Universidade Federal do Rio de Janeiro

Escola Politécnica

Departamento de Eletrônica e de Computação

LibAntispam – Biblioteca Antispam de Propósito Geral

Autor:

Rafael Jorge Csura Szendrodi

Orientador:

Prof. Jorge Lopes de Souza Leão, Dr. Ing.

Examinador:

Prof. Antônio Cláudio Gómez de Sousa, M. Sc.

Examinador:

Prof. Aloysio de Castro Pinto Pedroza, Dr.

DEL

Maio de 2009

DEDICATÓRIA

Dedico este trabalho:
À Zeus (Jupiter), deus do Céu e da Terra, pai e rei dos deuses e dos homens, senhor do Olímpio e deus supremo deste universo.
À Hera (Juno), rainha dos deuses, protetora da vida, das mulheres, da fecundidade e do matrimônio.
À Athena (Miverva), deusa da sabedoria, do oficio, da inteligência e da guerra justa. Protetora do povo de Atenas.
À Ártemis (Diana), deusa da caça, da natureza, da colheita, da serena luz da lua, dos nascimentos e protetora das Amazonas.
À Afrodite (Venus), deusa da beleza e do amor, mãe de Enéias, fundador da raça romana, e matriarca da dinastia Julia (a dinastia de Julio Cesar).
À minha mãe, Ildi e ao meu pai Gyorgy, pelo meu nascimento e por, de certa forma, terem contribuído para que eu me moldasse no que sou hoje.

AGRADECIMENTO

Ao povo brasileiro que contribuiu de forma significativa à minha formação e estada nesta Universidade. Este projeto é uma pequena forma de retribuir o investimento e confiança em mim depositados.

Ao professor Leão, meu orientador neste projeto, por ter aceitado me guiar nesta minha jornada final do meu curso.

Aos professores Baruqui (meu orientador acadêmico), Joarez, Gabriel, Petraglia e Mariane, meus amigos há vários anos que sempre me incentivaram a não desistir do curso de eletrônica.

Aos demais professores do DEL, alguns já aposentados, pelos anos de dedicação à cátedra e a paciência que tiveram com seus alunos. Em especial: Osvaldo, Ricardo Rhomberg, Antonio Cláudio, Frederico (Fico), Lizarralde, Gelson, Aloysio, Mauros, Luis Wagner, Brafman, Casé, Otto, Rezende e Villas-Boas.

Aos professores do PEE (Programa de Engenharia Elétrica da COPPE/UFRJ). Em especial: Glauco, Antonio Carlos, Aredes, Rolim, Ramon, Pedreira, Pedroso, Tony, Eduardo, Lima Neto, Watanabe e os demais que não me vem à memória agora.

Aos meus amigos usuários do PADS e da COPPE, do passado e do presente, pelos anos de muitas alegrias que tive com vocês. Agradeço-lhes também por servirem de *beta-testers* para a LibAntispam durante todos estes anos.

Aos meus amigos ex-colegas e ex-administradores de redes do DEL, que agora já estão em novas carreiras: Poppe, Pappires, Edulima, Durval, Adriano, Josemar, Jonny, João Rocha (Piu-Piu), Solimar (Floyd), Daniel Mendes, Danilo Melges, Roberto Aziz, Azambuja e os outros que agora não estou me lembrando.

Aos queridos funcionários do DEL. Do LEG: Luis (in memoriam), Jorge (Jorgão) e Marcio (“Luis”). Da secretaria: Conceição, Eliomar, Luizinho e Fernando.

Aos queridos funcionários e colegas do PEE: Wilson (in memoriam), Arthur, Calvet, Daniele (Dani), Daniele Castro (Dani2), Márcia, Maurício, Rosa, Telma e Wanderley. E ainda as ex-secretárias do PEE, Solange e Bia.

Agradecimentos especiais às secretarias do PEE, Rosa e Daniele, pela reserva do auditório do PEE (sala de defesa de teses – H-322), onde pude defender o meu Projeto Final de maneira mais cômoda para mim e fazer as minhas demonstrações práticas, via rede sem-fio do PADS, do meu Projeto Final.

RESUMO

O objetivo deste trabalho foi o desenvolvimento de uma biblioteca (library) em linguagem C/C++ em que foram implementadas as regras antispam mais comumentemente usadas para a filtragem de *mensagens não-solicitadas* (conhecidas como *SPAMs*). Tais regras são implementadas em chamadas de funções que permitem que os *Agentes de Transporte de Mensagens* (sigla *MTA* em inglês), escritos em código-aberto C, mais utilizados (Sendmail, MeTA1, Exim e Postfix), modificados através de *patches*, executem tais regras, que são controladas por um grupo de arquivos de configuração padrão. Foi ainda desenvolvida uma *interface gráfica de usuário* (sigla GUI em inglês) para acesso com navegadores *WEB* e que permite que as regras antispam possam ser alteradas a nível global de sistema para todos os usuários ou até a um nível particularizado usuário por usuário.

Palavras-Chave: antispam, spam, MTA, mensagem não-solicitada, biblioteca.

ABSTRACT

The objective of this work was the development of a library write in C/C++ language that implements the most used antispam rules to filter *unsolicited messages* (most know as *SPAMs*). These rules are implemented in function calls that permits that the *Mail Transport Agents* (*MTA*), wrote in C open-source, most used (Sendmail, MeTA1, Exim e Postfix), modified using *patches*, use these rules, that are controlled by a group of default configuration files. It was developed a *graphical user interface* (GUI) for access using *WEB* browsers and permits that the antispam rules can be changed at global system level to all users or at user only level.

Key-words: antispam, spam, MTA, unsolicited message, library.

SIGLAS

ARPAnet – *Advanced Research Projects Agency Network*

ASCII – *American Standard Code for Information Interchange*

BBS – *Bulletin Board System*

BIND – *Berkeley Internet Name Domain*

BUG – *Termo técnico em inglês usado para denominar erros de programação*

CGI – *Common Gateway Interface*

CIDR – *Classless Inter-Domain Routing*

COPPE – *Instituto Alberto Luiz Coimbra de Pós-Graduação (formalmente, Coordenação dos Programas de Pós-Graduação em Engenharia)*

CPU – *Central Processing Unit (Unidade Central de Processamento)*

CTSS – *Compatible Time-Sharing System*

DDoS – *Distributed Denial-of-Service Attack (Ataque Distribuido por Negação de Serviço)*

DEL – *Departamento de Eletrônica e Computação*

DIALUP – *Designação para acesso à Internet por telefone*

DNS – *Domain Name Service*

DNSBL – *Domain Name Service Black List*

FQDN – *Fully Qualified Domain Name*

GID – *Group ID*

GUI – *Graphic User Interface*

HD – *Hard Drive*

HTTP – *HyperText Transfer Protocol*

HTTPS – *HyperText Transfer Protocol Secure*

ISC – *Internet Systems Consortium*

ISP – *Internet Service Provider*

LPS – *Laboratório de Processamento de Sinais*

MAPS – *Mail Abuse Prevention System*

MB – *Mega Byte*

MIT – *Massachusetts Institute of Technology*

MTA – *Mail Transport Agent (Agente de Transporte de Mensagens)*

MUA – *Mail User Agent (Agente de Mensagens de Usuário)*

MUD – *Multi-User Dungeons*

MX – *Mail Exchange*

PADS – *Laboratório de Processamento Analógico e Digital de Sinais*

PERL – *Pratical Extraction and Report Language*

POLI – *Escola Politécnica*

PTR – *Pointer (usado nas tabelas de DNS reverso)*

RAM – *Random Access Memory*

RDSSC – *Remote Domain SMTP Sender Checking*

RBL – *Realtime Black List*

SMTP – *Simple Mail Transfer Protocol*

SPAM – *Unsolicited Mail Message (Mensagem Não-Solicitada)*

SSH – *Secure Shell*

TCP/IP – *Traffic Control Protocol / Internet Protocol*

UFRJ – *Universidade Federal do Rio de Janeiro*

WEB – *Mesmo que WWW*

WWW – *World Wide Web*

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 – Tema

O tema do trabalho é o desenvolvimento de uma biblioteca (*library*) em linguagem C/C++ em que se implementem as regras antispam mais comumentemente usadas para a filtragem de *mensagens não-solicitadas* (conhecidas com *SPAMs* [1]). Tais regras são implementadas em chamadas de funções que permitem que os *Agentes de Transporte de Mensagens ou Mails Transport Agents* (*MTA*) em código-aberto C mais utilizados (Sendmail, MeTA1, Exim e Postfix), modificados através de *patches*, executem tais regras que são controladas por um grupo de arquivos de configuração padrão. Foi ainda desenvolvida uma interface gráfica GUI para acesso com *browsers* e que permite que as regras antispam possam ser alteradas a nível global de sistema para todos os usuários ou até a um nível particularizado usuário por usuário.

1.2 – Delimitação

O objeto de estudo é a implementação e padronização das regras antispam mais comuns, de forma que elas tenham o mesmo comportamento independentemente do *Agente de Transporte de Mensagem ou Mail Transport Agent* (MTA) usado, possam ser configuradas em arquivos de configuração padronizados e possam ser facilmente intercambiáveis pelos administradores de redes entre sistemas heterogêneos entre si.

1.3 – Justificativa

Os efeitos danosos da pratica do SPAM são bastante conhecidos hoje em dia: *Aumento do trafego TCP/IP desnecessário nos backbones*, *aumento da carga de trabalho dos roteadores e dos servidores de mensagens*, *aumento do custo de armazenamento das mensagens dos usuários* e *aumento do custo homem-hora no ambiente de trabalho*, que é gasto na verificação e remoção das mensagens indesejadas,

isso sem contar ainda as perdas quando o processo de remoção não provocar a remoção acidental de mensagens legitimas.

Existem três diferentes níveis técnicos ou abordagens técnicas para se combater o SPAM: A primeira é a filtragem durante o processo de verificação do envelopamento da mensagem (antes de receber o conteúdo da mesma), a segunda é a filtragem pelo conteúdo da mensagem (que é verificado ainda no MTA) e a terceira é a filtragem à nível de usuário (quando a mensagem já está depositada na *mailbox* do usuário), sendo que esses dois últimos níveis podem utilizar *Filtros Bayesianos* [2] com grande grau de sucesso ainda nos dias de hoje.

Neste trabalho, *a LibAntispam atuará no primeiro nível técnico de combate ao SPAM*, que ocorre durante o processo de verificação do envelopamento da mensagem. Umas das maiores dificuldades encontradas atualmente neste nível técnico de combate ao SPAM está na não-uniformidade dos arquivos e das regras de configuração devido à heterogeneidades dos MTAs existentes, sendo que muitas regras antispam se confundem freqüentemente com as regras normais de configuração de alguns MTAs. Outra grande dificuldade encontrada ocorre quando se necessita particularizar as configurações para que elas atuem ou não para usuários específicos.

1.4 – Objetivos

O objetivo é o desenvolvimento de uma biblioteca antispam de propósito geral. Isto visa: (1) Padronizar o funcionamento das regras antispam mais comumentemente utilizadas em servidores SMTP (MTA); (2) Facilitar o intercambio das regras antispam entre administradores de sistema heterogêneos, pois se desvinculará as regras antispam das configurações normais dos diferentes MTA; (3) Facilitar a aplicação das regras antispam tanto em nível global para todos os usuários do sistema como facilitar a particularização das regras antispam usuário por usuário.

1.5 – Metodologia

Este trabalho se valeu de um estudo das especificações mais recentes do protocolo SMTP, descritas na RFC-5321 [3], que trata do processo de envelopamento das mensagens eletrônicas e se valeu ainda do estudo do funcionamento interno dos

servidores SMTP (MTA) de código-aberto em C mais comumentemente usados, que são, nesta ordem, Sendmail [4], MeTA1 [5], Exim [6] e Postfix [7].

O estudo e a implementação das regras antispam teve que ter um tratamento especial, pois de todas as regras antispam comumentemente mais usadas atualmente, apenas a regra SPF (*Sender Policy Framework*) [8] possui uma especificação oficial no IETF (*Internet Engineering Task Force*) [9], as demais regras ou são especificas de um ou mais MTAs em particular ou então foram implementadas como módulos separados, feitas por programadores diferentes, sem nenhuma relação direta entre si, e em muitos casos usando outras linguagens de programação, como PERL, por exemplo. Ou seja, para a maioria das regras antispam não existe uma especificação formal para as mesmas, pois seus autores não tiveram originalmente esta intenção, com exceção dos autores do SPF. Com isto teve que ser feito um trabalho em campo na *WEB* para catalogar as regras, verificar seu funcionamento mais comum para, finalmente, se fazer a implementação das mesmas dentro da LibAntispam. Logo, além da implementação das regras antispam, se teve que fazer um trabalho de especificação e padronização das mesmas.

Para cada regra antispam levantada, foi feita a especificação e a implementação da mesma dentro da LibAntispam, sendo feitos os testes de funcionamento das mesmas e a implementação da chamada da mesma dentro de um MTA padrão, no caso o MeTA1, para verificar seu correto funcionamento em tempo real. Após isto, foi feita a implementação da respectiva chamada da regra para os outros MTAs (Sendmail, Exim e Postfix) procurando, quando foi o caso, adaptar a implementação devido a alguma necessidade de um dos MTAs usados sem, contudo, prejudicar a padronização e sem criar exceções de compilação dentro da LibAntispam.

Concomitantemente, foi desenvolvida a interface GUI que é acessada via browser e, após a implementação de uma nova regra, a mesma foi adiciona as opções de regras disponíveis ao administrador e ao usuário do sistema.

1.6 – Descrição

No capítulo 2 será falado sobre as origens do SPAM e suas técnicas mais usadas, incluindo também uma discussão sobre a fragilidade do protocolo SMTP e uma diferenciação necessária que deve ser feita entre *SPAM* (envio de mensagens não-solicitadas) e *Phishing* (fraude eletrônica).

