figura 4.29: Detalhes sobre a função Backup

4.4.11 - Função Principal

A função Principal é responsável por definir algumas variáveis importantes para a funcionalidade do *script*, preparar o cabeçalho do relatório, fazer as chamadas às outras funções descritas e apagar os arquivos temporários criados após a execução da ferramenta Avisa.

Como pode ser visto na Figura 4.30, nesta primeira parte, são definidas as variáveis caminho, saída, contador, datacerta e destino. A variável datacerta é necessária pelo mesmo motivo explicado anteriormente no tópico da função Data. A variável destino contém o endereço do servidor remoto para onde será despachado o arquivo relatório. Para fazer esta cópia, é necessário fazer uso do recurso de chaves públicas e privadas do serviço *OpenSSH*. Estas informações podem ser encontradas no anexo.

```
 #!/bin/bash

Principal () {
CAMINHO=/root/avisa
SAIDA=$CAMINHO/registro.log
contador=`cat $CAMINHO/contador`
if
[ $contador -lt `awk '$1 ~ /#Contador/,/#\/Contador/
{ print }' $CAMINHO/avisa.conf | grep -iv '#'` ] || \
[ $contador -eq 1 ]
 then
  DATACERTA=`date +%d%m%Y`
 else
  DATACERTA=`date +%d%m%Y --date "1 day ago"`
fi
DESTINO=gcomp.no-\
ip.info:/srv/www/default/html/servidores/gcomp/$DATACERTA.log
```

Figura 4.30: Primeira parte da função Principal

A segunda parte pode ser observada na Figura 4.31, onde é apresentada a programação para criar o arquivo relatório e seu cabeçalho. Como o padrão para os nomes dos servidores é o nome da empresa, utilizou-se do comando *uname* para capturar o nome do cliente.

```
echo -n "RELATÓRIO DE SERVIDORES DE COMPARTILHAMENTO DE \
ACESSO À INTERNET - " > $SAIDA
   echo $DATACERTA >> $SAIDA
   echo >> $SAIDA
   echo -n "Cliente: " >> $SAIDA
   uname -n | cut -d "." -f1 >> $SAIDA
   echo "============================" >> $SAIDA
   echo >> $SAIDA
```

Figura 4.31: Segunda parte da função Principal

A terceira parte é composta pela chamada às funções, como pode ser vista na Figura 4.32. Organizando o *script* desta forma, a fase de testes fica bem mais simples de ser executada, pois cada função é independente da outra,

podendo ser executada individualmente, o que ajuda na hora de encontrar erros durante a programação.

```
  HW
   Rede
   Sistema
   Smart
   Tripwire
   Firewall
   Squid
   Envio
   Backup
```

Figura 4.32: Terceira parte da função Principal

A última parte é a responsável por cuidar da remoção dos arquivos temporários e manter o arquivo `contador` trabalhando corretamente. Quando o valor do arquivo se iguala ao da seção no arquivo de configuração `avisa.conf`, ele recebe o valor "0" e em seguida é incrementado. Como foi explicado, este arquivo garante a consistência das informações do relatório da função Firewall.

```
rm -f $CAMINHO/tmp_* > /dev/null
if [ $contador -eq `awk '$1 ~ /#Contador/,/#\/Contador/ \
{ print }' $CAMINHO/avisa.conf|grep -iv '#'` ]
   then
        contador=0
fi
echo `expr $contador + 1` > $CAMINHO/contador
}
```

Figura 4.33: Quarta parte da função Principal

4.5 – Considerações finais

Após a finalização da construção e da organização de cada função, a estrutura deste trabalho ficou composta por um arquivo de configuração (`avisa.conf`), o *script* (`avisa.sh`), quatro arquivos auxiliares permanentes (`reincidentes`, `localtime.pl`, `formata.awk` e `contador`), além de um *IDS* de *hosts*

que foi adicionado (*Tripwire*) e a ferramenta *smartmontools*. Já os comandos utilizados fazem parte dos pacotes básicos da distribuição utilizada.

A utilização de vários arquivos temporários como visto durante a demonstração dos detalhes das funções é justificada pela facilidade para depurar problemas ocorridos durante o desenvolvimento. No processo de depuração, a utilização de `#!/bin/bash -x` no *script* faz com que as variáveis sejam resolvidas e os resultados exibidos na tela. Com estes procedimentos, foi possível diminuir o tempo de confecção desta parte da ferramenta. A Figura 4.34 mostra o resultado deste capítulo.

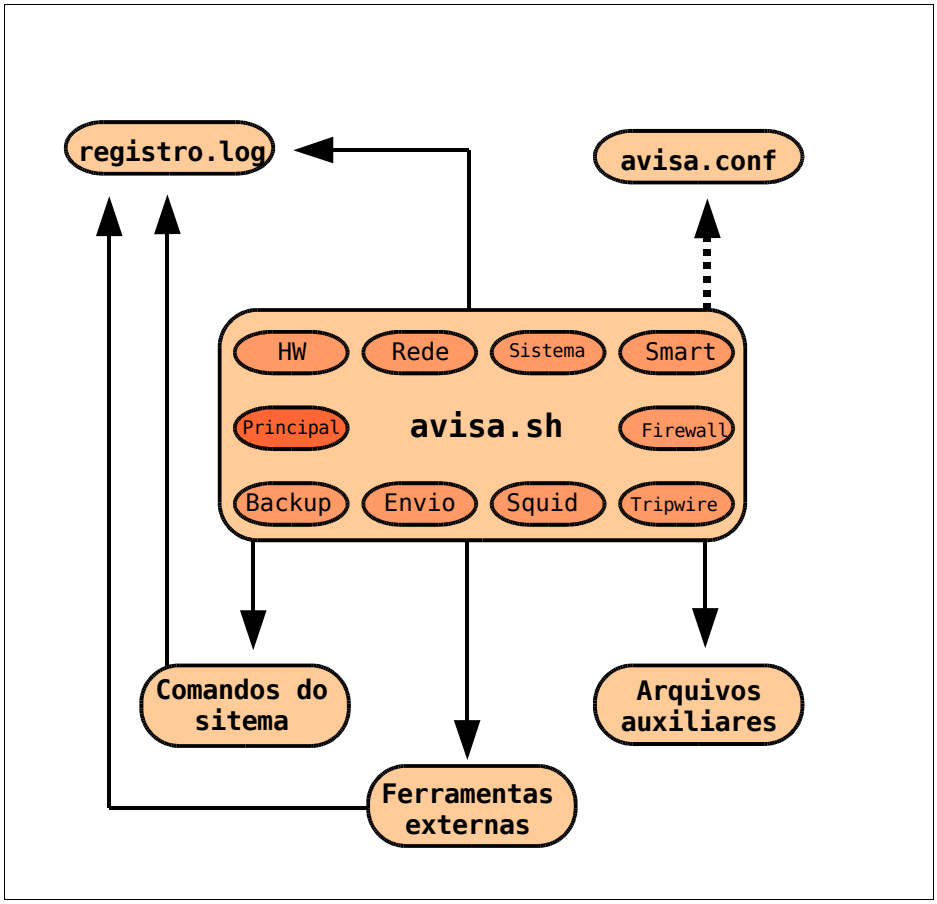


Figura 4.34: Representação da ferramenta Avisa

Capítulo 5 - Construção de um instalador.

5.1 - Considerações iniciais

Neste capítulo será abordada a criação de um instalador para a ferramenta Avisa. Como pôde ser observado no Capítulo 4, a estrutura se tornou complexa, envolvendo vários arquivos auxiliares, ajustes necessários no *script e* alterações em arquivos do sistema. Por isso, um instalador que pudesse auxiliar no processo de configuração e instalação foi cogitado. A apresentação do instalador segue o mesmo padrão da explicação do *script* `avisa.sh`, a descrição das partes relevantes através de figuras e explanação sobre os pontos mais complexos.

5.2 – Planejamento e implementação

Após a finalização da etapa de construção do *script* `avisa.sh`, alguns testes iniciais foram realizados em um servidor. A primeira impressão foi que existiam procedimentos demais no processo de instalação. Por isso, os testes foram interrompidos para que fosse estudada a possibilidade de se criar um instalador para a ferramenta.

O primeiro problema ocorre quando for necessário lembrar de todos os comandos que são pré-requisitos para o funcionamento do *script* e, ainda, verificar se eles estão instalados. O segundo problema é não ser suficiente copiar o arquivo de configuração `avisa.conf` e o *script* `avisa.sh` para um diretório e executá-lo. Faz-se necessário copiar os arquivos auxiliares, conferir se o conteúdo está de acordo com os requisitos e agendar sua execução através do *Crontab*.