O capítulo 3 apresenta as técnicas antispam de 1º nível, discorrendo sobre os três níveis de técnicas antispam existentes, as vantagens e desvantagens de cada uma delas, uma defesa para a utilização de regras de 1º nível de combate ao SPAM e ainda serão mostradas as técnicas antispam de 1º nível mais utilizadas atualmente.

Os detalhes da LibAntispam são apresentados no capítulo 4. Nele serão explicitadas as especificações e os pormenores das implementações das funções principais destinadas ao interfaceamento e comunicação com os MTAs.

O capítulo 5 detalha as funcionalidades da LibAntispam-GUI e demonstra sucintamente o seu funcionamento.

Uma avaliação de desempenho da LibAntispam é apresentada no capítulo 6, onde é feita uma comparação, durante o período de 1 mês, de três contas reais de usuários, sendo uma com nível de proteção baixo (sem regras antispam habilitadas), uma como nível de proteção médio (com apenas as regras antispam menos agressivas habilitadas) e uma com nível de proteção alto (com todas as regras antispam disponíveis habilitadas).

No capítulo 7 são feitas as conclusões finais deste trabalho.

Capítulo 2

O SPAM, suas origens e técnicas

2.1 – O que é o termo SPAM?

O termo SPAM se refere à prática do envio em massa de mensagens eletrônicas não-solicitadas. Esse termo foi tomado emprestado de um condimentado de carne suína enlatada fabricada pela empresa alimentícia americana *Hormel Foods Corporations* e que era um dos poucos gêneros alimentícios que não estavam sujeitos às restrições de racionamento impostas na Inglaterra durante a Segunda Guerra Mundial. O motivo de tal termo ter sido usado para designar a prática do SPAM se deve a um quadro humorístico, protagonizado pelo grupo humorístico inglês *Monty Python* [10], satirizando o referido racionamento de comida. Neste quadro, *vikings* dentro de um restaurante repetiam sem parar o termo "SPAM", de forma que qualquer tipo de conversa dentro daquele ambiente ficava totalmente inviável. Mais tarde, por analogia, o termo passou a associar o SPAM como uma prática que prejudicava a comunicação por mensagens eletrônicas (E-mail).

2.2 – O protocolo SMTP e sua fragilidade

O protocolo SMTP (*Simple Mail Transfer Protocol* – RFC-5321) é o protocolo para envio de mensagens eletrônicas *defacto* utilizado na Internet. Ele foi elaborado por Jonathan B. Postel da *University of Southern California* em agosto de 1982, a partir do sistema de correios eletrônicos que rodavam na ARPAnet [11], a precursora da Internet atual, e descrito na RFC-821 [12]. Ele foi revisado mais tarde por J. Klensin da *AT&T Laboratories*, em abril de 2001, e descrito na RFC-2821 [13] e novamente em outubro de 2008, sendo descrito na RFC-5321, porém sempre mantendo a essência básica do que foi escrito na RFC-821.

O protocolo SMTP é um protocolo de entrega (*delivery*) bastante simples e que se divide em 5 fases, que são: *Conexão, Envelopamento, Mensagem, Finalização e Desconexão*. A seguir, se poderá ver um exemplo de sessão cliente-servidor que é realizado utilizando-se o protocolo SMTP. As fases são identificadas por caracteres entre parênteses junto à identificação de Servidor (S) ou Cliente (C).

Exemplo de sessão SMTP cliente-servidor [1] [2]:

C(c): telnet mail.pads.ufrj.br 25
C(c): Connected to mail.pads.ufrj.br.
C(c): Escape character is '^]'.
S(c): **220 heres.pads.ufrj.br ESMTP Sendmail 8.14.3/8.14.3; Sat, 31 Jan 2009 21:22:37 -0200**
C(c): helo santuario.pads.ufrj.br
S(c): **250 heres.pads.ufrj.br Hello santuario.pads.ufrj.br [146.164.48.5], pleased to meet you**
C(e): mail from: <szendro@pads.ufrj.br>
S(e): **250 2.1.0 <szendro@pads.ufrj.br>... Sender ok**
C(e): rcpt to:<szendro@pee.ufrj.br>
S(e): **250 2.1.5 <szendro@pee.ufrj.br>... Recipient ok**
C(m): data
S(m): **354 Enter mail, end with "." on a line by itself**
C(m): From: “Rafael Jorge Csura Szendrodi” <szendro@pads.ufrj.br>
C(m): To: “Rafael Jorge Csura Szendrodi” <szendro@pee.ufrj.br>
C(m): Subject: Teste
C(m):
C(m): Teste
C(m):
C(m): []’s, Rafael
C(f): .
S(f): **250 2.0.0 n0VNMbRm007376 Message accepted for delivery**
C(d): quit
S(d): **221 2.0.0 heres.pads.ufrj.br closing connection**
C(d): Connection closed by foreign host.

Notas: [1] *Como foi utilizado o aplicativo “telnet” do UNIX, não está visível que os comandos terminam com os caracteres de controle CR e FL e as linhas do corpo da mensagem também terminam por CR e FL. A finalização da mensagem é identificada quando o servidor recebe a seqüência <CRLF>.<CRLF>. O próprio aplicativo “telnet” já se encarrega de acrescentar esses caracteres de controle às linhas que o usuário digita durante a comunicação cliente-servidor.*

[2] *A padronização do conteúdo passado após o comando DATA, i.e. cabeçalhos e corpo da mensagem, é ditada pela RFC-5322 [14] (originalmente ditada pela RFC-822 [15], sendo revisada em abril de 2001 pela RFC-2822 [16] e em outubro de 2008 pela RFC-5322)*

Na fase de *conexão*, ocorre o pedido de sessão SMTP e a identificação do cliente que envia a mensagem (comando SMTP HELO).

Na fase se *envelopamento*, são declarados o remetente e o destinatário da mensagem.

Na fase de *mensagem*, são escritos os cabeçalhos de identificação do remetente e do destinatário e o assunto da mensagem, podem também ser escritos outros cabeçalhos padronizados além daqueles que são normalmente escritos pelos MTA, porém aqui foram omitidos para simplificar o exemplo. Além disso, separada por uma linha em branco do cabeçalho, segue-se o corpo da mensagem, com a informação que se deseja transmitir. O formato do conteúdo desta fase é ditado pela RFC-5322 [14].

Na fase de *finalização*, que é ativada quando se escreve uma linha com um ponto de finalização (quando se deseja escrever linhas com apenas um ponto de finalização que façam parte do corpo da mensagem, deve-se utilizar caracteres de e*scape*), o servidor informa se a mensagem foi aceita e colocada na pilha de envio.

Na fase de *desconexão*, o cliente informa que quer encerrar a sessão e se desconectar e o servidor informa que está fechando a conexão e a mesma e desfeita.

Toda conexão SMTP é feita em cima do protocolo TCP/IP, o cliente envia comandos (HELO, MAIL FROM, RCPT TO, etc.) e o servidor responde sempre utilizando códigos numéricos padronizados pela RFC-5321 no início das respostas, que identificam se o comando foi ou não aceito e qual o motivo da falha, e podem ou não adicionar mensagens de texto para facilitar o entendimento caso elas sejam lidas por pessoas humanas. Quase sempre são adicionadas mensagens de texto, pois os motivos das falhas são geralmente enviados para os remetentes.

O protocolo SMTP originalmente descrito na RFC-821, pelos padrões atuais de segurança da informação, não seria considerado um protocolo seguro para envio e recebimento de mensagens, pois os endereços passados no comando de conexão HELO e nos comandos de envelopamento MAIL FROM e RCPT TO não são verificados [1], pois não existia essa exigência originalmente. As verificações feitas atualmente por diversos MTAs se devem as extensões no protocolo SMTP e nelas podemos ai incluir as regras antispam.

Notas: [1] *Os campos de cabeçalho da mensagem (From: e To:), que eram originalmente ditados pela RFC-822, também não são verificados quanto à validade dos endereços.*

Tal atitude de aparente desleixo com a segurança se deve ao fato que na época em que a RFC-821 foi escrita, 1982, a Internet era uma rede de acesso restrito apenas ao complexo militar-industrial-acadêmico americano e não se achou necessário na época criar mecanismos de controle nesse sentido (Templeton, Brad [17] apud Wikipédia [1]).

Cabe frisar novamente que a RFC-5321 **não modificou o protocolo SMTP em sua essência**, apenas atualizou o protocolo e adicionou a ele alguns quesitos de segurança. Isso foi feito para garantir a compatibilidade com as redes que rodam o protocolo SMTP legado da RFC-821. Cabe aos desenvolvedores dos MTAs e aos administradores de redes adicionarem regras para restringir o uso de seus servidores SMTP, evitando abusos de *spammers* contra os mesmos.

Para demonstrar a fragilidade do protocolo SMTP, serão feitas alterações no exemplo anterior de forma a ficar evidente o que acontece num hipotético servidor (MTA) rodando apenas o protocolo SMTP, sem nenhuma proteção administrativa extra configurada. A maquina cliente hipotética será hackers.ru [155.134.5.88].

Exemplo de sessão SMTP cliente-servidor modificado:

C(c): telnet mail.sac.br 25
C(c): Connected to mail.sac.br.
C(c): Escape character is '^]'.
S(c): **220 yki.sac.br ESMTP Postfix (Debian/GNU)**
C(c): helo spamtest.com.br
S(c): **250 yki.sac.br**
C(e): mail from: <news@sexnet.com>
S(e): **250 Ok**
C(e): rcpt to:<bruno@inovare.br>
S(e): **250 Ok**
C(m): data
S(m): **354 End data with <CR><LF>.<CR><LF>**
C(m): From: “Laurent Kabila” <kabila@president.gov.rc>
C(m): To: “Bruno Pães” <bruno@inovare.com.br>
C(m): Subject: I need your help!
C(m):
C(m): I need your help, please, call me!
C(m):
C(m): att. Laurent Kabila
C(m): President of Democratic Republic of Congo
C(f): .
S(f): **250 Ok: queued as 56C5176407**
C(d): quit
S(d): **221 Bye**
C(d): Connection closed by foreign host.

Neste exemplo, a máquina hipotética foi hackers.ru (IP: 155.134.5.88) e observamos que todos os endereços em todos os comandos ou campos de cabeçalhos, com exceção dos endereços no comando RCPT TO e no campo *To:* do cabeçalho da mensagem foram forjados e o protocolo SMTP não fez nenhuma restrição aos mesmos.

Em verdade, até 1996 esse tipo de E-mail era conhecido entre usuários e administradores de rede pela denominação de *Fake-Mail* [18] e era uma brincadeira muito popular que se fazia com os internautas de primeira viagem. Hoje em dia esse tipo de E-mail recebe adjetivos mais sérios como *Fraude* ou *Estelionato*. É importante saber que o *Fake-Mail* foi o embrião do SPAM no protocolo SMTP, pois com o passar do tempo se observou que, se era possível enviar esse tipo de brincadeira, então seria possível também enviar mensagens de caráter comercial da mesma forma. Vale lembrar que, até 1996, os servidores SMTP geralmente ficavam com o *relay* (reenvio) de mensagens aberto, o que era considerado de acordo com o principio de solidariedade e união de esforços que a Internet transmitia até então [19].

2.3 – As origens da prática do SPAM

Não existe um consenso quanto à origem da prática do SPAM. Houve dois episódios, um em 1971 ocorrido no CTSS MAIL do MIT com uma mensagem pacifista enviada por Peter Bos (Vleck, T. V. [20] apud Wikipédia [1]) e outro em 1978, ainda na época da ARPAnet, que por alguns é considerado o primeiro caso de envio em massa de uma mensagem não-solicitadas de caráter comercial (Templeton, Brad [17] apud Wikipédia [1]), porém nesta época não se utilizou o termo SPAM para se designar esse tipo de mensagem, que alias era pouco comum naquele tempo, e somente entre o final da década de 1980 e o início da década de 1990 esse termo viria a se popularizar.

Segundo consta na Wikipédia [1], o termo SPAM teria sido originado nos ambientes virtuais de multiusuários conhecidos por MUDs para designar os ataques de *trashing* (destruição) e *flooding* (inundação) destinados a prejudicar os sistemas rodando MUD. Esses ataques passaram a ser conhecidos como *spamming* quando alguns usuários passaram a fazer a analogia com a comédia do grupo Monty Python. Daí essa expressão foi levada para os sistemas BBS pelos usuários de MUD e, posteriormente, passou a ser utilizada pelos internautas egressos dos antigos BBSs para também designar o SPAM no contexto do protocolo SMTP.

2.4 –SPAM vs Phishing, uma diferenciação necessária

Existe atualmente uma aparente confusão que faz com a que as pessoas denominem a prática de *Phishing* [21] como um tipo de SPAM, o que a meu ver está completamente errado pelas características dispares que ambos possuem, apesar de ambos terem uma origem em comum no protocolo SMTP, que foi o *Fake-Mail* e apesar de ambos fazerem uso do envio em massa de mensagens. Vejamos então quais são elas:

- No SPAM, o remetente da mensagem procura, de todas as formas possíveis, dificultar a identificação de seu endereço eletrônico de origem pelo destinatário, forjando endereços falsos e utilizando máquinas com *relay* aberto ou endereços de redes DIALUP ou dinâmicos, para evitar o recebimento de reclamações e protestos que podem ocorrer quando se faz Mala-Direta eletrônica. Geralmente não existe nenhum endereço de resposta válido, mas apenas números de telefones ou de endereços de lojas em cidades. O SPAM é geralmente fácil de ser barrado ao se procederem testes básicos para verificar a veracidade dos endereços passados nos comandos HELO, MAIL FROM e RCPT TO. O caráter do SPAM geralmente é comercial e não tem a intenção de prejudicar financeiramente ou roubar dados do destinatário, mas lhe vender um produto, ainda que o mesmo possa ser ilegal.
- No *Phishing*, o remetente da mensagem procura facilitar que o destinatário tenha algum canal de contato fácil com ele, embora o mesmo esteja geralmente usando uma identidade que não é a dele, i.e. uma identidade roubada, no que juridicamente se denomina da **falsidade ideológica**. O *Engenheiro Social* [22], nome dado ao indivíduo que pratica o *Phishing*, procura utilizar servidores de mensagens legais de provedores de acesso, ainda que possam assinar o serviço com uma identidade falsa, para procurar se cercar com uma “aura” de legalidade de forma que possam passar incólumes por diversos tipos de filtros antispams. O caráter do *Phishing* é geralmente criminoso, procurando ludibriar o destinatário para conseguir roubá-lo através de fraudes ou golpes ou então roubando dados sensíveis a partir da ingenuidade do usuário.

Neste trabalho, eu faço essa diferenciação, pois as regras antispam utilizadas na confecção da LibAntispam são muito mais eficientes para combater a prática do SPAM do que para combater a prática do *Phishing*, pois as regras antispam utilizadas na LibAntispam partem do princípio que o *spammer* tentará esconder sua identidade, em contrapartida, o *Engenheiro Social* adota uma identidade aparentemente "legal" para poder ludibriar a maioria das regras antispam.

Fazendo uma analogia com o correio do mundo real, as regras antispam implementadas na LibAntispam verificam se o carteiro está uniformizado e se a correspondência que ele entrega não possui endereços de remetentes duvidosos. Geralmente no SPAM o carteiro estará sem uniforme (ou até nu) e a correspondência poderá ter endereços duvidosos, enquanto no *Phishing* o carteiro estará bem vestido com seu uniforme e os endereços dos remetentes não serão duvidosos, ficando a interpretação do conteúdo da mensagem a cargo do bom senso do destinatário.

2.5 – As técnicas de SPAM

Segundo a Wikipédia [1], sobre o embate SPAM vs ANTISPAM, "[...] Este conflito é similar a uma guerra armamentista, no sentido de que ambos os lados se esforçam para evoluir a tecnologia favorável a seus interesses e superar a tecnologia favorável ao lado oposto, em um ciclo vicioso.". Dessa forma, necessitamos ver quais são as técnicas utilizadas atualmente para a prática do SPAM para que possamos então ver quais são as técnicas ANTISPAM antagônicas a elas.

As técnicas de SPAM passaram por inúmeras fases [19] e evoluíram de forma bastante sofisticada [23], a seguir temos um resumo das técnicas que podem ser filtradas no primeiro nível de combate ao SPAM, onde está focada a atuação da LibAntispam:

- **SPAM DIRETO E AGRESSIVO:** Essa técnica consiste em enviar o SPAM diretamente a partir do servidor de mensagens de uma empresa ou de um ISP. É basicamente a transposição da Mala-Direta do correio convencional para o sistema de mensagens eletrônicas da Internet (SMTP). Foi comum na 1ª fase do SPAM na Internet (1994-1996), mas é pouco usada atualmente, pois é simples de ser bloqueada.
- **SPAM via servidor Open-Relay:** Surgiu na 2ª fase do SPAM na Internet (1996-1999), consiste no uso "não-autorizado" de servidores

SMTP com o re-envio de mensagens aberto (Open-Relay). Nesta técnica, o *spammer* utiliza servidores de terceiros para enviar as mensagens não-solicitadas, de uma maneira semelhante a usada para enviar-se *Fake-Mails*, cabendo o ônus da culpa pelas mensagens aos administradores que as configuraram erroneamente. É ainda muito utilizado, principalmente devido à má configuração de muitos servidores SMTP.

- **SPAM utilizando de faixas de IPs dinâmicos:** Surgiu na 3ª fase do SPAM (1999-2003), utilizando-se dos IPs das faixas delegadas a ISPs comerciais, que não bloqueiam que maquinas nesta faixa conectem diretamente nos servidores de mensagens de terceiros. Ou seja, o *spammer* utiliza-se de um ISP que permite que seus clientes enviem mensagens eletrônicas sem passar pelo servidor SMTP oficial do ISP. Atualmente o nível de refinamento desta técnica permite que o *spammer* possa até sequestrar temporariamente a faixa de IPs de um ISP usando protocolo de roteamento BGP mal configurado (REKHTER, Y., LI, T. [24] apud TAVARES, D. M. [23]). Outra variante dessa técnica é através da utilização de máquinas zumbis ou *botnets* (RAMACHANDRAN, A., FEAMSTER, N. [25] apud TAVARES, D. M. [23]), onde o *spammer* comanda o envio de mensagens não-solicitadas através de máquinas invadidas por *vírus* [1] ou *worms* [2] do tipo *Trojan* [3] (Cavalo de Tróia).
- **HARVESTING (Coleta de Dados):** Popularmente chamada de WEB-SPAM [19]. É uma técnica (da 3ª fase) de coleta de dados que vasculha páginas na Internet, através de *robots* similares aos usados em sites de buscas do tipo do *Google* ou *Yahoo*, a procura de endereços eletrônicos, através da busca do caractere @ ou seus anacrônicos como *at* ou *[at]*, que são guardados em listas para futuro envio de SPAMs. Outra variante dessa técnica é o uso de pragas digitais como *vírus* e *Trojans* para a coleta de endereços eletrônicos em computadores pessoais.

Notas: [1] *É um código malicioso que se insere no código binário de arquivos executáveis para poder se propagar.*

[2] *É uma variante do vírus, porém se propagando através de E-mails ou sites infectados ou maliciosos.*

[3] *É um programa que permite o controle dos recursos de um computador, sua propagação faz analogia com a celebre lenda do Cavalo de Tróia.*

- **SPAM usando domínios locais no comando MAIL FROM:** É mais um método para tentar ludibriar algumas regras antispam e filtros do que uma técnica em si para tentar reenviar ilicitamente E-mails. Ela consiste no uso de domínios e subdomínios locais para compor o endereço no comando MAIL FROM e é muito utilizada, pois confunde muitos usuários locais incautos da rede local que pensam que o E-mail recebido veio de dentro da rede e não de fora dela. Um exemplo:

  MAIL FROM:<admin@heres.pads.ufrj.br>
  RCPT TO:<szendro@pads.ufrj.br>

- **SPAM-SPOOFING:** É uma técnica da 4ª fase (a mais atual do SPAM, 2004 até a presente data deste trabalho) que consiste na tentativa simultânea de envio de centenas ou milhares de mensagens para os endereços de um domínio ao qual um determinado servidor SMTP seja responsável pelo recebimento de mensagens. O *robot spammer*, utilizando listas de endereços obtidas por *Havesting* ou utilizando técnicas de ataques por dicionários ou por força-bruta (WITTEL, G. L., WU, S. F. [26] apud TAVARES, D. M. [22]) tenta enviar as mensagens não-solicitadas em grande número, o que em muitos casos provoca uma perda de desempenho muito grande do servidor SMTP ou até a queda do mesmo, numa situação semelhante a um *Ataque Distribuído de Negação de Serviço* (DDoS).
- **SPAM-ROUTING:** É uma técnica de 4ª fase e que se utiliza das outras técnicas já vistas como uso de redes *botnets*, seqüestro de faixas de IPs, SPAM-SPOOFING, etc, para tentar enviar o SPAM por diversos caminhos, centralizado as operações no servidor-mestre do *spammer*. Por exemplo, se o *robot* não consegue enviar o SPAM a partir de um caminho, por exemplo, uma faixa de endereços seqüestrada, o servidor mestre pode rotear a mensagem para ser enviada por uma *máquina zumbi* ou um servidor *Open-Relay*. Geralmente só *spammers* profissionais com uma boa estruturação de rede conseguem realizar este tipo de técnica.

Capítulo 3

As Técnicas Antispam de 1º Nível

3.1 – O que é são técnicas antispam de 1º, 2º e 3º níveis?

Como dito anteriormente em 1.3, existem três diferentes níveis técnicos ou abordagens técnicas para se combater o SPAM que são: 1º nível – filtragem na fase de envelopamento da mensagem no servidor SMTP, 2º nível – filtragem do conteúdo da mensagem no servidor e 3º nível – filtragem da mensagem pelo usuário do sistema.

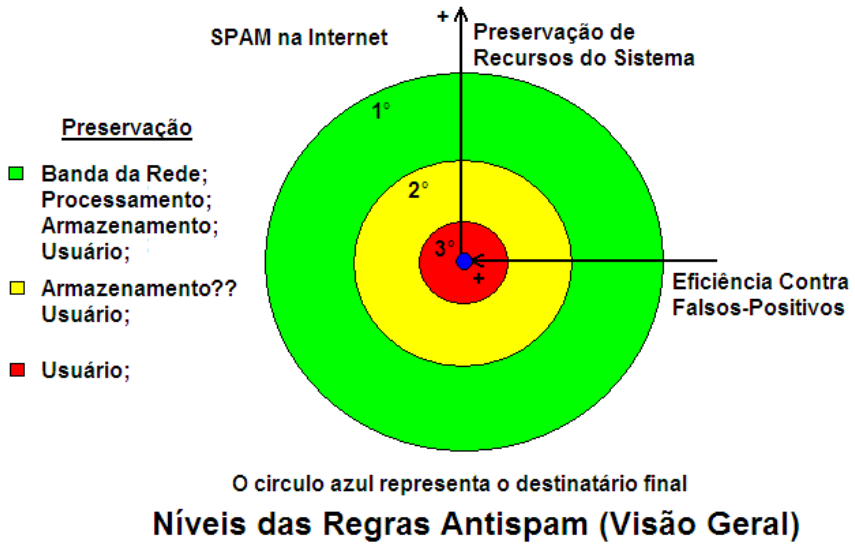

Fig. 3.1 – Níveis técnicos das regras antispam

Fonte: O autor do Trabalho

A filtragem de 1º nível ocorre no momento que os comandos SMTP HELO, MAIL FROM e RCPT TO estão sendo executados num servidor MTA MX do domínio do destinatário e tem a vantagem de ser feita antes da execução do comando SMTP DATA, que permite a recepção do corpo principal com os dados da mensagem, com isso se preserva os recursos computacionais como tempo de CPU e espaço em disco, e ainda se preserva a banda de entrada e saída de dados da rede. A desvantagem das técnicas deste nível de filtragem é que se o servidor/cliente SMTP legal remoto que

envia a mensagem não estiver corretamente configurado, por exemplo, se não tiver o DNS reverso configurado, que é umas das técnicas de verificação que veremos adiante neste capítulo, a mensagem não-SPAM poderá ser rejeitada num caso que se denomina de *falso positivo* para SPAM, o que geralmente é inconveniente ao destinatário.

A filtragem de 2º nível ocorre após o corpo principal da mensagem ter sido recebido, após o comando SMTP DATA. É comum neste ponto já ter ocorrido a desconexão do servidor/cliente SMTP legal remoto, através do comando SMTP QUIT. O corpo da mensagem é verificado para a ocorrência de palavras-chaves comumentemente utilizadas em SPAM, i.e. **sex**, **ofertas**, **lolitas**, etc., geralmente através do uso de *Filtros Bayesianos*. Neste nível de filtragem, a banda de entrada e saída de dados da rede e o tempo de CPU do servidor que abriga o MTA não são preservados, porém a ocorrência de *falsos-positivos* é relativamente menor. Um exemplo de aplicativo utilizado neste nível de mensagem é o *SpamAssassin* [27].

A filtragem de 3º nível é opcional e totalmente de responsabilidade do usuário da rede. Tal qual no 2º nível, ela faz uso de *Filtros Bayesianos* e nela não há preservação da banda de entrada e saída de dados, tempo de CPU, espaço em disco, etc. Porém aqui, neste nível, a ocorrência de *falsos*-positivos é muito menor. Um exemplo de aplicativo usado neste nível é o *filter* do UNIX e os filtros internos de aplicativos como OpenWebMail, ThunderBird, MS Outlook Express, etc.

3.2 – Por que utilizar regras antispam de 1º nível?

Existe uma discussão sobre qual nível de filtragem de SPAM deve ser utilizado para o combate deste mal da Internet da atualidade. Existem argumentos pró e contra cada um deles, então listemos eles partindo do 3º nível até o 1º nível.

Os filtros de 3º nível permitem um controle maior, por parte do usuário final, do que está sendo filtrado e, ainda por cima, permitem que se recuperem quaisquer mensagens onde o *falso-positivo* tenha ocorrido. Porém, o argumento contra esse nível de filtragem e que ele toma muitos recursos do sistema, como espaço em disco, tempo de CPU, recursos de banda de entrada e saída da rede, além de tomarem o tempo de recurso humano, pois o usuário muitas vezes deve descartar por si só as mensagens classificadas como SPAM, podendo ocorrer inadvertidamente a deleção de alguma mensagem legítima, caso o usuário esteja fazendo a deleção com muita pressa.

Os filtros de 2º nível permitem um controle maior, por parte do administrador da rede, sobre o conteúdo do que está sendo filtrado pelos *Filtros Bayesianos*, através da configuração de seus dicionários. Porém, neste nível, os recursos do sistema também não são preservados, tal qual no 3º nível, e ainda, para que o usuário tenha controle do que está sendo filtrado, as mensagens têm a string [SPAM] adicionada aos seus campos "SUBJECT:". Os filtros de 2º nível são os mais utilizados atualmente.