Além destes problemas, dependendo do horário em que o *script* será instalado no servidor, o arquivo `contador` precisará ter seu valor alterado. Caso contrário, o *script* não registrará as informações referente àquele dia de forma correta.

Este projeto pode ser útil para outras pessoas que tenham dificuldades semelhantes a estas apresentadas no trabalho, logo ele deve ser produzido da forma mais clara e simples possível. Porém, tantos detalhes para serem

acertados pelo administrador antes de colocar a ferramenta para funcionar, pode confundir e desestimular sua utilização.

Disponibilizar uma ferramenta com tantos detalhes particulares não é viável. Todas estas questões preliminares dificultam a instalação nos vários servidores para os quais esta ferramenta pretende atender. Pensando nestes e em outros problemas que podem acontecer, nada melhor do que automatizar ao máximo o processo de instalação. Uma solução é criar um instalador que possa auxiliar esta tarefa.

Como existe um arquivo de configuração, ele concentrará todos os ajustes necessários para efetuar uma instalação personalizada. O arquivo receberá comentários que ajudem a orientar o administrador a escolher as opções que melhor lhe atenda.

O primeiro passo foi criar um arquivo chamado Leiame, que recebeu instruções de como deveria proceder a instalação. Este arquivo também contém uma cópia do *script* `avisa.sh` devidamente comentado.

A ferramenta Avisa é apresentada em um novo modelo composto pelo arquivo de instrução (Leiame) para orientar a instalação, um arquivo de configuração (`avisa.conf`) onde serão feitos os ajustes necessários para a instalação e um arquivo instalador (`instalador.sh`) responsável por checar as configurações no arquivo `avisa.conf`, gerar o arquivo `avisa.sh` e todos os outros arquivos auxiliares. Um exemplo pode ser visto na Figura 5.1.

```
[root@park avisa]# ls -lha
total 104K
drwxrwxr-x   2 root  root  4,0K 2001-01-15 21:21 .
drwx------  40 andin andin 8,0K 2001-01-15 21:44 ..
-rw-rw-r--   1 root  root   50K 2001-01-02 05:29 Leiame
-rw-r--r--   1 root  root  5,3K 2001-01-02 05:04 avisa.conf
-rwx------   1 root  root   28K 2001-01-01 22:39 instalador.sh
[root@park avisa]#
```

Figura 5.1: Arquivos que compõem a ferramenta Avisa

O arquivo `instalador.sh` tem suas atribuições demonstradas na Figura 5.2.

- checagem de pré-requisitos (conforme definido no arquivo `avisa.conf`)
- atribuição da variável CONTADORINST (conforme definido em `avisa.conf`)
- criação de um arquivo auxiliar 4rclocal.sh (necessário em servidores 24h/7dias)
- alteração do arquivo rc.local (necessário somente em servidores 24h/7dias)
- alteração do arquivo crontab
- criação do arquivo auxiliar reincidentes
- criação do arquivo auxiliar `contador`
- criação do arquivo auxiliar localtime.pl
- criação do arquivo auxiliar `formata.awk`
- atribuição da variável caminhoinst (conforme definido no arquivo `avisa.conf`)
- criação do arquivo `avisa.sh`
- alteração das permissões nos arquivos `instalador.sh` e `avisa.sh`

Figura 5.2: Atribuições do arquivo `instalador.sh`

A checagem de pré-requisitos é feita utilizando *awk*, acessando o arquivo `avisa.conf` da mesma forma que foi explicado no capítulo anterior. Uma seção Requisitos foi criada e cada comando utilizado foi inserido, para que a checagem possa dar prosseguimento a instalação. Como pode ser visto na Figura 5.3, foi utilizada uma estrutura *for* para armazenar cada entrada da seção Requisitos e em seguida foi utilizado o comando *test* para verificar se existe ou não o arquivo. Caso o arquivo não seja encontrado, o processo de instalação é abortado e retorna uma mensagem informando qual comando necessita ser instalado.

```
 echo "Checando dependências..."
 for needed in `awk '$1 ~ /#Requisitos/,/#\/Requisitos/ \
{ print }' avisa.conf | grep -iv '#'`
      do
       echo -n "Localizando $needed ..."
       test -s `whereis $needed | cut -d " " -f2`
       if [ $? == 0 ]
        then
         echo "... encontrado!"
        else
         echo "... não encontrado! A ferramenta não pode \
prosseguir sem $needed. Instale e tente novamente."
         exit
       fi
      done
```

Figura 5.3: Checagem de requisitos feita pelo arquivo `instalador.sh`

Como foi explicado no Capítulo 4, foram definidos dois tipos de configuração para a utilização do *script*. Na primeira, ele é configurado para ser executado apenas uma vez por dia (atendendo aos servidores que funcionam de acordo com o expediente do cliente). Na segunda, o *script* é executado quatro vezes por dia (atendendo aos servidores que funcionam 24horas por dia) e para isso a seção Contador do arquivo `avisa.conf` é definida com os seguintes valores "1" ou "4" respectivamente.

O arquivo `instalador.sh` checa a opção na seção Contador e armazena na variável CONTADORINST. Desta forma o instalador saberá como proceder dali em diante. Caso a variável registre o valor "1", o *script* fará as alterações no arquivo `/etc/crontab` conforme exibido na Figura 5.4.

```
cat /etc/crontab > tmp_crontab
echo "50 17 * * * root `pwd`/avisa.sh" >> tmp_crontab
cp tmp_crontab /etc/crontab
rm -f tmp_*
```

Figura 5.4: Alteração realizada no arquivo crontab

Caso a variável registre o valor "4", o *script* procederá como descrito na Figura 5.5. O arquivo *`crontab`* receberá quatro novas linhas, referentes às quatro solicitações de execução durante o dia. Como explicado no Capítulo 4, a última execução acontece a zero hora e cinco minutos do dia seguinte.

```
cat /etc/rc.local > tmp_rc.local
echo `pwd`/4rclocal.sh >> tmp_rc.local
cp tmp_rc.local /etc/rc.local
cat /etc/crontab > tmp_crontab
echo "00 08 * * * root `pwd`/avisa.sh" >> tmp_crontab
echo "00 12 * * * root `pwd`/avisa.sh" >> tmp_crontab
echo "00 18 * * * root `pwd`/avisa.sh" >> tmp_crontab
echo "05 00 * * * root `pwd`/avisa.sh" >> tmp_crontab
cp tmp_crontab /etc/crontab
rm -f tmp_*
```

Figura 5.5: Alteração realizada no arquivo crontab (segundo caso)

Como planejado, o arquivo `contador` estará valendo "1" durante a

primeira execução (às 08hs), “2” durante a segunda (ao meio dia), “3” durante a terceira (às 18hs) e “4” durante a última execução (às 0h e 05 minutos).

Quando o instalador é executado, ele deverá verificar que horas são naquele exato momento, para decidir com qual valor o arquivo `contador` será preenchido naquele primeiro momento. Por exemplo, caso a instalação seja após ao meio dia, o arquivo `contador` deverá ser criado com o valor “três” para que o arquivo `contador` inicie o dia seguinte com o valor correto, que deve ser “um”. Se isto não ocorrer, o *script* se confundirá na hora de tomar algumas decisões durante sua execução.

Outro problema também pode ocorrer, por exemplo uma queda de energia, o servidor é desligado às onze horas da manhã e só volta ao funcionamento às treze horas. Naquele exato momento, o contador deveria estar com o valor “3”, mas como a execução do horário de meio dia não ocorreu, o arquivo não foi incrementado (logo seu valor continua “2”) e conseqüentemente toda a ordem será comprometida.

Para resolver estes dois problemas, o arquivo `instalador.sh` cria um outro arquivo auxiliar, o arquivo `4rclocal.sh`. Ele tem a tarefa de garantir que o valor do arquivo `contador` sempre estará correto. Então caso o *script* seja instalado em um servidor que funcione vinte e quatro horas por dia, além de fazer as alterações no arquivo *`crontab`*, o `instalador.sh` criará o `4rclocal.sh`, que acertará o valor inicial do arquivo `contador` durante a instalação e criará uma entrada no arquivo *`rc.local`*, para que sempre que o servidor for reiniciado, ele corrija o valor do arquivo `contador` de acordo com o horário.

No próximo passo, o *script* criará os arquivos auxiliares `formata.awk`, `localtime.pl` e `reincidentes`. Para isso, foi utilizada a técnica de escrita com o comando *`cat`* como descrito no capítulo 3. O arquivo `instalador.sh` contém todos os arquivos auxiliares dentro de sua estrutura, criando-os conforme necessário.

Antes de criar o arquivo `avisa.sh`, o instalador checa a seção Destino e armazena o conteúdo. Desta forma, quando criar o arquivo `avisa.sh`, ele estará configurado com a variável destino corretamente.

A Figura 5.6 exibe a parte inicial do arquivo `avisa.sh` que será gerado pelo `instalador.sh`.

Neste ponto do arquivo `instalador.sh`, o comando *`cat`* é executado gravando sua saída em um arquivo chamado `avisa.sh` até que encontre a linha `EOF4`, o que indica que o *`cat`* deve encerrar seu trabalho.

Pode-se notar que para serem lidos como comuns, todos os caracteres especiais foram protegidos com a "\" tornando possível a criação do arquivo `avisa.sh` de forma correta. Em duas situações não ocorrem a proteção com o uso da "\" . O primeiro caso , é na linha `CAMINHO=`pwd``, que permite criar o arquivo `avisa.sh` com o *path* do local onde o `instalador.sh` está sendo executado. O segundo caso é na linha `DESTINO=$CAMINHOINST/\$DATACERTA.log`, onde a variável CAMINHOINST dará lugar ao endereço informado no arquivo `avisa.conf`.

```
cat <<EOF4>> avisa.sh
#!/bin/bash

Principal () {
CAMINHO=`pwd`
SAIDA=\$CAMINHO/registro.log
contador=\`cat \$CAMINHO/contador\`
if
[ \$contador -lt \`awk '\$1 ~ /#Contador/,/#\/Contador/ \
{ print }' $CAMINHO/avisa.conf | grep -iv '#'\` ] || \
[ \$contador -eq 1 ]
 then
  DATACERTA=\`date +%d%m%Y\`
 else
  DATACERTA=\`date +%d%m%Y --date "1 day ago"\`
fi
DESTINO=$CAMINHOINST/\$DATACERTA.log
```

Figura 5.6: Trecho responsável por criar o *script* `avisa.sh`

Uma observação a fazer são as contra-barras que não fazem parte do arquivo `instalador.sh` e estão localizadas no final da oitava e nona linha da Figura 5.6. Esta contra-barra significa apenas que as linhas nove e dez formam uma mesma linha no *script.*

5.3 - Considerações finais

A construção de um instalador foi benéfica ao projeto. Além de automatizar o procedimento de instalação, evitou que algum passo fosse esquecido pelo administrador durante esta etapa. A instalação se tornou rápida e prática, vários testes de instalação foram realizados e sua eficiência comprovada. Apesar de não haver comentários dentro do arquivo `instalador.sh`, o arquivo é de simples compreensão e um arquivo de instruções (Leiame) contém uma réplica do arquivo `instalador.sh` comentado. A Figura 5.7 traz uma representação de sua funcionalidade.

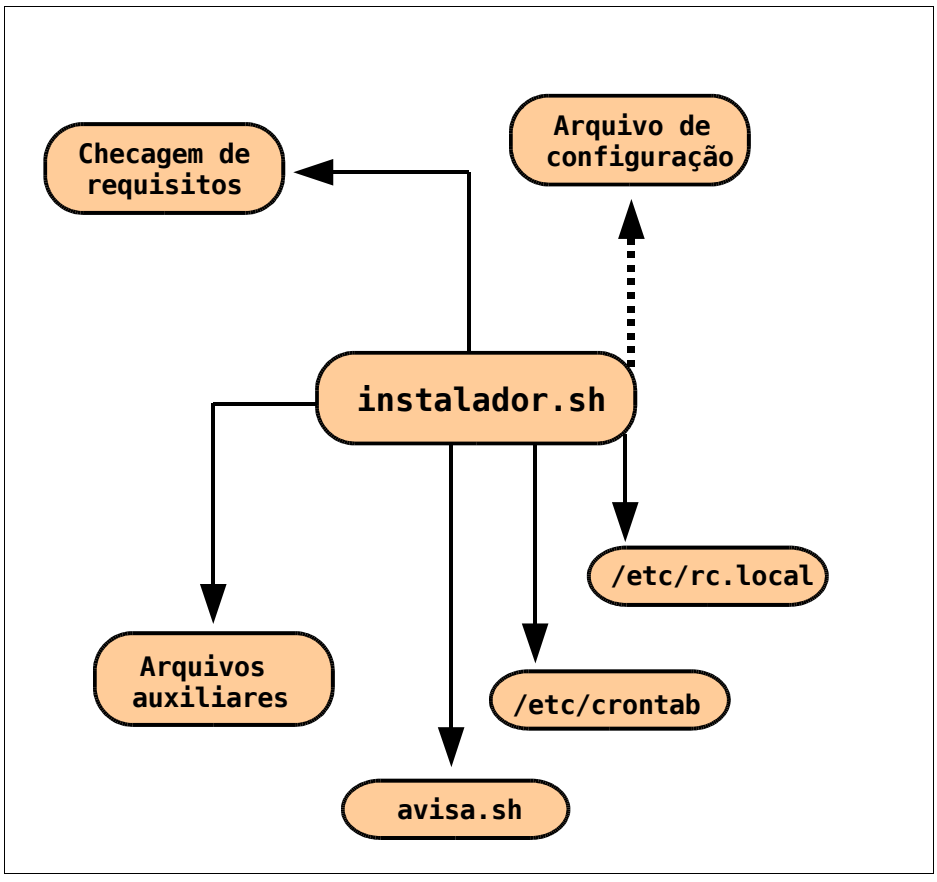


Figura 5.7: Representação do instalador para a ferramenta Avisa.

Capítulo 6 - Avaliações e Resultados

6.1 - Considerações iniciais

Este capítulo apresenta relatos da fase de testes que foram realizados após a conclusão do desenvolvimento da ferramenta Avisa. São comentadas as principais particularidades que influenciaram na etapa de desenvolvimento, além de descrições sobre os resultados obtidos com a ferramenta e as conclusões sobre a eficiência na utilização de cada função.

6.2 - Avaliações

A explicação apresentada nos Capítulos 4 e 5, são referentes à versão final e definitiva da ferramenta, porém desde a primeira versão, vários testes foram realizados e melhoramentos foram implementados até o *script* `avisa.sh` estar pronto.

Um importante ponto a se destacar, durante os testes no processo de desenvolvimento e que ajudou a organizar o trabalho que estava sendo feito, foi a possibilidade de criar as diferentes funções de forma totalmente independente no mesmo *script*. Este método permitiu comentar as chamadas às funções concluídas, evitando a necessidade de se trabalhar em arquivos separados. Isso tornava o ambiente de trabalho mais simples e otimizava o tempo na fase das avaliações.

Com o isolamento do código de cada função, ficou mais rápida a identificação de erros durante a programação. Apesar de ter sido usado o pacote *syntax* do editor VI que faz uso de cores para distinguir comandos e variáveis, problemas com o código aconteceram. Saber que o erro estava em uma área restrita onde o código estava sendo escrito, trouxe mais tranqüilidade no desenvolvimento e, principalmente, nos momentos que precisaram de depuração do código.

O processo de avaliação não apresentou complicações até a finalização da função Redes. Até este ponto, apenas os procedimentos para registrar as versões do *Iptables, Squid, OpenSSH* e *OpenSSL* precisaram de um pouco mais de atenção, pois seria interessante registrar também as versões dos pacotes. Para

cumprir esta tarefa, alguns ajustes nos comandos utilizados foram necessários principalmente no caso do *OpenSSL*.

Por se tratar apenas de redirecionamento da saída de comandos, o trabalho mais demorado, nesta fase, foi acertar a formatação do relatório criado pelo *script* `avisa.sh`. Vale ressaltar que o planejamento e o estudo do que cada comando iria fazer ajudaram a tornar as coisas mais fáceis nesta etapa. Para solucionar as tarefas do *script* `avisa.sh`, quase sempre era necessário utilizar vários comandos agrupados, como pode ser visualizado na Figura 6.1

```
uptime|tr -s " "|cut -d " " -f4|cut -d ":" -f2|tr -d ","
```

Figura 6.1: Exemplo da utilização de múltiplos comandos em uma mesma linha.

Durante testes na função Sistema, um detalhe importante que passou despercebido foi o trecho onde se registra o tempo de atividade do servidor. Como pode ser visto na Figura 6.1, este era o código das versões iniciais do *script* `avisa.sh`.

```
echo -n "Este servidor esta ativo a " >> registro.log
uptime | tr -s " " | cut -d " " -f3 >> registro.log
echo >> registro.log
```

Figura 6.2: Código inicial para registro de tempo de atividade do servidor.