Porém, tanto os filtros de 2º nível como o de 3º nível são vulneráveis a uma modalidade de SPAM que utiliza imagens, onde o texto do SPAM é escrito dentro de uma ou mais imagens gráficas. Como atualmente não existe software capaz de ler textos dentro de imagens gráficas, essa modalidade de SPAM destrói a utilidade dos *Filtros Bayesianos*. Contudo, como muitos usuários não dispõem ainda de conexões de banda larga, essa modalidade de SPAM ainda é pouco utilizada atualmente, porém promete muitas dores de cabeça para os usuários de aplicativos com *Filtros Bayesianos* no futuro (no que será um dia a 5º fase do SPAM na Internet). Segue um exemplo de uma mensagem que poderia muito bem fazer uso dessa modalidade de SPAM:

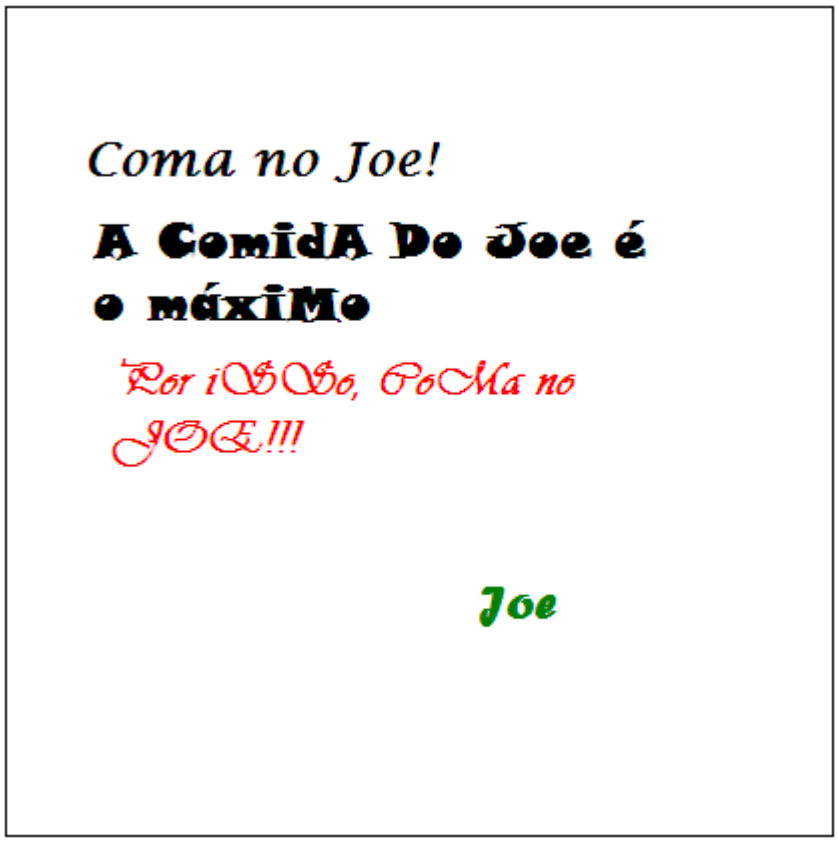

Fig. 3.2 – Exemplo de SPAM com uso de texto em imagem gráfica

Fonte: O autor do Trabalho

Os filtros de 1º nível permitem um controle maior, por parte do administrador da rede, sobre as máquinas remotas que tentam enviar mensagens para os usuários dentro do domínio. A grande idéia por trás deste nível de filtragem se baseia na premissa que os *spammers*, em sua imensa maioria, são covardes e tentarão, de todas as formas possíveis, se manterem anônimos para evitar reclamações diretas por parte dos destinatários finais das mensagens. Desta forma, eles tentam enviar mensagens de máquinas que utilizem IPs sem DNS reverso (dificultar o rastreamento), que utilizem IPs dinâmicos (dificultar o rastreamento), com o *return-path* (endereço passado no comando MAIL FROM) nulo (*null-reverse path*) ou invalido, utilizando máquinas com *relay* de mensagens aberto (*open-relay*), etc. Seguindo essa lógica, criando-se regras que verifiquem a ocorrência de tais casos, pode-se bloquear uma grande quantidade de SPAMs. As duas desvantagens deste nível de filtragem são a ocorrência dos *falsos-positivos* e o fato de não se poder guardar a mensagem que foi rejeitada, caso seja ela um *falso-positivo*. A grande vantagem dela é que a mesma é imune à modalidade do SPAM com texto dentro de imagens gráficas, pois os campos HELO, MAIL FROM e RCPT TO devem ser strings ASCII sempre e não aceitam imagens gráficas. Outra vantagem é a facilidade de se implementarem funções para as regras deste nível, pois se está sempre trabalhando com strings ASCII puras. Daí o foco deste trabalho ser a utilização de regras antispam de 1º nível.

Seja qual for o nível de atuação do filtro antispam, o aplicativo e o tipo de regra de filtragem utilizado, deve-se ter em mente que nunca se conseguirá ter o filtro 100% perfeito, sempre haverá alguma porcentagem de *falsos-positivos* (não-SPAMs) que serão filtrados e alguma porcentagem de *falsos-negativos* (SPAMs) que conseguirão escapar as regras de filtragem, sejam elas quais forem. Assim, recomenda-se sempre utilizar várias regras antispam em conjunto, para minimizar ao máximo a quantidade de *falsos-negativos* que conseguem passar pela filtragem e ficar sempre atento para minimizar ao máximo a ocorrência de *falsos-positivos*.

3.3 – As técnicas antispam de 1º nível

Visto a utilidade das regras de 1º nível, vejamos agora quais são as regras mais utilizadas nele.

- **Filtragem direta de número IP:** Consiste de uma lista de IPs de máquinas que ostensivamente enviam SPAMs. Não é uma técnica muito utilizada, pois a manutenção de listas individuais deste tipo é muito difícil e trabalhosa, mas é disponibilizada por razões históricas.

- **Filtragem utilizando DNSBL ou BlackLists:** O DNSBL (conhecido antigamente como RBL) é um serviço, geralmente gratuito, oferecido por diversas entidades na Internet, que utilizam alguns mecanismos, como *harvesting* ou verificação humana, para catalogar máquinas *spammers* ou *Open-Relays*, e disponibilizam tais listas para consulta na Internet utilizando a infra-estrutura legada dos servidores de DNS. O serviço de DNSBL mais famoso é o MAPS [28], que inclusive foi o primeiro a fazê-lo e já foi o mais utilizado, chegou a ter cerca de 40% das consultas à DNSBLs dirigidas a ele até o serviço passar a ser de subscrição obrigatória. Atualmente os serviços de DNSBLs mais consultados são providos pela SPAMHAUS [29], sediada na Grã-Bretanha.

  O principio de funcionamento de um serviço DNSBL é simples, digamos que temos um serviço desse chamado spamout.xyz.org e que constatamos os IPs 192.168.20.5 e 192.168.244.3 como máquinas *spammers*. Para distribuirmos esta informação usando servidores DNS, devemos criar uma tabela DNS que contenha esses IPs de forma semelhante a uma tabela de DNS direto, porém cada entrada apontará para um número IP da classe A 127.0.0.0/8, com exceção do número IP 127.0.0.1. que é usado nos endereços de loopback das configurações de rede das máquinas que usem TCP/IP:

5.20.168.192.rbl.spamout.xyz.org	IN	A	127.0.0.2
3.244.168.192.rbl.spamout.xyz.org	IN	A	127.0.0.2

Dessa forma, no arquivo de configuração das regras antispam, devo adicionar o serviço rbl.spamout.xyz.org, quando um servidor ou cliente remoto conecta no meu servidor SMTP, ele fará uma submissão de busca DNS utilizando o número IP da máquina que conectou nele, por exemplo, se o número IP 146.164.70.2, a string que busca será 2.70.164.146.rbl.spamout.xyz.org, como essa string não existe na tabela DNS de rbl.spamout.xyz.org, o servidor DNS retornará um código de domínio inexistente, que para o MTA será interpretado como uma mensagem de que está tudo bem e a máquina remota não está listada na BlackList consultada e que ela está habilitada a enviar a mensagem.

- **Filtragem utilizando GrayListing:** A GrayListing (GreyListing) [30] pode ser considerado um meio termo entre a BlackList e a não existência de filtros antispam. O conceito por trás da GrayListing é simples, quando uma máquina remota conecta no MTA para enviar uma mensagem, são guardados o IP da maquina de origem, endereço de origem passado no comando MAIL FROM e o endereço de destino passado no comando RCPT TO. Então o MTA retorna um sinal de *"Temporary Faillure"* e a conexão é rejeitada. Se a máquina que tentou enviar a mensagem foi um servidor SMTP legítimo, ela tentará enviar novamente, depois de transcorrido algum tempo, após o que a mensagem será aceita na segunda tentativa.

  Esta técnica de filtragem antispam se destina principalmente a combater a técnica do SPAM-SPOOFING, pois ela parte do princípio de que como a lista do *spammer* conterá milhares, senão milhões de endereços, o *robot* não tentará reenviar novamente uma mensagem que foi rejeitada.

  A GrayListing é umas das técnicas antispam mais usadas hoje em dia, porém ela apresenta alguns problemas: Está técnica acaba com a expectativa de recebimento quase instantâneo da mensagem pelo destinatário, pois ela provoca um atraso no recebimento da mesma; Alguns servidores SMTP, principalmente de provedores ISP muito usados, podem enviar uma mensagem de erro para o usuário ou simplesmente fazer voltar a mensagem original para ele sem tentar fazer quaisquer novas tentativas de reenvio; Está técnica pode causar

problemas para clientes SMTP remotos que tentem enviar E-mails legítimos através dos servidores SMTP aonde eles tenham contas, pois muitos clientes SMTP não sabem fazer tratamento de filas para os casos de E-mails que recebam uma mensagem de falha transitória. Existe ainda o problema de que muitos *robots spammers* mais elaborados trabalham com uma segunda fila de envio, para tentar reenviar as mensagens que tenham recebido algum tipo de erro na tentativa anterior de envio.
Geralmente, quando um servidor passa na filtragem por SPF, que será vista mais adiante, a filtragem por GrayListing é *by-passada*.

- **Filtragem de máquinas sem DNS reverso**: Esta técnica se baseia na interpretação rigorosa do item 2.1 da RFC-1912 [31], que diz que as entradas PTR e A das tabelas DNS de uma máquina registrada num domínio devem bater, em outras palavras, se uma máquina tem um registro no DNS do domínio exemplo.org para um dado FQDN:

  xxx.exemplo.org IN A 10.0.0.100

  Na tabela do DNS reverso esse número IP deverá apontar exatamente para o FQDN xxx.exemplo.org, i.e.:

  100.0.0.10 IN PTR xxx.exemplo.org

  Assim, parte-se do pressuposto que se a máquina está devidamente registrada nos DNS direto e reverso, então ela é uma máquina legitima.
  A teoria por trás desta regra de filtragem está no fato que, como dito anteriormente, os *spammers* são covardes em sua maioria e não querem ser identificados. Analisando os cabeçalhos de E-mails de SPAM verifica-se facilmente que uma grande parte dessas mensagens vem de máquinas cujos números IPs apresentam algum tipo de falha de registro ou no DNS direto ou, o que é mais comum, no DNS reverso.
  Esta é uma regra pouco utilizada no passado, mas que a partir de três ou quatro anos atrás começou a ganhar cada vez mais adeptos. A grande desvantagem desta regra é que grande parte das redes na Internet não

estão com suas tabelas DNS diretas e reversas corretamente configuradas, seja por ignorância, preguiça ou desconhecimento dos seus administradores de rede, o que pode acarretar um número considerável de *falsos-positivos*. Em verdade, a RFC-1912 continua sendo umas das normas mais desrespeitadas pelos administradores de rede na Internet.

- **Filtragem de strings no DNS reverso:** Esta regra de filtragem é geralmente utilizada junto com a filtragem de DNS reverso vista anteriormente. Ela serve para verificar *strings* típicas de redes *dialups*, com IPs dinâmicos, com IPs estáticos e não delegados ainda (com FQDN genéricos), etc. A idéia por trás dessa regra é que se o servidor SMTP for legal, o seu administrador vai querer que ele esteja bem registrado nos DNS direto e reverso. Por exemplo, costuma-se barrar FQDNs que contenham palavras como **dynamic**, **dinamico**, **dhcp**, **users**, **customers**, **dialup**, etc.

  Tal qual a regra de filtragem anterior, esta regra apresenta como desvantagem o fato que as tabelas de DNS reversos de muitas redes na Internet estão mal-configuradas ou inexistem devido à ignorância, preguiça ou desconhecimento dos seus administradores de rede, logo também nesta regra podem ocorrer casos de *falsos-positivos*.

- **Filtragem por Sender Policy Framework (SPF):** Este tipo de filtragem começou a ser especificado no final dos anos 90 e foi transformado em norma pela RFC-4408 [8] em 2006. Neste tipo de filtragem, a responsabilidade pela publicação ou não das regras de um domínio são da responsabilidade do administrador de redes do mesmo, feita através da adição de uma entrada TXT e/ou SPF no registro SOA da tabela do DNS direto da rede. Um exemplo de uma estrada SPF para o domínio pads.ufrj.br pode ser vista abaixo:

```
pads.ufrj.br            IN      SOA     ns hostmaster.ns (…)

;; Configuração das entradas de SPF do PADS
                IN      TXT     "v=spf1 ip4:146.164.6.209 mx ~all"
                IN      SPF     "v=spf1 ip4:146.164.6.209 mx ~all"
```

O funcionamento do SPF se dá da seguinte forma: Após o comando MAIL FROM ser executado, a regra executa a resolução para procurar por uma entrada TXT ou SPF válida (deve conter a string v=spf1), se não existir nenhuma entrada válida, a ação *default* é deixar passar a mensagem, se a entrada existir, então serão verificadas as regras publicadas na mesma. Por exemplo, nas regras publicadas para o domínio pads.ufrj.br, são permitidos os *relays* a partir do IP 146.164.6.209, a partir das maquinas listadas como MAIL EXCHANGE do mesmo (mx) e para os demais é retornado um SOFTFAIL, que significa que não se pode confirmar a autenticidade das demais máquinas na Internet e deve-se fazer uso de outros testes, como *GrayListing*, para verificar a autenticidade da máquina que está enviando a mensagem.

Como a RFC-4408 é uma norma extensa, deixarei de comentar os detalhes da mesma, porém este tipo de filtragem é bastante popular atualmente, embora um grande número de domínios não publique registros SPF nas suas tabelas de DNS direto.

Cabe explicar, porém, que também este tipo de filtragem também apresenta seus pontos fracos e em alguns casos pode ser até uma *faca-de-dois-gumes*, por exemplo, um *spammer* pode comprar um domínio e utilizá-lo como *laranja* para enviar SPAMs a partir de quaisquer máquinas na Internet, bastando para isso publicar uma entrada do tipo:

IN TXT “v=spf1 +all”

Com a regra acima, consegue-se passar incólume por quaisquer verificações SPF e ainda pode, por tabela, *by-passar* alguns outros filtros como *GrayListing* ou *verificação por reciprocidade mútua*.

Outro problema que a SPF pode provocar é o mau-funcionamento do serviço de *mail forwarding*, i.e. o re-encaminhamento de mensagens para outro endereço utilizando-se, por exemplo, o arquivo *“.forward”* no UNIX, pois se o domínio do remetente declarado no comando MAIL FROM estiver utilizando uma regra muito restritiva, i.e. “-all”, o MTA que faz o re-encaminhamento da mensagem poderá receber uma rejeição do MTA de destino da mensagem.

- **Filtragem verificando a consistência dos endereços de origem:** Neste tipo de filtragem, procura-se verificar a consistência, a validade e a pertinência dos endereços passados no comando MAIL FROM. A filtragem por SPF é um caso particular derivado deste tipo de filtragem.

  Por exemplo, o *NULL-REVERSE path* (que é passado no comando MAIL FROM como <>), é utilizado apenas administrativamente pelo próprio MTA para evitar *loops* de mensagens internas expedidas por ele para o administrador de rede ou os usuários locais reportando erros graves no sistema, logo é pertinente deixar o uso do *NULL-REVERSE path* restrito. Outro exemplo desta verificação é o de restringir o uso de endereços locais no comando MAIL FROM apenas às máquinas e usuários locais da rede. Parece lógico, pois soaria estranho algum cliente na China querer declarar, por exemplo, MAIL FROM:<admin@pads.ufrj.br> quando ele irá mandar um E-mail para RCPT TO:<mariane@pads.ufrj.br>, sendo que esse cliente na China nada tem haver com a rede pads.ufrj.br e/ou sua administração. Outro exemplo ainda desta verificação é colocar caracteres estranhos para endereços de E-mail, mais muito utilizados por *robots spammers* em uma listagem de caracteres proibidos no comando MAIL FROM, por exemplo, endereços com caracteres estranhos (ou seqüencias) do tipo __xjdgf@gmail.com[1], wedsw$@yahoo.com ou $osie#ade$@aol.com.

- **Filtragem baseada na reciprocidade mutua:** Esse tipo de filtragem se baseia no principio dos direitos iguais, ou seja, se você está enviando uma mensagem para mim, então o endereço declarado por você no comando MAIL FROM deve aceitar receber mensagens minhas.

  O funcionamento deste filtro é bastante simples, após um cliente remoto conectar no MTA e após o endereço do remetente ser passado no comando MAIL FROM, a implementação do filtro abre uma conexão

Notas: [1] *O caractere underscore "_" é considerado "non-compliant" nos servidores DNS ISC BIND [32] [33] e não pode ser utilizando no FQDN do endereço, porém pode ser utilizado no username dele. Ele é muito utilizado por robots spammers, porém, infelizmente, também é consideravelmente utilizado por muitos usuários que preferem usá-lo no username ao invés de usar o caractere hífen "-".*

SMTP com os MAIL EXCHANGE ou, na falta destes, com o endereço IP apontado pelo domínio, quando existir. Se a conexão for recusada, ou se o servidor SMTP remoto não aceitar fazer o re-envio da mensagem para aquele endereço de destino, presume-se que ele seja falso e rejeita-se a recepção do corpo da mensagem que iria ser enviada.

Esta regra funciona como um SPF ao inverso, sendo realizado pelo servidor que recebe a mensagem e é bastante comum se encontrar servidores que implementem alguma versão da mesma.

Um dos problemas desta técnica é que, tal qual a GrayListing, ela atrasa o recebimento das mensagens (perda de instantaneidade do E-mail), mas o *calcanhar de Aquiles* desta técnica diz respeito a não-uniformidade das implementações da mesma, pois não há uma norma para ela nem uma especificação oficiosa clara. Muitas vezes ocorrem casos em que um servidor, que utiliza esta técnica, conecta em outro servidor, em que se implementou esta mesma técnica de uma forma um pouco diferente, e um pode tentar testar o endereço de teste do outro, o que pode provocar uma situação de *mail-loop* ou até uma rejeição inesperada por *falso-positivo*.

Outros problemas que se verificam nesta técnica são aqueles causados por códigos de resposta mal escolhidos nas implementações em alguns MTAs, que podem confundir a rotina de verificação e acabar gerando casos de *falsos-positivos*.

Capítulo 4

Especificação e Implementação da LibAntispam

4.1 – Especificações Gerais

Antes da especificação e implementação das regras antispam, teve-se que fazer as especificações gerais que possibilitam não só o correto funcionamento da LibAntispam, como facilitam futuras implementações de novas regras nela. São essas as especificações gerais:

1. A LibAntispam deve ser MTA-independente, ou seja, ela deve ser compilada como uma biblioteca separada e que não esteja intimamente ligada a nenhuma característica particular de qualquer MTA. Dentro do código do MTA deverão ser implementadas as variáveis ou parâmetros necessários para que as chamadas das funções da LibAntispam sejam feitas de forma correta. A biblioteca pronta deverá poder ser *linkada* com qualquer um dos MTAs suportados através de *patches*, sem gerar quaisquer erros de *compilação* ou de *linkedição*.
2. A LibAntispam deve levar em consideração que um MTA pode ser tanto implementado num modelo de programação do tipo *Parent-Child usando forking* como num modelo de programação utilizando *multithreading*. Devido a isto, as chamadas de funções devem prever controles para evitar situações de *race-condition* que causem mau-funcionamento do MTA ou até casos inesperados de *falsos-positivos* e/ou *falsos-negativos*. Para os MTAs que implementam o modelo do tipo *Parent-Child usando forking*, o MTA padrão cujo *patch* servirá de base para a implementação em outros MTAs dessa categoria é o *Sendmail*, já para os MTAs que implementam o modelo por *multithreading*, o MTA padrão cujo *patch* servirá de base em outros MTAs dessa categoria é o *MeTA1*. Quaisquer

novas funcionalidades são primeiro implementadas e testadas nesses dois MTAs e daí replicadas para os demais das categorias relacionada anteriormente.

3. Seguindo a mesma linha de raciocínio, a LibAntispam e a LibAntispam-GUI, que é a interface *WEB* de configuração da LibAntispam, devem prever mecanismos de sinalização do tipo *lock*/*unlock* para evitar que enquanto a primeira esteja carregando as regras dos arquivos de configuração, a segunda esteja escrevendo nos mesmo arquivos (*race-condition*).
4. A LibAntispam deve ser auto-suficiente, ou seja, ela deve implementar todas as funções auxiliares, como por exemplo para tratamento de strings ou resolução de nomes, sem necessitar de nenhuma implementação de qualquer MTA. A LibAntispam deve necessitar apenas das funções primitivas já implementadas nas bibliotecas C do sistema operacional.
5. A LibAntispam é especificada e implementada de forma a esperar que os MTAs suportados através de *patches* passem corretamente os parâmetros, nas chamadas das funções, na ordem correta e que as funções também serão chamadas dentro da ordem esperada. Essa ordem de chamada será explicada mais adiante.
6. Se houver a necessidade do retorno de alguma *string*, por uma função, a LibAntispam proverá *buffers* internos para armazenar temporariamente quaisquer *strings* retornadas e o chamador da função proverá os ponteiros necessários para que as *strings* possam ser corretamente apontadas.
7. Todos os códigos e mensagens de erros internas serão fornecidos pela LibAntispam, sendo que o chamador da função deverá fornecer as variáveis e os ponteiros necessários à eles.
8. Como regra geral, as variáveis globais da LibAntispam não poderão ser acessadas diretamente pelo chamador da função, salvo em casos excepcionais previstos.
9. A LibAntispam é focada na segurança, logo a implementação das funções devem ser feitas de forma a evitar quaisquer riscos que possam a vir a ser uma porta para invadir o sistema. Dessa forma, na programação devem-se evitar erros clássicos como, por exemplo, *buffers overruns*.

10. Finalmente, também como regra geral, a LibAntispam consultará apenas seus próprios arquivos de configuração, que residem no diretório /etc/antispam, independente do sistema operacional Unix-Like utilizado (Linux ou FreeBSD) e independente da distribuição utilizada (Slackware, Debian, Fedora, Ubuntu, etc).

Para facilitar a compreensão, e tendo em mente que a LibAntispam foi inicialmente implementada nele, neste trabalho referenciarei apenas a implementação do *patch* da LibAntispam feita no MTA Sendmail, porém os *patches* para os demais MTAs suportados, MeTA1, Exim e PostFix encontram-se na versão digital deste trabalho, junto com o código-fonte da LibAntispam e do *patch* para o MTA Sendmail.

Para atender o postulado no item 2 das especificações gerais, foi utilizado um controle interno do tipo sinalizador onde, contando com variáveis internas booleanas ou inteiras, se sinaliza para os processos (ou *threads*) "filhos" que o processo (ou *thread*) "pai" está realizando uma operação critica como, por exemplo, re-carregando as configurações antispam que foram modificadas, ou ainda um ou mais processos "filhos" sinalizando ao processo "pai" que eles estão acessando funções e que as configurações não devem ser modificadas até eles sinalizarem os finais das operações. Geralmente, as variáveis sinalizadoras booleanas são de controle e uso exclusivo do processo "pai" e as variáveis sinalizadoras inteiras são de controle e uso exclusivo dos processos "filhos".

Exemplo de controle por sinalização booleana (processo "pai") utilizado na LibAntispam (segmento de código extraído de *mta-heuristics-config.c*):

```
// There are another rebulding process running!
// Prevent race condition!
if(rebuilding_antispam_rules)
{
  a_err_war->antispam_warning = MTA_HR_NTRTRUE;
  return(A_TRUE);
}
```

Neste exemplo, temos um sinalizador global para avisar que está ocorrendo um recarregamento das regras antispam. No Sendmail, a chamada da função da LibAntispam que verifica a modificação dos arquivos de regras fica no *loop* principal

do programa, que se repete a cada 1 segundo e não há riscos de *race-condition*, porém em MTA *multithreads* como o MeTA1 e o Postfix, não há *loop* principal, os processos ficam “dormindo” até a chamada deles serem ativadas pelo processo que escuta a porta 25 de comunicação TCP/IP, só então eles são “acordados”. Dentro da função de entrada do MTA, a primeira a ser executada quando o processo é acordado, existe a chamada da função da LibAntispam que executa o recarregamento das regras. Como podem ter várias conexões SMTP ao mesmo tempo em paralelo num mesmo processo *multithread*, a primeira que executar a função de recarregamento sinaliza para as demais esperarem enquanto a recarga das regras é feita por ela, se necessário, e depois sinaliza para elas que podem prosseguir, pois o recarregamento já terá sido feito.

Se, por acaso, algum outro processo estiver mais adiantado no momento que as regras são atualizadas, por exemplo, numa *query* de resolução de nomes para pegar uma entrada SPF e um grande atraso de tempo está sendo gerado, o mesmo processo poderá verificar se as regras estão sendo atualizadas. A seguir segue um trecho de código extraído do arquivo *mta-heuristics-graylist.c*:

```
// There are a rebulding process running!
// Prevent race condition!
if(rebuilding_antispam_rules)
{
  do
  {
    sleep(1);
    if(!rebuilding_antispam_rules)
      break;
    count++;
    if(count >= MAX_CHILD_SLEEP)
      break;
  }
  while(1);
  count = 0;
}
```

Note que existe um controle de *timeout* para que o processo não fique esperando *ad eternum* que o recarregamento das regras seja feito. Numa máquina rodando com um processador Pentium 4, esse recarregamento não demora mais que 1 segundo.

Exemplo de controle por sinalização inteira (processos “filhos”) utilizado na LibAntispam (segmento de código extraído de *mta-heuristics-reverse-dns.c*):

```
in_libantispam_checking++;

// Verify if mta-heuristics support was disabled!
if(disable_mta_heuristics)
{
  in_libantispam_checking--;
  return(DNS_CHECK_OK);
}

if(!Want_RDNS_control)
{
  in_libantispam_checking--;
  return(DNS_CHECK_DISABLED);
}
```

A sinalização inteira é bem simples, quando um processo *thread* executa uma função da LibAntispam, ele soma uma unidade ao sinalizador. Quando ele termina a execução, ele decrementa uma unidade do sinalizador. Isto permite que o processo “pai”, que no caso dos MTAs *multithread* é aquela *thread* que no momento verifica o recarregamento das regras, verifique se existe alguma função da LibAntispam sendo executada naquele momento. Se sim, o valor da variável *in_libantispam_checking* será maior que zero, caso contrário será zero, conforme podemos verificar no trecho de código abaixo extraído de *mta-heuristics-config.c*:

```
// Wait for all processes accessing LibAntispam functions
// finish its tasks...
if(in_libantispam_checking > 0)
{
```

```
  do
  {
   sleep(1);
   if(in_libantispam_checking <= 0)
   {
     in_libantispam_checking = 0;
     break;
   }
   count++;
   if(count >= MAX_PARENT_SLEEP)
     break;
  }
  while(1);
  count = 0;
}
```

Novamente aqui podemos ver que foi previsto um mecanismo para evitar um tempo de espera *ad eternum*, pois de uma máquina rodando com processador Pentium 4 não deve-se esperar atrasos de tempo que demorem mais que 1 segundo, mesmo para uma resolução de nomes numa entrada SPF não se deve esperar atrasos superiores a 5 segundos.