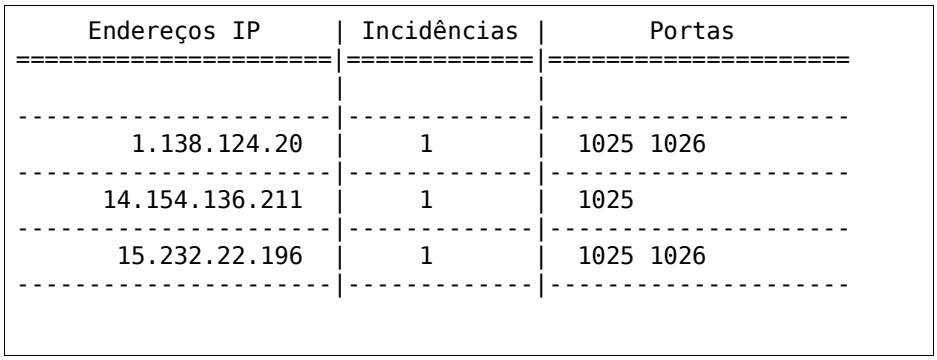
Quando foi realizado um teste com um servidor que estava a mais de um dia ativo, foi notório que este código não poderia permanecer no *script*, uma vez que ele não tratava esta situação e acabava deturpando a saída. Para resolver este problema, o código foi alterado para aquele apresentado na Figura 4.11 do capítulo 4.

A partir da Função Firewall, o processo de testes passou a ser mais trabalhoso. Um auxílio importante, nesta fase, foi a utilização de arquivos temporários para armazenar as saídas dos vários comandos utilizados para a construção desta função. Todo o código foi desenvolvido utilizando o editor Vi; assim, para depurar alguns erros, os arquivos temporários foram fundamentais.

Nas primeiras versões da função Firewall, o relatório era apenas a transposição dos bloqueios registrados nos *logs* para o arquivo de saída da ferramenta (`registro.log`). Acontece que os relatórios ficavam longos demais e

de difícil visualização.

Para gerar o relatório do *firewall* no formato exibido na Figura 6.3, muitos comandos foram envolvidos e vários arquivos temporários utilizados. O processo para separar endereços, portas e contabilizar as incidências diárias, necessitou de muita pesquisa, principalmente, sobre os comandos *sed* e *awk*. Conseqüentemente muitas tentativas foram realizadas até que a saída estivesse de acordo com o desejado.



Endereços IP	Incidências	Portas
1.138.124.20	1	1025 1026
14.154.136.211	1	1025
15.232.22.196	1	1025 1026

Figura 6.3: Saída gerada pela função Firewall

Para manter a base das incidências neste relatório, foi preciso planejar exatamente como seria este processo. Mesmo com todo o planejamento a necessidade de usar um arquivo que permitisse ao *script* decidir se ele deveria ou não registrar aquela nova ocorrência na base de endereços reincidentes, só foi observada quando o *script* já estava em produção.

Segundo a proposta, o endereço deveria ser registrado apenas uma vez por dia na tabela de reincidentes. Realizando um teste onde a ferramenta Avisa havia sido instalada a três dias, o maior indicativo de incidências só poderia ser três. Porém foi observado que haviam endereços com cinco registros. O que estava ocorrendo é a ferramenta não ser orientada a decidir se naquele momento ela deveria ou não incrementar o índice daquele endereço, que estava registrado no arquivo `reincidentes`. Assim que foi descoberta esta falha, ela foi corrigida e a criação do arquivo auxiliar `contador` resolveu o problema.

Desta forma a criação dos arquivos auxiliares surgiram a partir da fase de avaliação e foram incluídos de acordo com o desenvolvimento da ferramenta. Após a criação da primeira versão funcional, uma nova sessão de

ajustes e testes aconteceu. A própria fase de testes foi lapidando o *script* `avisa.sh`. As versões foram testadas e melhoradas até que os principais problemas tivessem sido resolvidos.

Mesmo finalizando esta etapa, a ferramenta poderá apresentar deficiências e conseqüentemente necessitar de ajustes, mas a ferramenta está estável para ser usada nos servidores em produção. Correções ou novas funcões podem ser registradas para uma nova versão.

6.3 - Resultados

O primeiro procedimento antes de ativar a ferramenta é estabelecer a conexão com o servidor onde ela será instalada. Para facilitar o contato permanente com os servidores instalados nos clientes que usam conexão *adsl*, o uso de um serviço de *DNS* dinâmico é interessante. Também é necessário configurar o *OpenSSH* para trabalhar com chaves públicas e privadas para que as cópias não solicitem senhas no processo de transmissão do arquivo.

Apesar de, neste ponto do projeto, a ferramenta já ter capacidade de alertar sobre a falta de componentes essenciais para seu funcionamento, é válido lembrar que as ferramentas *Tripware* e *Smart* deverão ser instaladas caso suas funções queiram ser utilizadas. Estas funções são responsáveis pelo registro de alteração no sistema de arquivos e integridade do disco rígido, respectivamente. Estes procedimentos necessários antes de se instalar a ferramenta Avisa, estão detalhados no anexo.

Seguindo os procedimentos descritos no arquivo Leiame, as chances de erro diminuem, isto porque o primeiro problema que poderia ocorrer ao executar o arquivo `instalador.sh`, é a ausência de algum pré-requisito para a utilização da ferramenta. Se isto acontecer, uma mensagem como a apresentada na Figura 6.4 irá aparecer. É importante frisar que existe uma explicação dentro do arquivo `avisa.conf`, informando que, para desativar algum pré-requisito, deve-se comentar a linha na seção Requisitos.

```
Checando dependências...
Localizando awk ...... encontrado!
Localizando cat ...... encontrado!
Localizando df ...... encontrado!
Localizando free ...... encontrado!
Localizando grep ...... encontrado!
Localizando history ...... encontrado!
Localizando ifconfig ...... encontrado!
Localizando iptables ...... encontrado!
Localizando last ...... encontrado!
Localizando lspci ...... encontrado!
Localizando perl ...... encontrado!
Localizando ping ...... encontrado!
Localizando ps ...... encontrado!
Localizando route ...... encontrado!
Localizando smartctl ...... encontrado!
Localizando squid ...... encontrado!
Localizando uname ...... encontrado!
Localizando Tripwire ...... não encontrado! A ferramenta não
pode prosseguir sem Tripwire. Instale e tente novamente.
```

Figura 6.4: Verificação dos pré-requisitos pelo *script* `instalador.sh`

Nenhum outro inconveniente deverá acontecer durante a instalação da ferramenta e, ao término de sua execução será emitido um aviso, como pode ser visto na Figura 6.5

```
Atenção!
O valor do arquivo contador neste momento é 1.

Este arquivo é muito importante para que as informações do
registro de reincidencias dos ips bloqueados pelo firewall
sejam precisas. Então, caso execute o script avisa.sh
manualmente para testes, lembre-se de verificar o valor deste
arquivo antes da execução, para posteriormente voltá-lo a seu
valor original. Lembre-se que sempre que o script é executado,
ele altera seu valor.
```

Figura 6.5: Aviso gerado pelo *script* `instalador.sh`

Em todos os servidores que a ferramenta foi instalada, nenhum problema ocorreu durante a instalação. Foram feitas instalações em diversos horários do dia, em todos os casos o valor do arquivo `contador` foi checado e seu conteúdo estava como o esperado. Simulações de desligamento e

reinicializações também foram realizadas e o arquivo `4rclocal.sh` respondeu como deveria quando sua utilização foi solicitada. O sistema de cópia de segurança foi feito e rotacionado como planejado, um exemplo do resultado pode ser visto na Figura 6.6.

```
-rw-r--r--   1 root root  37K 2005-11-07 00:08 backup_06112005.log
-rw-r--r--   1 root root  44K 2005-11-08 00:09 backup_07112005.log
-rw-r--r--   1 root root  42K 2005-11-09 00:09 backup_08112005.log
-rw-r--r--   1 root root  50K 2005-11-10 00:10 backup_09112005.log
-rw-r--r--   1 root root 102K 2005-11-11 00:11 backup_10112005.log
-rw-r--r--   1 root root  44K 2005-11-12 00:10 backup_11112005.log
-rw-r--r--   1 root root  34K 2005-11-12 18:04 backup_12112005.log
```

Figura 6.6: Arquivos de backup

Com o acompanhamento constante dos relatórios gerados, algumas características dos servidores acabam se tornando familiares. Este é um dos objetivos desejados neste trabalho, conhecer os servidores de uma forma bem particular. Observar a árvore de processos, as conexões e os serviços de rede ativos, acompanhar as reinicializações do sistema, observar a estrutura de diretórios, entre tantos outros dados disponíveis, é uma fonte muito rica de informações. No início, os olhos não estão acostumados, ler o relatório de cada servidor pode parecer uma tarefa cansativa e enfadonha. Mas com o tempo as informações passam a ficar mais claras, principalmente as alterações.

6.3.1 - HW

A parte mais estática do relatório sem dúvida é a função HW, só ocorrerão mudanças se algum componente físico do servidor for modificado. Esta parte do relatório funciona como um registro sobre alguma manutenção ocorrida no servidor que não tenha sido comunicada, por exemplo. O resultado do trabalho da função HW está na Figura 6.7

```
##### HARDWARE ###############################################

Informações sobre o processador:

processor       : 0
vendor_id       : AuthenticAMD
cpu family      : 5
model           : 8
model name      : AMD-K6(tm) 3D processor
stepping        : 0
cpu MHz         : 266.329
cache size      : 64 KB
--------------------------------

Modelo do HD Primary Master: ST34321A
--------------------------------

Quantidade de memória instalada: 27940 KB
----------------------------------

Informações sobre modelo da(s) placa(s) de rede:
0000:00:09.0 Ethernet: Realtek RTL-8139/8139C/8139C+ (rev 10)
0000:00:0a.0 Ethernet: VIA Tech. VT6105 [Rhine-III] (rev 85)
-------------------------------------------------------------

Informações sobre modelo da placa de video:
0000:00:13.0 VGA compatible controller: Silicon Integrated
Systems [SiS] 5597/5598/6326 VGA (rev 65)
-------------------------------------------------------
```

Figura 6.7: Resultado emitida pela função HW

6.3.2 - Sistema

A função Sistema é a que gera o maior número de informações sobre diferentes tópicos no relatório. Na Figura 6.8 pode-se observar a parte com as informações sobre versões do *Kernel* e dos serviços instalados no servidor.

```
##### SISTEMA ###############################################

Este servidor está instalado com a distribuição Conectiva
Linux 10

Versão do Kernel: 2.6.5-63077cl
------------------------------------------------------------

Versão do Iptables: v1.2.11
Referente ao pacote iptables-1.2.11-72578U10_2cl
------------------------------------------------------------

Versão do Squid: 2.5.STABLE9
Referente ao pacote squid-2.5.5-77559U10_12cl
------------------------------------------------------------

Versão do SSH: OpenSSH_3.8.1p1
Referente ao pacote openssh-server-3.8.1p1-60281cl
------------------------------------------------------------

Versão do OpenSSL: openssl0.9.7
openssl
Referente ao pacote openssl0.9.7-0.9.7c-52922cl
openssl-progs-0.9.7c-52922cl
------------------------------------------------------------
```

Figura 6.8: Primeira parte do relatório gerado pela função Sistema

O controle dos acessos realizados e os registros das reinicializações também são úteis para saber qual é o hábito dos responsáveis por desligar o servidor e cadastrar *sites* na lista de liberados ou bloqueados gerenciada pelo *Squid*. Estas informações aliadas com os registros da função *Tripware* ajudam a provar, por exemplo, se aconteceu um acesso indevido, caso alguma coisa seja desconfigurada no servidor. Na Figura 6.9, pode ser vista esta parte do relatório.

```
Este servidor foi reiniciado pela ultima vez em: Tue Nov 8
12:20
------------------------------------------------------------


Este servidor esta ativo a 4 dia(s), 11 hora(s) e 44 minuto(s)
-------------------------------------------------


Informações sobre os três últimos logins:

ssh  pts/3  200.243.66.27   Sat Nov 12 20:13   still logged in
ssh  pts/2  201.19.50.209   Tue Nov  8 14:50 - 15:11  (00:20)
reboot system boot 2.6.5-63077cl Tue Nov  8 12:20  (4+11:44)
------------------------------------------------------------
```

Figura 6.9: Segunda parte do relatório gerado pela função Sistema

Além destas informações, também são exibidos quadros com os processos e os módulos ativos, incluindo o total de cada um deles. Estas informações podem parecer desnecessárias, mas ajudam a encontrar, por exemplo, um padrão para a variação dos processos que nascem e morrem nos servidores. Isto pode ser comprovado na Figura 6.10. Com uma pesquisa rápida no diretório onde ficam armazenados os relatórios, pode-se observar que existe uma média bem definida no total de processos ativos neste servidor. Isto pode ajudar a detectar facilmente alguma anomalia, caso este padrão mude abruptamente.

```
[root@Linux-ibor]# find -exec grep -i "processos ativos" {} \;

Registrados 40 processos ativos.
Registrados 42 processos ativos.
Registrados 40 processos ativos.
Registrados 40 processos ativos.
Registrados 40 processos ativos.
Registrados 40 processos ativos.
Registrados 41 processos ativos.
Registrados 43 processos ativos.
Registrados 40 processos ativos.
Registrados 40 processos ativos.
Registrados 40 processos ativos.
Registrados 40 processos ativos.
Registrados 40 processos ativos.
Registrados 40 processos ativos.
Registrados 45 processos ativos.
Registrados 40 processos ativos.
Registrados 40 processos ativos.
Registrados 40 processos ativos.
Registrados 40 processos ativos.
Registrados 41 processos ativos.
Registrados 40 processos ativos.
Registrados 42 processos ativos.
Registrados 40 processos ativos.
Registrados 40 processos ativos.
Registrados 42 processos ativos.
```

Figura 6.10: Utilizando o *find* para pesquisar entre os relatórios

6.3.3 - Rede

A função Rede é bem simples, seu relatório é composto por uma consulta com os comandos *arp*, *netstat* e a execução do comando *ifconfig* nas interfaces selecionadas no arquivo `avisa.conf`. O acompanhamento destas informações ajuda a verificar problemas nas interfaces de rede, variações excessivas no tráfego e detalhes sobre as conexões ativas e seus processos correspondentes. Cruzando informações com a função Sistema, pode-se observar informações sobre os processos de serviços de rede inicializados no sistema.

6.3.4 - Smart

A função Smart é responsável por emitir um relatório sobre a integridade do disco rígido, porém como pacote *smartmontools* só consegue ler as informações em discos a partir do tipo *ATA-3*, o recomendado neste caso é fazer um teste, como descrito no anexo, após a instalação do pacote. Caso o disco não seja suportado, a chamada à função Smart deve ser comentada no arquivo `avisa.conf` para evitar que o relatório emita informações erradas sobre o disco rígido. No caso do disco suportar a utilização, informações como demonstrado na Figura 6.11 irão aparecer no relatório.

```
      Relatório de integridade do disco rígido:

========================================

HDA

SMART overall-health self-assessment test result: PASSED

SMART Self-test log structure revision number 1

Num Test Status Remaining LifeTime(hours) LBA_of_first_error
21  Short offline Completed without error 00%  6570    -

-------------------------------------------------------
```

Figura 6.11: Relatório da Função Smart

Testes com um disco rígido danificado ainda não foram realizados para avalizar a eficiência da utilização da ferramenta *smartmontools*. Desta forma não se pode garantir que as informações do relatório sejam totalmente confiáveis. Nos servidores que esta função foi habilitada, os registros foram sempre de sucesso nos testes. Como mostrado na Figura 6.11, o primeiro teste é uma avaliação geral do estado do disco e uma falha pode significar que o disco não está mais confiável e pode parar a qualquer momento. O segundo é um teste físico que é realizado em aproximadamente dois minutos, sem a possibilidade de corrompimento dos dados enquanto é executado.