Para atender a exigência postulada no item 3 das especificações gerais, foi utilizado um sinalizador booleano do tipo *file* (arquivo), onde a existência de um determinado arquivo especifico funciona como um *lock file*. Isto foi feito por três motivos, o primeiro é que é bem mais simples de ser implementado, o segundo é porque se usasse um *socket* de comunicação, seria mais um complicador que poderia provocar o travamento do MTA caso desse algum problema na criação do *socket* ou se o mesmo fosse fechado devido a algum erro grave do sistema operacional (isto costuma ser comum no gerenciados de antivírus Amavis do Unix), o terceiro motivo é porque não há uma utilização tão grande da LibAntispam-GUI que justificasse o uso de um *socket*.

Assim, quando a LibAntispam está lendo os arquivos de configuração, a LibAntispam-GUI não escreve, e quando a LibAntispam-GUI está escrevendo, a LibAntispam não lê. Este tipo de sinalização é conhecido também como o caso clássico dos *Leitores/Escritores*.

Exemplo de controle por sinalização booleana de arquivo (processo “pai”) utilizado na LibAntispam (segmentos de código extraído de *mta-heuristics-config.c*):

```
// Configurations changing... Check if is a zombie lock file
if(stat(CONFIGURATIONS_LOCK_FILE, &stat_buffer) == 0)
{
 // Check if life time of the configurations lock file was reached...
 file_timer = stat_buffer.st_mtime;
 diff_timer = timer - file_timer;

 // If was (file unused), remove it
 if(diff_timer >= MAX_CONFIGURATIONS_LIFETIME)
 {
   unlink(CONFIGURATIONS_LOCK_FILE);
 }
 else // Else... Skip and return late (1 second in Sendmail-8.13.x).
 {
   a_err_war->antispam_warning = MTA_HR_NTRTRUE;
   return(A_TRUE);
 }
}

...

// If reached, then lock configurations access
lockfile = fopen(CONFIGURATIONS_LOCK_FILE, "w");
if(lockfile)
{
 fprintf(lockfile,"locked by %d!\n", (int)getpid());
 fclose(lockfile);

 // To void bad processe's umask
 chmod(CONFIGURATIONS_LOCK_FILE, A_FILE_MODE);
}
```

```
else
{
  a_err_war->antispam_error = MTA_HR_ICCCULF;
  return(A_FALSE);
}
```

Para atender aos postulados dos itens 5, 6 e 7 das especificações gerais, foi descrito um formato comum que todas as funções da LibAntispam devem ter e que é mostrado abaixo nos protótipos de funções genéricas:

```
void        Sz_MTA_Heuristics_FUNCTION_NAME(function_parameters, a_ew_status_t *a_err_war);
int         Sz_MTA_Heuristics_FUNCTION_NAME(function_parameters, a_ew_status_t *a_err_war);
a_boolean_t Sz_MTA_Heuristics_FUNCTION_NAME(function_parameters, a_ew_status_t *a_err_war);
typed_struct *Sz_MTA_Heuristics_FUNCTION_NAME(function_parameters, a_ew_status_t *a_err_war);
```

Todas as funções principais da LibAntispam, especialmente aquelas chamadas pelos MTAs, são identificadas. As chamadas pelos MTAs possuem o prefixo Sz_MTA_Heuristics_ seguido do *nome da função*, seguido dos *parâmetros da função* e por último, um ponteiro que deve ser passado por referencia, ou seja, uma variável de tipo estruturado que em C é passado como *&typed_struct_variable*. Nesta variável são armazenados dois inteiros, o primeiro referente ao código de erro retornado e o segundo referente ao código de aviso retornado. Se os valores deles forem zero significa sem erro e/ou sem aviso. Abaixo pode-se visualizar como é declarado o tipo estruturado de erro ou aviso *a_ew_status_t*, que foi extraído do arquivo *antispam_error.h*:

```
typedef struct a_ew_status_s
{
    /** Antispam error integer. */
    int antispam_error;
```

```
    /** Antispam warning integer. */
    int antispam_warning;

} a_ew_status_t;
```

As strings relativas aos códigos de erro ou de aviso são retornadas pelas duas funções abaixo:

```
char *str_antispam_error(a_ew_status_t a_err_war);
char *str_antispam_warning(a_ew_status_t a_err_war);
```

Quanto ao retorno das funções da LibAntispam, elas podem retornar *void*, *int*, *a_boolean_t* ou até um tipo estruturado, dependendo da necessidade. Cabe ao implementador do MTA que chama a função da LibAntispam criar as variáveis de retorno e o tipo estruturado para retorno do código de erro dentro do escopo do programa onde eles irão atuar. Algumas vez, o retorno de um tipo estruturado pode ser usando como parâmetro de uma outra função da LibAntispam, quando isso for necessário.

Para tender o postulado no item 9 das especificações gerais, fez-se como regra geral que quando é necessário alocar memória para alguma variável, ou quando é criada alguma variável dentro de uma função e seu conteúdo é passado pelo chamador da função ou recebida externamente, por exemplo numa resolução DNS ou passado como parâmetro nos comandos SMTP, se trabalhará com um *offset* de 5 *bytes*, para se evitar ataques do tipo *off-by-one* e ainda se toma o cuidado para sempre verificar o tamanho da *string* que está sendo copiada para o *buffer*. O trecho de código abaixo, extraído do arquivo *mta-heuristics-reverse-dns.c* denota bem essa preocupação e o cuidado adotado na programação:

```
    string_size = strlen(RFC5321_Remote_Hostname_IP_Information);

    // String size is too long...
    if(string_size > BUFFER_SIZE)
    {
      a_err_war->antispam_warning = MTA_HCRD_MDISTL;
```

```
    string_size = BUFFER_SIZE;
  }

  hostname_ip_information =
  (char*)malloc(string_size+MALLOC_SIZE_OFFSET);

  if(!hostname_ip_information)
  {
    a_err_war->antispam_error = MTA_HCRD_OOMM;
    in_libantispam_checking--;
    return(DNS_CHECK_TEMP_FAIL);
  }

  // Reset alloced memory with '\0' chars
  memset(hostname_ip_information, 0, (string_size +
MALLOC_SIZE_OFFSET));

  // Copy RFC5321_Remote_Hostname_IP_Information
  strncpy(hostname_ip_information,
RFC5321_Remote_Hostname_IP_Information, string_size);
```

Nota-se que como regra geral, prefere-se uma perda de informação essencial, que no caso por ser exageradamente muito grande pode significar uma tentativa de ataque, do que correr o risco de comprometer a segurança do sistema.

Finalmente, para atender o postulado no item 10 das especificações gerais, foram definidos os arquivos de configuração em formato ASCII que residirão no diretório /etc/antispam, e que a seguir são listados:

mta-antispam.conf: Arquivo de configuração principal da LibAntispam, possui as configurações gerais da rede (IP, domínios, etc.), habilitação/desabilitação de regras antispam, configurações de ajuste para algumas regras e listas especiais internas.

accepted.hosts: Arquivo que contém os *hosts* aceitos pelo administrador e que poderão *by-passar* algumas regras.

control.users: Arquivo com a ativação/desativação de regras personalizadas por usuário, domínios, etc.

mail-addresses.good: Arquivo que contém endereços de E-mail considerados seguros pelo administrador e aceitos sempre sem qualquer verificação.

mail-addresses_or_domains.bad: Arquivo que contém endereços de E-mail ou domínios considerados inseguros e que deverão sempre ser rejeitados pela LibAntispam.

prohibited.hosts: Arquivo que contém os *hosts* proibidos pelo administrador e que deverão sempre ser rejeitados.

rejected.FQDN.strings: Arquivo que contém strings proibidas e que se aparecerem nos FQDNs, apontados no DNS reverso, devem provocar a rejeição do recebimento da mensagem.

users (diretório): Diretório que pode armazenar algumas regras e listagens personalizadas de cada usuário listado no arquivo *control.users*.

4.2 – A Carga, Recarga e Descarga das Regras de Configuração

Para a carga (*load*), recarga (*reload*) e descarga (*unload*) das regras de configuração, foi definido que todas as regras dos arquivos de configuração em formato ASCII são carregadas em variáveis e estruturas globais, com alocação dinâmica de memória se necessário, recarregadas da mesma forma quando desejado e descarregadas com desalocação da memória utilizada. As motivações por esse tipo de abordagem são os seguintes:

1. O acesso a memória RAM é mais rápido do que o acesso ao HD;
2. O custo em Mb de memória RAM é baixo nos dias de hoje;
3. Os administradores de rede preferem editar arquivos de configuração em formato ASCII e não gostam de ter que rodar aplicativos para convertê-los em formato de banco de dados;
4. Este projeto foi direcionado pensando-se em termos da LibAntispam ser utilizada em redes pequenas e médias, não mais de 1000 usuários, logo a alocação de recursos de memória não será crítica;
5. Arquivos de configuração em formato texto são mais facilmente intercambiáveis, e comunicação com bancos de dados, por *socket* ou TCP/IP pode trazer resultados indesejáveis se o servidor de bancos de dados tiver algum problema;

6. O autor do trabalho não domina o uso de banco de dados, porém não descarta no futuro a utilização de algum sistema de banco de dados neste projeto.

Para a carga inicial das regras foi criada a função *Sz_MTA_Heuristics_Init()* que lê os arquivos de configuração e armazena suas informações em variáveis e estruturas globais na memória RAM. Esta função deve ser adicionada junto às funções de inicialização dos MTAs.

Caso qualquer um dos arquivos de configuração seja modificado, uma nova carga das regras é determinada e feita pela função *Sz_MTA_Heuristics_Rebuild()* que deve ser sempre colocada dentro do *loop* principal do MTA, caso o mesmo seja do tipo *Parent-Child usando forking*, ou então dentro da função de entrada do processo de negociação SMTP, caso o MTA seja do tipo *multithread*.

Para a descarga total e correta das regras da memória RAM, foi criada a função *Sz_MTA_Heuristics_Clear()* que deve ser adicionada nas rotinas de finalização e de saída por erro crítico dos MTAs. Esta função varre as variáveis de ambiente e as estruturas as quais tenha sido necessário alocação dinâmica de memória, desalocando a memória utilizada e resetando os ponteiros para apontarem para NULL.

4.3 – Sintaxes dos Parâmetros Passados pelos MTAs

Existem três parâmetros que todos os MTAs deverão passar para as funções da LibAntispam em algum momento, em algumas funções poderão ser necessários menos de três parâmetros, porém o importante é que os MTAs deverão seguir uma sintaxe correta esperada para cada um desses parâmetros, de forma as funções da LibAntispam sejam executadas corretamente. Os parâmetros são:

- **RFC5321_Remote_Hostname_IP_Information:** Este parâmetro informa o hostname e o número IP do cliente remoto que está conectado ao servidor SMTP enviando a mensagem. Este parâmetro segue a indentação original do Sendmail que é:

  Hostname [número_IPv4] ou Hostname [IPv6:número_IPv6]

Exemplo: helena.pads.ufrj.br [146.164.48.7] para IPv4
helena.pads.ufrj.br [IPv6:2001:12f0:4c1:6643::7] para IPv6

Obs.: A informação do *hostname* é opcional e pode ser omitida!

- **Mail_From:** Este parâmetro informa o endereço passado no comando MAIL FROM é deverá ter o formato <username@domainname>.

  Exemplo: <szendro@pads.ufrj.br>

- **Rcpt_User:** Este parâmetro informa o endereço passado no comando RCPT TO é deverá ter o formato <username@domainname>.

  Exemplo: <admin@del.ufrj.br>

Existe ainda um quarto parâmetro especial, mas ele é gerado pela própria LibAntispam quando a função *Sz_MTA_Heuristics_Search_In_Controled_Users_List()* é executada, aliais, essa função DEVE ser executada obrigatoriamente antes de quaisquer outras funções que necessitem desse parâmetro dentro do mesmo escopo local. O quarto parâmetro é definido como um tipo estruturado *control_user_search_r_t* e nele são retornadas as informações sobre o endereço de E-mail (usuário) passado no comando RCPT TO e que estão gravadas nos arquivos de configuração control.users e no arquivo de configuração pessoal, se o mesmo existir. Caso o usuário não exista no arquivo de configuração, opções *default* definidas pelo administrador de redes são retornadas nesse tipo estruturado. Abaixo podemos visualizar o tipo estruturado *control_user_search_r_t* extraído do arquivo *mta-antispam.h*:

```
/**
 * \struct control_user_search_r_s
 * \brief Store the results for a search in control-users list.
 * The structure for antispam heuristics for a search in control-users list.
 * \sa Sz_MTA_Heuristics_Search_In_Controled_Users_List()
 */
```

```
typedef struct control_user_search_r_s
{

    /** Global user search return code that is shared with several LibAntispam
functions. */
    uint64_t global_user_search_return_code;

    /** The relative position of the user in some lists used by some
LibAntispam functions. */
    int global_user_search_position;
    /** Global user search type that is used with several LibAntispam
functions. */
    global_user_search_type_t global_user_search_type;

    /** User found in reduced (login username only) format. */
    a_boolean_t global_user_found_in_reduced_format;

} control_user_search_r_t;
```

Por motivo de clareza, listo também o tipo *global_user_search_type_t* que aparece no tipo estruturado *control_user_search_r_t*:

```
/**
 * \typedef global_user_search_type_t
 * \brief Global user search types definition
 */
typedef enum
{
    NOT_USER                = 0,
    DEFAULT_USER            = 1,
    NORMAL_USER             = 2
} global_user_search_type_t;
```

A motivação de este parâmetro existir e de ter que ser gerado no mesmo escopo local onde as demais funções da LibAntispam serão executadas é simples, para que a LibAntispam possa trabalhar com MTAs *multithreads*. No passado, quando a LibAntispam só funcionava no Sendmail, não havia essa necessidade, pois por *forking* era gerado um processo separado do processo pai e a variável *control_user_search_r_t* podia ser uma variável global e interna da biblioteca, porém no MeTA1 e no Postfix, isso não é possível, pois existe ai o risco real de *race-condition*.