6.3.5 - Tripware

A função Tripwire emite resultados bem completos, a parte exibida na Figura 6.12 por exemplo, é um quadro geral das alteração no sistema de arquivo. Além deste quadro geral, o relatório exibe cada categoria separadamente informando quais arquivos sofreram modificações.

```
---------------------------------------------------------------------------

  Rule Name                     Severity Level    Added    Removed  Modified
------                        --------------    -----    -------  --------
  Invariant Directories           66                0        0        0
  Temporary directories           33                0        0        0
  Tripwire Data Files             100               0        0        0
  Critical devices                100               0        0        0
* User binaries                   66                1        0        1
  Tripwire Binaries               100               0        0        0
* Libraries                       66                2        0        1
  File System and Disk Administraton Programs
                                  100               0        0        0
  Kernel Administration Programs  100               0        0        0
  Networking Programs             100               0        0        0
  System Administration Programs  100               0        0        0
  Hardware and Device Control Programs
                                  100               0        0        0
  System Information Programs     100               0        0        0
  Application Information Programs
                                  100               0        0        0
  (/sbin/genksyms)
  Shell Related Programs          100               0        0        0
  Operating System Utilities      100               0        0        0
  Critical Utility Sym-Links      100               0        0        0
  Critical system boot files      100               0        0        0
* System boot changes             100               16       0        35
* OS executables and libraries    100               2        0        1
* Critical configuration files    100               0        0        2
  Security Control                100               0        0        0
  Login Scripts                   100               0        0        0
  Shell Binaries                  100               0        0        0
* Root config files               100               25       7        9

Total objects scanned:  5607
Total violations found:  102
```

Figura 6.12: Quadro Geral fornecido pela função Tripware

O único trabalho gerado por esta função é fazer a atualização da base do *Tripware.* Assim o quadro mostrado na Figura 6.12 ficará com os registros todos limpos novamente, facilitando a identificação de alterações. Esta atualização obviamente não deve ser agendada, já que cabe ao administrador decidir se as alterações são seguras. Os procedimentos se encontram no anexo.

6.3.6 - Firewall

As informações emitidas pela função Firewall são compostas por duas partes, a primeira é um quadro informativo mostrando os endereços bloqueados, as portas referentes e o número de reincidências diárias daquele endereço. A segunda é um registro de tentativas de acesso remoto via *ssh* (porta 22) que falharam durante o fornecimento da senha. As informações são interessantes justamente por proporcionar uma série de dados para avaliação. Um exemplo pode ser visto na Figura 6.13, onde é mostrado um número excessivo de bloqueios partindo da mesma origem e contra as mesmas portas em dois servidores diferentes.

```
Servidor acotel
----------------------|-------------|---------------------
      221.211.255.14  |    24       |  1026 1027
----------------------|-------------|---------------------

Servidor gcomp
----------------------|-------------|---------------------
      221.211.255.14  |    21       |  1026 1027
----------------------|-------------|---------------------
```

Figura 6.13: Bloqueios registrados pela função Firewall

Após uma pesquisa na internet, pôde-se verificar que estas portas são freqüentemente utilizadas para explorar uma falha no serviço de mensagens de sistemas *Microsoft*, que acaba permitindo o recebimento de *spam* por estas portas. A própria *Microsoft* alerta para este fato como pode ser observado em (STOPSPAM, 2004). Outro *site* alerta para este problema, e permite fazer um teste para verificar se seu sistema está vulnerável. Pode ser acessado em (WATCHMAN, 2004) . Para comprovar as informações, um teste foi realizado em uma estação instalada com o sistema operacional Windows 2000, protegida pelo servidor Linux, os bloqueios são semelhantes como apresenta a Figura 6.14.

```
Nov 13 13:25:52 servidor-acotel kernel: INPUT-Bloqueado:
IN=ppp0 OUT= MAC= SRC=66.110.201.18 DST=201.19.37.74 LEN=794
TOS=0x00 PREC=0x00 TTL=114 ID=59811 PROTO=UDP SPT=6910
DPT=1026 LEN=774

Nov 13 13:25:52 servidor-acotel kernel: INPUT-Bloqueado:
IN=ppp0 OUT= MAC= SRC=66.110.201.18 DST=201.19.37.74 LEN=794
TOS=0x00 PREC=0x00 TTL=114 ID=59812 PROTO=UDP SPT=14192
DPT=1027 LEN=774

Nov 13 13:25:52 servidor-acotel kernel: INPUT-Bloqueado:
IN=ppp0 OUT= MAC= SRC=66.110.201.18 DST=201.19.37.74 LEN=794
TOS=0x00 PREC=0x00 TTL=114 ID=59813 PROTO=UDP SPT=9574
DPT=1028 LEN=774

------------------------------------------------------------

Nov 13 10:27:20 servidor-acotel kernel: INPUT-Bloqueado:
IN=ppp0 OUT= MAC= SRC=221.211.255.14 DST=201.19.37.74 LEN=502
TOS=0x00 PREC=0x00 TTL=46 ID=0 DF PROTO=UDP SPT=50181 DPT=1026
LEN=482

Nov 13 10:27:20 servidor-acotel kernel: INPUT-Bloqueado:
IN=ppp0 OUT= MAC= SRC=221.211.255.14 DST=201.19.37.74 LEN=502
TOS=0x00 PREC=0x00 TTL=44 ID=0 DF PROTO=UDP SPT=50181 DPT=1027
LEN=482
```

Figura 6.14: Comparação entre bloqueios registrados pela função Firewall

Os três primeiros registros são provenientes do teste que foi realizado. Os outros dois são os últimos registros dos bloqueios registrados como mostrado na Figura 6.13. É perceptível a semelhança entre eles e o que provavelmente está ocorrendo, é o endereço 221.211.255.14, que pertence a faixa de domínio concedido a *China Network Communications Group Corporation*, estar contaminado com algum tipo de *spammer*. Foi realizado um contato com os responsáveis através dos *emails* contidos nas informações de registro de domínios, mas nenhuma resposta retornou.

Desta forma, pode-se perceber que o relatório, apesar de despretensioso, pode ser fonte para pesquisas e descobertas interessantes. Deixando claro que não se pode compará-lo a um relatório fornecido por um *IDS*, que fornece estatísticas muito refinadas e detalhadas.

A partir de algumas ocorrências registradas no relatório, como pode ser

visto na Figura 6.15, foi verificado que por algumas vezes ocorreram tentativas de invasão no servidor.

```
Tentativas de acesso ilegal via SSH
============================================================

192.168.0.1
200.88.118.10

2 endereços IP registrados.
------------------------------------------------------------
```

Figura 6.15: Registro de tentativas de acesso ilegal via ssh

Utilizando uma técnica chamada *Brute Force*, o atacante tenta ganhar acesso através de uma ferramenta que utiliza um dicionário de palavras para tentar adivinhar a senha. Cruzando as informações da Figura 6.15 com o registro de *log* do servidor (Fig. 6.16), pode-se confirmar a tentativa de invasão.

```
...
Oct 18 20:37:08 LinuxServer sshd[8304]: Illegal user nate from 200.88.118.10
Oct 18 20:37:09 LinuxServer sshd[8304]: Failed password for illegal user
nate from 200.88.118.10 port 32772 ssh2
Oct 18 20:37:12 LinuxServer sshd[8306]: Illegal user will from 200.88.118.10
Oct 18 20:37:12 LinuxServer sshd[8306]: Failed password for illegal user
will from 200.88.118.10 port 32915 ssh2
Oct 18 20:37:17 LinuxServer sshd[8310]: Illegal user natal from
200.88.118.10
Oct 18 20:37:17 LinuxServer sshd[8310]: Failed password for illegal user
natal from 200.88.118.10 port 33052 ssh2
Oct 18 20:37:21 LinuxServer sshd[8312]: Illegal user start from
200.88.118.10
Oct 18 20:37:21 LinuxServer sshd[8312]: Failed password for illegal user
start from 200.88.118.10 port 33257 ssh2
Oct 18 20:37:25 LinuxServer sshd[8314]: Illegal user verwaltung from
200.88.118.10
Oct 18 20:37:25 LinuxServer sshd[8314]: Failed password for illegal user
verwal from 200.88.118.10 port 33399 ssh2
Oct 18 20:37:28 LinuxServer sshd[8316]: Illegal user tomhao from
200.88.118.10
Oct 18 20:37:28 LinuxServer sshd[8316]: Failed password for illegal user
tomhao from 200.88.118.10 port 33557 ssh2
Oct 18 20:37:32 LinuxServer sshd[8318]: Illegal user tatiane from
200.88.118.10
Oct 18 20:37:32 LinuxServer sshd[8318]: Failed password for illegal user
tatiane from 200.88.118.10 port 33721 ssh2
...
```

Figura 6.16: Registro fornecido pelo sistema *syslog*

Com o acontecimento de outras ocorrências como esta descrita, procurou-se novas formas de proteger o servidor de ataques como este. Duas medidas para evitar ataques por *Brute Force* foram tomadas. A primeira foi a restrição ao acesso via ssh à faixa de endereços IP mais comum no país, a segunda foi a utilização do atributo *limit* do *iptables* sugerida em (NETO,2004) que bloqueia o acesso por um tempo determinado após o número de tentativas permitidas.

6.3.7 - Squid

Os relatórios desta função são um informativo interessante e revelam dados que podem refletir o perfil do usuário e, principalmente, fazer uma relação com problemas nas estações.

Existem clientes que preferem não bloquear o acesso e, assim, acabam prejudicando a própria rede. Em um destes clientes que não faz restrição ao

acesso, foi realizado um teste. Conforme descrito no capítulo 4, uma das atribuições da função Squid é justamente capturar as saídas que tenham uma palavra chave na *url* do endereço acessado e registrá-las no relatório. Esta palavra chave é inserida no arquivo `avisa.conf` e o *script* `avisa.sh` se encarrega de gerar os registros. Na Figura 6.17 pode-se observar um pequeno trecho de um relatório que faz o registro com a palavra "sexo".

```
Oct 24 14:30:23 2005 192.168.1.197  http://sexo.vtudo.com.br/vip/
Oct 24 14:30:23 2005 192.168.1.197  http://sexo.vtudo.com.br/vip/sabrina/
Oct 24 14:35:44 2005 192.168.1.197  http://sexo.vtudo.com.br/famosas
Oct 24 14:35:45 2005 192.168.1.197  http://sexo.vtudo.com.br/famosas/
Oct 24 14:35:45 2005 192.168.1.197  http://sexo.vtudo.com.br/exit.htm
Oct 24 14:35:45 2005 192.168.1.197  http://sexo.vtudo.com.br/famosas/css.css
Oct 24 14:35:46 2005 192.168.1.197  http://sexo.vtudo.com.br/ttt-out.php
Oct 24 14:35:49 2005 192.168.1.197  http://sexo.vtudo.com.br/exit.htm
Oct 24 14:38:17 2005 192.168.1.197  http://sexo.vtudo.com.br/ttt-out.php
Oct 24 16:16:49 2005 192.168.1.197  http://sexo.uol.com.br/sexoadulto/
Oct 24 17:18:42 2005 192.168.1.197  http://amazonsex.com/sexo/
Oct 24 17:19:50 2005 192.168.1.197  http://amazonsex.com/sexo/join.php
Oct 24 17:26:19 2005 192.168.1.197  http://www.allinternal.com/sexo
Oct 24 17:27:21 2005 192.168.1.197  http://sexo.katatudo.com.br/
Oct 24 17:32:42 2005 192.168.1.197  http://ct.lp.uol.com.br/sexo/index.js?
```

Figura 6.17: Uso de palavra chave pela função Squid

Se esta estação estivesse usando o sistema operacional Windows, certamente estaria avariada, pois o sistema é extremamente vulnerável a vírus e o navegador Internet Explorer contém inúmeras falhas de segurança. Como o sistema utilizado nesta estação é o Linux, ele acaba auxiliando em mantê-la funcional apesar dos hábitos do usuário.

Outro registro realizado é sobre tentativas de acesso a *sites* bloqueados. Este registro pode revelar usuários problemáticos e ajudar a identificar focos de vírus, *spywares* e outras pragas que podem atacar uma rede. Um usuário que sempre aparece nesta lista, certamente comprometerá seu equipamento na primeira oportunidade. A Figura 6.18 exibe os bloqueios realizados em uma rede onde só é liberado acesso a alguns *sites*.

```
192.168.0.7
================
http://runonce.msn.com
http://update.messenger.yahoo.com/msgrcli6.html
http://www.terra.com.br
http://www.incredibarvuz1.com/incrediappserver.dll

192.168.0.8
================
http://www.yahoo.com.br
http://gateway.messenger.hotmail.com/gateway

192.168.0.12
================
http://ar.atwola.com/image
http://cb.icq.com/cb
http://ui.skype.com/ui

192.168.0.15
================
http://www.terra.com.br/thegirl
http://www.uol.com.br/sexo
```

Figura 6.18: Registro fornecido pela função Squid

6.4 - Considerações finais

O planejamento no processo de desenvolvimento refletiu uma fase de testes onde não foi necessário fazer grandes alterações no código escrito. Os resultados revelaram que, apesar da utilização quase exclusiva de comandos simples, o relatório traz informações relevantes para o acompanhamento dos servidores. Como pôde ser observado, as informações contidas no relatório final gerado pela ferramenta proporcionam desde o conhecimento de alterações no *hardware* do servidor até descobertas sobre atuações de *spammers* e tentativas de invasão por técnica de força bruta contra o serviço de conexão remota *ssh.*

A ferramenta, atualmente, funciona em dez empresas. Em uma transportadora que controla sua frota de caminhões através de um sistema

desenvolvido para *web*, a ferramenta Avisa forneceu dados sobre uma excessiva taxa de tentativas de acesso à porta 80, bloqueadas pelo *firewall.* Com estes dados, foi sugerida a troca do servidor *web IIS*, por um servidor com *Apache*, a criação de uma zona desmilitarizada e a troca do endereço IP do *link.*

Em uma empresa do setor atacadista foi possível prever um problema no *logrotate,* que não estava funcionando corretamente. Os relatórios apresentavam um crescimento anormal no espaço utilizado do disco-rígido, fato que alertou para o problema do sistema de rotação dos registros de *log*.

Relatórios foram enviados para as empresas que se mostraram muito satisfeitas com o trabalho realizado, inclusive agendando atualizações para seus servidores. Os relatórios ajudaram a fechar novos negócios, mostrando que a empresa prestadora do serviço está preocupada com o pós-venda.

Pôde-se notar que a satisfação e a credibilidade aumentaram após a aplicação da ferramenta Avisa nos servidores dos clientes.

Capítulo 7 - Conclusão

Este foi um projeto desenvolvido em várias etapas, a definição do problema, a pesquisa por alternativas para solução, a implementação, a realização dos testes e a implantação. O resultado foi uma solução simples, mas capaz de atender todas as exigências do problema. O motivo para ter alcançado o resultado desejado foi ter o problema bem definido. Abordar um problema que não se sabe exatamente onde começa e onde termina, aumenta chance de não alcançar bons resultados. Como o assunto escolhido para este trabalho de conclusão de curso partiu de um problema vivido no dia-a-dia do autor, não foi difícil detalhá-lo, visto que suas características eram bem conhecidas. Um grande aprendizado neste trabalho é aprimorar a capacidade de descrever problemas, relacionar seus pontos chaves e, principalmente, dimensioná-lo. Desta forma dificilmente, a solução não será alcançada por mais complexo que um problema possa parecer.

A linguagem *Shell* se comportou de forma adequada, oferecendo todos os recursos necessários para a implementação da ferramenta. Apesar de ter sido feito um esforço para desenvolver a ferramenta da forma mais simples possível, a falta de prática não permitiu que partes do código ficassem mais refinadas. Como existem vários comandos com finalidades similares, principalmente quando se trata da manipulação e filtragem em arquivos texto, extrair a informação desejada não é o problema, pois, mesmo trilhando caminhos distintos, os comando acabam convergindo em um mesmo resultado. O diferencial é por conta da demora na execução, o que no caso desta ferramenta não é exatamente um problema, uma vez que trabalha de forma agendada. Pode-se concluir que em outras situações como por exemplo ferramentas interativas, o código deve ser mais trabalhado, ou alternativas como a linguagem Perl e C podem ser utilizadas.

O conhecimento dos vários comandos utilizados neste trabalho foi fundamental para que o resultado fosse alcançado. Praticamente todo o relatório é gerado por comandos que fazem parte dos discos de instalação da distribuição. Apenas duas ferramentas externas foram utilizadas, o *IDS* de *host Tripwire* e a ferramenta para verificação de integridade de disco-rígido, *Smart*. A possibilidade de trabalhar manipulando as saídas de comandos tanto para arquivo quanto para outros comandos permitiu que o relatório fosse construído sem maiores dificuldade.

Apesar de haver um *firewall* configurado nestes servidores e uma pequena estatística de bloqueios fornecida pela ferramenta desenvolvida, não foi objetivo deste trabalho abordar o tema segurança. A criação do relatório de *firewall* foi motivada apenas pela possibilidade de registrar informações que podem ser fontes de pesquisa para assunto de um trabalho futuro. A curiosidade é fundamental para motivar novas descobertas, a função Firewall trabalha com esta característica. Na verdade, ela não procura exatamente por alguma coisa, apenas observa e registra os bloqueios ocorridos nas *chains input* e *forward* do *firewall*. Ainda que sem maiores pretensões, os relatórios desta função exibiram dados que acabaram ajudando a modificar alguns procedimentos de segurança nestes servidores (Relatado no capítulo 6).