4.4 – Especificações e Implementações das Regras Antispam

As especificações das regras antispam da LibAntispam foram feitas tomando-se por base o que já é de uso comum delas pelo administradores de redes. Cabe relembrar que as regras antispam em sua maioria, à exceção da SPF, por exemplo, não possuem uma norma que formalize como as regras deverão se comportar. Tais regras antispam surgiram mais por motivações de desespero e necessidade de se conter o SPAM do que por um desejo de se criarem novas normas para o protocolo SMTP.

Uma vez coletadas as informações sobre o funcionamento mais aceito para cada regra, uma especificação formal é feita para a mesma e ela é implementada e testada, só então ela é oficialmente acrescentada na biblioteca.

A seguir são listados os procedimentos de especificação e implementação de cada regra antispam adotada, por ordem de implementação:

4.4.1 - Proteção básica por número IP ou de rede

Está regra é a mais básica de todas e é implementada por razões históricas, apesar de ser a mais facilmente burlável, pois deve-se ter uma forma de bloquear uma máquina pelo número IP da mesma. Não é admissível um sistema antispam que não tenha esse tipo de filtro.

Especificação: Na especificação desta regra, foi determinado que há três métodos básicos de filtragem de números IP e de redes, o primeiro é colocando o número IP que se deseja bloquear, por exemplo, 222.222.222.222. O segundo é se colocando o número de uma rede sem o uso de CIDR, que é como o

Sendmail faz, por exemplo, 222.222.222.. O terceiro método é através do uso de CIDR como é usado modernamente, por exemplo, 222.222.222.0/26.

Implementação: Na implementação desta regra, foi feito o design da função *Sz_MTA_Heuristics_Check_For_Prohibited_IP_Network()*, que é a função chamada pelos MTAs para que a LibAntispam cheque se um determinado numero IP é um número IP proibido ou pertence a uma rede proibida. Ao varrer a listagem de números IP ou redes proibidas, esta função verifica a cada elemento da lista se ele é um IP completo, um número de rede sem CIDR ou um número de rede com CIDR, chamando as funções auxiliares, se necessário e retornando um tipo estruturado *prohibited_ip_net_r_t* que reporta o status final da verificação. A seguir é mostrado o código com a estrutura do tipo *prohibited_ip_net_r_t*, extraído do arquivo *mta-antispam.h*:

```
/**
 * \struct prohibited_ip_net_r_s
 * \brief Store the results for check prohibited IP and Networks.
 * The structure for antispam heuristics check prohibited IP/Networks result.
 * \sa Sz_MTA_Heuristics_Check_For_Prohibited_IP_Network()
 */
typedef struct prohibited_ip_net_r_s
{

    /** Result of the check. */
    a_boolean_t     has_or_belongs_prohibited_ip_or_network;

    /** Pointer to denied ip or Network. */
    char *          deny_ip_network_ptr;

} prohibited_ip_net_r_t;
```

Internamente, esse tipo possui uma variável booleana que informa se o IP pertence a um número IP ou de rede proibidos e, caso positivo, é retornando um ponteiro para o mesmo.

Um destaque pode ser feito para a função interna da LibAntispam, *Sz_MTA_Heuristics_Compare_IP_Network()* que se encontra no arquivo *mta-antispam-internal.c*, que é um *front-end* para a função interna *compare_ip_network()*, que se encontra no arquivo *ip_utils.c*, é esta última que faz a comparação do número IP com um número de rede no formato CIDR. Para isto ser feito, é necessário se extrair o netmask no formato CIDR, o que é feito pela função *Sz_MTA_Heuristics_Compare_IP_Network()* e aplica-lo tanto ao IP como ao número de rede, o que é feito pela função *compare_ip_network()*, que chamando a função *fast_search_network_number()*, também do arquivo *ip_utils.c*, faz a operação matemática para descobrir o número de rede tanto do IP como do número de rede passado (para uma checagem final de segurança). O resultado disso é repassado para a função *compare_ip_network()*, que irá comparar ambos os resultados e, se forem iguais ou diferentes, retornará o resultado de sucesso ou insucesso, conforme for o caso.

4.4.2 - Checagem de DNS reverso (RFC-1912)

Esta regra é uma das que mais vêm se adotando nos últimos anos. O motivo de não ter sido adotada antes se deve a quantidade muito grande de servidores SMTP legais que não tinham o DNS reverso corretamente configurado até meados de 2003. Hoje em dia essa situação mudou, embora o número de servidores SMTP com problemas no DNS reverso ainda seja considerável.

Especificação: Apesar desta regra não ter uma especificação formalmente escrita até hoje, ela é totalmente baseada no item 2.1 da RFC-1912 que diz que se um número IP está alocado para uso na Internet, obrigatoriamente este IP deve apontar reversamente para um FQDN (ele pode apontar para vários FQDNs) e, por sua vez, esse mesmo FQDN deve apontar diretamente para aquele número IP (ele pode apontar para vários IPs). Desta forma, a especificação desta regra será: O IP da máquina que conecta no MTA deve ter ao menos um apontamento reverso para um FQDN, e esse mesmo FQDN deve ter ao menos um apontamento direto para o número IP da máquina que conecta no MTA.

Implementação: Na implementação desta regra, foi feito o design da função *Sz_MTA_Heuristics_Check_Reverse_DNS()*, que é a função que os MTAs devem executar para que a LibAntispam proceda o teste de DNS reverso. Esta função é basicamente simples de ser implementada, mesmo que o número IP da máquina remota tenha múltiplos apontamentos (o que não é comum de acontecer), o importante é verificar se qualquer um desses apontamentos, não importa qual, obedece a essa regra. Portanto, essa função faz uma simples chamada a função C *gethostbyaddr()* para obter um PTR válido para esse IP, se ele não existir, é retornado a sinalização de erro para que o reenvio da mensagem seja negado, caso contrário, a partir do FQDN retornado, utiliza-se a função C *gethostbyname()* para obter uma lista dos IPs válidos que são apontados por esse FQDN, se nenhum dos apontamentos corresponder ao IP da máquina remota, então é retornado a sinalização de erro nesse caso e o reenvio da mensagem é negado.

A codificação de erro retornado por *Sz_MTA_Heuristics_Check_Reverse_DNS()* é um tipo enumerativo, abaixo mostramos o mesmo a partir do fragmento de código do arquivo *mta-heuristics.h*.

```
/**
 * \typedef dns_error_t
 * \brief Antispam heuristics dns_error_t type definition.
 */
typedef enum
{
      DNS_CHECK_DISABLED     = 0,
      DNS_CHECK_OK                = 1,
      DNS_CHECK_FAIL              = 2,
      DNS_CHECK_TEMP_FAIL    = 3,
      DNS_CHECK_FORGED         = 4
} dns_error_t;
```

Como observação final sobre essa regra antispam, observe que o código DNS_CHECK_TEMP_FAIL não é uma sinalização de erro, ele indica uma falha na resolução DNS, por algum problema no servidor DNS remoto, assim, a

ação *default* para este caso deve ser de retornar um código SMTP 4.x.x para indicar uma falha temporária e não 5.x.x que é usado para as falhas permanentes.

4.4.3 - Checagem de strings no DNS reverso

Essa regra é tem por objetivo verificar os FQDNs retornados no DNS reverso, caso existam, procurando strings que indiquem uma provável delegação de uso que esteja sendo feita para o número IP ou para a faixa de rede a qual aquele IP pertença. Dependendo do uso que esteja sendo feito da faixa, pode-se rejeitar o reenvio de mensagens, pois não se pode esperar receber uma mensagem legal de uma faixa de IPs dinâmicos, por exemplo. A idéia por trás dessa regra é que grande parte dos ISPs que criam o DNS reverso das faixas delegadas a eles, o fazem utilizando scripts ou programas que colocam nomes genéricos para identificar a utilização dada para a faixa. Por exemplo, é comum se encontrar faixas configuradas como *xxx.users.netvirtua.com.br*, yyy-*dhcp.net.br*, *zzz.customers.brasiltelecom.com.br* ou *20119104023.user.veloxzone.com.br*.

Especificação: A especificação para esta regra será então fazer-se a resolução do DNS reverso para o número IP da máquina de origem conectada ao MTA, caso a entrada de DNS reverso não exista, ignora-se a verificação é retorna-se ponteiro nulo, indicando que não existe DNS reverso. Caso a entrada exista, consulta-se a lista de *strings* proibidas de existir no FQDN retornado pelo DNS reverso, caso haja alguma *strings* proibida no FQDN, retorna-se um ponteiro para essa *string*, o que é interpretado como um motivo para rejeição do reenvio da mensagem pelo MTA, caso contrário, retorna-se ponteiro nulo para indicar que a busca não encontrou nenhuma *string* proibida no FQDN.

Implementação: Na implementação desta regra, foi feito o design da função *Sz_MTA_Heuristics_Search_In_Rejected_List()*, esta função internamente faz uso da função *Get_Reverse_DNS()*, implementada no arquivo *resolv_utils.c*, que utiliza a função *gethostbyaddr()*, porém aloca memória para copiar a string para um *buffer* temporário. Isto é feito para evitar problemas de *race-condition*, uma vez que a busca por strings proibidas e outras verificações feitas dentro da função *Sz_MTA_Heuristics_Search_In_Rejected_List()* podem demandar alguns

segundos enquanto o *buffer* da função *gethostbyaddr()* é modificado a cada chamada da mesma. Há dois tipos de busca feita dentro da função *Sz_MTA_Heuristics_Search_In_Rejected_List()*, a primeira é uma busca linear por *strings* proibidas, a segunda é uma implementação de comandos especiais que são passados dentro do arquivo *reject.FQDN.strings* para verificar por seqüenciamentos típicos usados em redes de máquinas com IPs temporários que certos ISPs, por conivência ou não com as atividades *spammers*, colocam no DNS reverso de suas faixas para dificultar esse tipo de varredura. Por exemplo, o domínio *20119104023.user.veloxzone.com.br* mostrado acima corresponde ao número IP 201.19.104.23, embora o Velox tenha colocado a *string users* para indicar faixa de usuários, vários ISPs costumam colocar coisas do tipo 176223589.domain.com apontado para 176.223.5.89, por exemplo. Se a *string domain* for um FQDN conhecido como *gmail* ou *yahoo*, não posso utilizar nenhuma dessas *strings* e nem *.com*, pois estaria-se rejeitando milhares E-mails legítimos inutilmente.

A saída, por exemplo, é verificar por seqüenciamentos numéricos com mais de quatro algarismos ou então seqüenciamentos numéricos que batam com os algarismos dos números IPs e rejeitar reenvio de máquinas nestas condições. A função interna *Sz_MTA_Heuristics_Execute_Special_Command()*, procura por comandos especiais dentro do arquivo de configuração, que são referenciados pelo caractere '!' no início do comando, pois esse caractere não pode ser usando em FQDN e por isso é um caractere neutro no arquivo *reject.FQDN.strings*, e se achar algum seqüenciamento que bata com alguma das regras de comando, a função *Sz_MTA_Heuristics_Search_In_Rejected_List()* retorna um erro semelhante a se tivesse encontrada um caractere proibido na listagem, porém avisando que se trata de um seqüenciamento proibido de caracteres.

A seguir estão listados todos os comandos especiais possíveis, extraídos de um fragmento do arquivo *reject.FQDN.strings-sample* padrão de instalação da LibAntispam:

```
#
# Special command rules to catch strings in dynamic or
# unused/non-registered static addresses
#
```

```
# Anything that has three digit sets or more separated by
# dashs will be rejected.
!cns(-,3)
# Anything that has three digit sets or more separated by
# dots will be rejected.
!cns(.,3)
# Mean that a group of five or more digits will be rejected.
!cng(5)
# Mean that FQDNs that use all digits of the pointed IPv4 numbers,
# in decimal or hexadecimal format, will be rejected.
!cip4fqdn()
# Mean that FQDNs that use all digits of the pointed IPv6 numbers
# will be rejected.
!cip6fqdn()
```

4.4.4 - Checagem de endereços no comando HELO

Esta regra tem por objetivo checar a consistência dos FQDN passados no comando HELO e objetivava originalmente ter um endereço alternativo consistente para a checagem por SPF conforme descrito na RFC-4408, que será vista mais adiante. Porém, ocorre aqui um paradoxo, pois segundo os tópicos 2.2 e 2.3 da RFC-4408, caso o endereço no comando MAIL FROM não possa ser verificado, pois ele pode ser passador com número IP literal sem DNS reverso, i.e. <szendro@[146.164.48.16]> ou pode ser passado um *null-reverse path*, então deve ser feita a verificação pelo FQDN passado no comando HELO, o que pode ocasionar a rejeição da mensagem, mas segundo o tópico 5.2.5 da RFC-1123 [34], a informação passada no comando HELO não deve ser motivo para a rejeição da mensagem. Logo temos aqui o caso raro de duas RFCs em conflito uma com a outra. Abaixo transcrevemos os fragmentos dos tópicos em questão da RFC-1123 e da RFC-4408:

Segundo o tópico 5.2.5 da RFC-1123 [34], *"The HELO receiver MAY verify that the HELO parameter really corresponds to the IP address of the sender.*

However, the receiver MUST NOT refuse to accept a message, even if the sender's HELO command fails verification."