Como foi explicado, o problema estava relacionado com falta de informações sobre servidores de compartilhamento de acesso a internet, depois que estes eram instalados nos clientes. Na verdade, este problema foi inconscientemente estimulado, pois a equipe que implantava e instalava os servidores não pensou nas conseqüências antes de oferecer o serviço.

O acompanhamento de mais de vinte servidores é uma tarefa complicada, principalmente por exigir muito tempo nas checagens. Os relatórios emitidos pela ferramenta Avisa, no pouco tempo de uso tem ajudado bastante nesta tarefa. O hábito de ler os registros acabam deixando a leitura mais simples e os olhos parecem mais treinados para encontrar detalhes. Aos poucos, a ferramenta esta sendo ativada nos servidores dos clientes e o acompanhamento sendo restabelecido.

O contato intenso com a linguagem *Shell* durante este trabalho também ajudou a consolidar vários conceitos e técnicas aprendidas durante o curso, que agora, após uma aplicação prática, estão muito mais consolidadas. Apesar de ter plena consciência das limitações que ainda possui, o autor se sente satisfeito com o progresso que obteve neste período em que foi aluno do curso de pós-graduação em Administração em Redes Linux.

Capítulo 8 - Trabalhos futuros

A solução implementada conseguiu suprir as necessidades do problema proposto, apesar de necessitar de alguns ajustes para se tornar mais aplicável como por exemplo, a adaptação automática para outras distribuições que não sejam baseadas em Red Hat.

Como demonstrado no Capítulo 6, os relatórios contém dados que podem ser cruzados ou contabilizados gerando novas informações sobre os servidores. Para melhorar este tipo de consulta, a criação de um outro script capaz de utilizar os relatórios para criar estatísticas sobre os servidores seria muito interessante. Outra alternativa é criar um sistema capaz de ler os relatórios que chegam dos servidores dos clientes, direcionar estas informações para um banco de dados e desenvolver uma aplicação em *PHP* (por exmplor) capaz de fazer pesquisas mais refinadas através da base de dados.

A criação de um módulo capaz de analisar os relatórios (registrados no servidor de destino) e emitir alertas para telefones celulares é um planejamento para versões futuras. O administrador poderá estabelecer padrões para estes alertas, como por exemplo, alterações nas médias de processos ativos, ativações de módulos ou espaço em disco.

Para facilitar o processo de configuração da ferramenta Avisa, pretende-se criar um sistema interativo através do utilitário *dialog*, que é o responsável por, por exemplo, as configurações realizadas com o aplicativo *ntsysv*, onde pode-se habilitar serviços na inicialização do sistema.

Este projeto deixou margem para muitas alterações que poderão melhorar a ferramenta e torná-la ainda mais completa e simples de manipular.

Referências Bibliográficas

ALLEN, B. Smartmontools Project. Disponível na Internet em: http://smartmontools.sourceforge.net/ - Acessado em 29 de Outubro de 2005.

BARBOSA, L. S. Realizando relatório do servidor. Disponível na Internet em: http://www.vivaolinux.com.br/scripts/verScript.php?codigo=973# - Acessado em 17 de novembro de 2005.

CAMARGO, H. A. Automação de Tarefas . Lavras: UFLA/FAEPE, 2005. (Curso de Pós-Graduação "*Lato Sensu"* (Especialização) a Distância em Administração em Redes Linux).

GNU. *Introduction to BASH*. Disponível na Internet em: http://www.gnu.org/software/bash/bash.html - Acessado em 24 de Setembro de 2005

JARGAS, A. M. Expressões Regulares – Guia de Consulta Rápida. São Paulo: Novatec, 2001

KANDASAMY, M. SystemInfo. Disponível na Internet em: http://freshmeat.net/projects/systeminfo/ - Acessado em 17 de novembro de 2005.

LOVE, R. Desenvolvimento do Kernel do Linux. Rio de Janeiro: Editora Ciência Moderna Ltda., 2004

MELO, S.;TRIGO, C. H. Projeto de Segurança em Software Livre. Rio de Janeiro: ALTABOOKS, 2004

NETO, U. Dominando Linux Firewall Iptables. Rio de Janeiro: Editora Ciência Moderna Ltda., 2004

NEVES, J. C. Programação Shell Linux 3ª Edição, Rio de Janeiro: Brasport, 2003

RESENDE, A. M. P. Monografia . Lavras: UFLA/FAEPE, 2002. (Curso de Pós-Graduação "*Lato Sensu"* (Especialização) a Distância em Administração em Redes Linux).

SICA, F. C,; UCHÔA, J. Q. Administração de Sistemas Linux Lavras: UFLA/FAEPE, 2005. (Curso de Pós-Graduação "*Lato Sensu"* (Especialização) a Distância em Administração em Redes Linux).

STOPSPAM. Disabling Messenger Service in Windows XP.
Disponível na internet em:
http://www.microsoft.com/windowsxp/using/security/learnmore/stopspam.mspx
Acessado em 23 de Novembro de 2005

THOBIAS. Lista `sed`-BR. Disponível na Internet em: http://thobias.org/sed/
Acessado em 07 de novembro de 2005.

TLDP. The Linux Documentation Project – I/O Introduction. Disponível na internet em: http://www.tldp.org/LDP/abs/html/ioredirintro.html – Acessado em 26 de setembro de 2005

WATCHMAN. myNetWatchman Alert - Windows PopUP SPAM. Disponível na Internet em:
http://www.mynetwatchman.com/kb/security/articles/popupspam/
Acessado em 23 de Novembro de 2005

ANEXO

ANEXO

Preparar Conexão SSH sem senha

No servidor do cliente, executar : `[prompt]# ssh-keygen -b 1024 -t rsa`

Quando for solicitado uma frase password, apenas confirmar com enter. Este comando irá gerar uma chave pública e uma privada. Copiar a chave publica para `/root/` do servidor destino. Em seguida, no servidor destino, executar: `[prompt#]  cat id-rsa.pub >> /root/.shh/authorized_keys`

Seguindo estes passos, o servidor do cliente poderá copiar arquivos e fazer acesso *ssh* sem confirmação de senhas.

Instalar o Smartmontools e testar compatibilidade com disco rígido.

Para instalar, executar: [prompt]# **apt-get install smartmontools**
Para testar á compatibilidade com o disco rígido:

1 - **smartctl –s on /dev/hdX**
2 - **smartctl –a /dev/hd**X
3 - **smartctl –i /dev/hdX**

Após estes passos, o seguinte quadro será mostrado:

Device Model: IC35L120AVV207-0
Serial Number: VNVD02G4G3R72G
Firmware Version: V24OA63A
Device is: In smartctl database [for details use: -P show]
ATA Version is: 6
ATA Standard is: ATA/ATAPI-6 T13 1410D revision 3a
SMART support is: Available - device has SMART capability.
SMART support is: Enabled

Observar as linhas em negrito, elas indicaram se o disco é ou não compatível com a ferramenta.

Preparar o Squid para bloquear os sites indesejados

Editar o arquivo squid.conf
Procurar por ACLS
Inserir a seguinte acl abaixo de: `acl all src 0.0.0.0/0.0.0.0`

```
acl bloqueados url_regex "/etc/squid/bloqueados"
```

onde, bloqueados é o nome do arquivo contendo os sites indesejados
Inserir a linha abaixo, antes da entrada que libera o acesso a rede interna

http_access deny bloqueados

Firewall

Adicionadar as regras para logar DROPS

iptables -A INPUT -j LOG --log-prefix "INPUT-Bloqueado: "

Logar conexoes ssh

iptables -A INPUT -p tcp --dport 22 -j LOG --log-prefix "ConexãoSSH: "

Tripwire

Instalar o pacote tripwire-2.3.1-17.i386.rpm
(www.wesmo.com/rpm2html/contributed/RPMS/tripwire-2.3.1-17.i386.html)

Executar:

```
twinstall.sh localizado dentro de /etc/tripwire/
tripwire --init 2> log
cat log | grep -i filename | cut -d " " -f3 > remover
```

A instalação padrão indica a checagem de vários arquivos que não existem no sistema Conectiva. Para revomer estas entradas basta criar e executar um *script* como descrito a seguir:

```
for saifora in `cat remover`
    do
       cat twpol.txt | grep -iv $saifora > temp
       mv temp twpol.txt
    done
```

Após executar o *script*, rodar twinstall.sh novamente para que a base que checa os arquivos seja alterada de acordo com o novo arquivo twpol.txt.