Já, segundo o tópico 2.1. da RFC-4408 [8], *"It is RECOMMENDED that SPF clients not only check the "MAIL FROM" identity, but also separately check the "HELO" identity by applying the check_host() function (Section 4) to the "HELO" identity as the <sender>."*

E ainda, segundo o tópico 2.2 da RFC-4408 [8], *"[...] When the reverse-path is null, this document defines the "MAIL FROM" identity to be the mailbox composed of the localpart "postmaster" and the "HELO" identity (which may or may not have been checked separately before)."*

Embora a checagem e eventual rejeição de E-mails devido a FQDNs inconsistentes do comando HELO sejam uma violação da RFC-1123, este tipo de checagem está se tornando bastante freqüente, mesmo quando não é feita verificação de SPF, pois os *robots spammers* são feitos para gerar *strings* aleatórias ou usando IPs literais de várias maneiras, como no caso do uso do endereço IP apontado no domínio de destino para ser usado de forma literal no comando HELO.
No caso deste tipo de regra, deve-se sempre colocar uma observação na mesma para que o administrador fique avisado que se a habilitar estará violando a RFC-1123.

Especificação: A especificação desta regra ficou então da seguinte forma: O argumento passado no comando HELO não pode ser nulo; Se forem passados números IPs literais, não se pode ter números IPv6 em conexões IPv4 e vice-versa; OS IPs literais devem ser passados em formato literal válido; Os números IPs locais da rede que recebe a mensagem não podem ser utilizados por clientes remotos não-locais; Os números IPs literais devem ter registro DNS reverso; Os IP literais não podem ser RFC-1918 [35]; O uso de IPs literais podem ser banidos pelo administrador; Os FQDNs, quando não possuem resolução DNS direta devem obrigatoriamente possuir entradas MX para IPs válidos; Caso os FQDNs apontem para números IPs RFC-1918, obrigatoriamente devem possuir

entradas MX para IPs válidos; Entradas MX válidas não podem apontar para IPs válidos da rede que recebe a mensagem, salvo os clientes remotos sejam locais; Erros temporários de DNS devem retorna código de erro temporário pelo MTA; Os FQDNs devem conter ao menos um caractere ponto, sendo que não podem iniciar nem terminar por ponto; O FQDN não poderá conter nenhum dos caracteres proibidos definidos pelo administrador/usuário e não poderá conter também os caracteres @, <, > e virgula, sendo estes quatro últimos obrigatoriamente; Os FQDNs, os IPs apontados por eles, os IPs literais e os FQDNs apontados pelos mesmos não podem estar na tabelas de domínios e IPs proibidos pelo administrador da rede.

Implementação: A implementação desta regra foi bastante complexa, pois como se pode observar pela especificação da mesma, existem vários pontos críticos que devem ser bem tratados para não permitir buracos que permitam aos *spammers* burlarem a verificação. Como pode ser verificado abaixo no fragemento do arquivo *mta-heuristics.h*, a especificação dos códigos de retorno desta verificação chegou a trinta tipos de possíveis códigos de resposta.

```
/**
 * \typedef helo_error_t
 * \brief Antispam heuristics helo_error_t type definition.
 */
typedef enum
{
    HELO_ERROR                                  = 0,
    HELO_OK                                     = 1,
    HELO_ARGS_REQUIRED                          = 2,
    HELO_IP_NOT_LITERAL                         = 3,
    HELO_IP_FORMAT_INVALID                      = 4,
    HELO_IPV4_RFC1918                           = 5,
    HELO_IPV6_HELO_IN_IPV4_CONNECTION           = 6,
    HELO_IPV4_HELO_IN_IPV6_CONNECTION           = 7,
    HELO_ARGS_MY_IP_AND_NETWORKS                = 8,
    HELO_ARGS_MY_DOMAINS                        = 9,
    HELO_DOMAIN_HAS_PROHIBITED_CHARS            = 10,
```

```
        HELO_DOMAIN_HAS_NOT_DOT                          = 11,
        HELO_DOMAIN_START_WITH_DOT                       = 12,
        HELO_DOMAIN_END_WITH_DOT                         = 13,
        HELO_DOMAIN_RESOLVE_INVALID_IPV4                 = 14,
        HELO_DOMAIN_RESOLVE_INVALID_IPV6                 = 15,
        HELO_DOMAIN_NOT_RESOLVE_AND_NOT_HAS_MX           = 16,
HELO_DOMAIN_RESOLVE_INVALID_IPV4_RFC1918_AND_NOT_HAS_MX
= 17,
        HELO_DOMAIN_BAD_MX_RECORD                        = 18,
        HELO_DOMAIN_RESOLVE_MX_INVALID_IPV4              = 19,
        HELO_DOMAIN_RESOLVE_MX_INVALID_IPV4_RFC1918      = 20,
        HELO_DOMAIN_RESOLVE_MX_INVALID_IPV6              = 21,
        HELO_DOMAIN_RESOLVE_MATCH_MY_IP_NETWORKS         = 22,
        HELO_DOMAIN_MX_MATCH_MY_DOMAINS                  = 23,
        HELO_DOMAIN_MX_RESOLVE_MATCH_MY_IP_NETWORKS
    = 24,
        HELO_DOMAIN_DNS_TEMP_FAIL                        = 25,
        HELO_HAS_PROHIBITED_IP_NETWORK                   = 26,
        HELO_PROHIBITED_ALL_IPV4_NUMBERS                 = 27,
        HELO_PROHIBITED_ALL_IPV6_NUMBERS                 = 28,
        HELO_HAS_PROHIBITED_DOMAIN                       = 29
    } helo_error_t;
```

A implementação desta regra resultou na criação da função *Sz_MTA_Heuristics_Helo_Check()*, está função retorna um tipo estruturado denominado *v_helo_r_t*, que é mostrado a seguir no fragmento de código do arquivo *mta-heuristics.h*:

```
/**
 * \struct v_helo_r_s
 * \brief Store the results for a bad helo ckeck.
 * The structure for antispam heuristics mta helo check result.
 * \sa Sz_MTA_Heuristics_Helo_Check()
 */
```

```
typedef struct v_helo_r_s
{

    /** HELO ckeck return code. */
    helo_error_t        return_value;

    /** Pointer to a prohibited character found. */
    char                *prohibited_char;

    /** Pointer to a bad HELO found. */
    char                *bad_helo_domain_ptr;

} v_helo_r_t;
```

Este tipo estruturado prevê o retorno do código de retorno para a busca da função e mais dois ponteiros, um para apontar para o caractere proibido e outro para um FQDN ou IP proibido, caso das respectivas rotinas acharem os mesmos. É importante comentar que a função *Sz_MTA_Heuristics_Helo_Check()*, no caso de encontrar um FQDN ou IP literal proibido, o armazena numa pilha de *buffers* internos, porque em alguma situações faz *parsing* de *strings*, cujo resultado pode ser modificado a cada chamada de *strsep()*, por isto deve-se evitar uma situação de *race-condition*, pois o MTA que está executando a função *Sz_MTA_Heuristics_Helo_Check()* pode ter arquitetura *multithread* e a cada chamada dessa função o *buffer* seria apagado se o mesmo não fosse feito em formato de pilha. Os segmentos de código abaixo, extraídos do arquivo *mta-heuristics-helo-check.c* mostram claramente como é feito tal controle de *buffers* internos:

```
static  int  bad_helo_domain_control = 0;
        int  bad_helo_domain_control_now = 0;  // To void race condition

…
```

```
    // Copy the stack control pointer location
    bad_helo_domain_control_now = bad_helo_domain_control;

    // Increment the stack control pointer
    bad_helo_domain_control++;

    // Reset bad_helo_domain array with '\0' chars
    memset(bad_helo_domain[bad_helo_domain_control_now], 0,
(BUFFER_SIZE + MALLOC_SIZE_OFFSET));

    helo_string_size = strlen(Helo);

    helo_information = (char *)malloc(helo_string_size +
MALLOC_SIZE_OFFSET);
    if(helo_information)
    {
      // Reset alloced memory with '\0' chars
      memset(helo_information, 0, (helo_string_size +
MALLOC_SIZE_OFFSET));

      // Copy Helo Information
      strncpy(helo_information, Helo, helo_string_size);

      // Lower case the copy
      a_strlwr(helo_information, a_err_war);
    }
    else
    {
      a_err_war->antispam_error = MTA_HHC_OOMM;
      result.return_value = HELO_ERROR;
      in_libantispam_checking--;
      return(result);
    }
```

Basicamente, a função *Sz_MTA_Heuristics_Helo_Check()* faz resoluções DNS e comparações de *strings* com elementos das listas de FQDNs, IPs e Números de Redes proibidos no comando HELO, para verificar a consistência do argumento passado no mesmo, conforme mostrado na especificação da regra. Existem algumas funções auxiliares importantes que são chamadas internamente nesta função e que listo a seguir:

compaddr_to_full_uncompaddr(): Está função está no arquivo *ipv6_utils.c* e é usada para fazer a descompressão de um endereço IPv6 e serve também para verificar a validade do formato do mesmo, pois se o formato for invalido, o número IPv6 não pode ser descomprimido, um exemplo de IPv6 em formatos comprimido e descomprimido:

IPv6 em formato comprimido: 2001:12f0:4c1:6643::5
IPv6 em formato descomprimido: 2001:12f0:04c1:6643:0000:0000:0000:0005

Sz_MTA_Heuristics_Compare_IP_Network(): Está função já foi vista anteriormente na verificação básica de endereços IP e de rede, ela é usada para verificar se um número IP pertence a uma rede com netmask em formato CIDR.

Sz_Check_Valid_IP_Number(): Está função se encontra no arquivo *antispam-check-ip-fqdn.c* e ela checa a validade dos números IPv4, ou seja, se o formato dos mesmos é valido, se eles pertencem a redes RFC-1918, etc.

Sz_Check_Valid_IPv6_Number(): Está função se encontra no arquivo *antispam-check-ip-fqdn.c* e é equivalente à anterior, mas apenas para números IPv6. Ela checa por endereços não-especificados ou de *loopback*, endereços *multicast*, endereços *link-local unicast* e *site local unicast*.

search_and_compare_mx_rr(): Está função se encontra no arquivo *resolv_utils.c* e ela é usada para buscar e comparar os MX de um domínio passado no comando HELO, basicamente para verificar se os MX apontados não são os MX das rede que recebe o E-mail, se os MX não apontam para IPs inválidos, para IPs internos à rede que recebe o E-mail, entradas MX invalidas, etc.

***OBSERVAÇAO*:** Apesar de parecer ser uma função simples de ser implementada, a função *search_and_compare_mx_rr()*, assim como outras que internamente chamam a função *res_search()*, representou um grande desafio

para mim, pois não se pode utilizar o método convencionalmente usado para resolver nomes, utilizando a função C *gethostbyaddr()*, pois essa função retorna apenas um dos FQDNs registrados a cada chamada da mesma, e não necessariamente virá o registro que por ventura esperaríamos, e isso é contra-produtivo de ser feito. Portanto, tive que utilizar as funções de mais baixo nível para a resolução DNS em C, o que significou ter que ler as especificações das RFC-1034 [36] e RFC-1035 [37] para ter idéia do tamanho dos cabeçalhos e dos campos de uma resposta de resolução DNS com a função *res_search()* para ter idéia de como poderia extrair os apontamentos de MX utilizando apenas uma chamada de função. Além disso, tive que pesquisar em códigos abertos publicados na Internet, aplicações que fizessem algo semelhante para que eu pudesse ter uma idéia de como fazer esse trabalho, que não foi nada trivial por deve-se fazer *parsing* e descompressão dentro da informação retornada pela função *res_search()*.

4.4.5 - Checagem de endereços no comando MAIL FROM

Esta regra tem por objetivo verificar o uso de técnicas *spammers* usadas freqüentemente para mascarar a o endereço de origem da mensagem que é passado durante a fase de envelopamento da mensagem no servidor SMTP de destino da mesma. Esta regra, por ser muito extensa e complexa, é subdividida em cinco sub-regras de checagem, com destaque para a mais famosa delas que é a verificação por *Sender Policy Framework* (SPF), sendo que cada uma delas foi implementada separadamente e depois adicionada ao conjunto de checagem de endereços no comando MAIL FROM.

Especificação: A especificação desta regra é vista com detalhes na especificação das cinco sub-regras.

Implementação: Basicamente esta regra foi implementada como uma chamada à função *Sz_MTA_Heuristics_Check_Mail_From()* que, quando a sub-regra é simples, ela mesma faz a verificação e retorna o resultado, quando a sub-regra é complexa, chama uma função auxiliar para realizar a checagem. Neste último caso, apenas a regra de *Sender Policy Framework* (SPF) necessitou ser checada separadamente por função auxiliar. Além dos argumentos normalmente

passados as funções da LibAntispam, já vistos anteriormente, a função *Sz_MTA_Heuristics_Check_Mail_From()* recebe ainda o argumento passado no comando HELO para o caso do mesmo ser necessário durante a checagem por SPF. O retorno da função *Sz_MTA_Heuristics_Check_Mail_From()* é o tipo estruturado *v_mail_from_t*, descrito no fragmento de código do arquivo *mta-heuristics.h*:

```
/**
 * \struct v_mail_from_s
 * \brief Store the results for a mail from ckeck.
 * The structure for antispam heuristics mta mail from check result.
 * \sa Sz_MTA_Heuristics_Check_Mail_From()
 */
typedef struct v_mail_from_s
{

    /** MAIL FROM ckeck return code. */
    v_mail_from_return_t    return_value;

    /** Pointer to a prohibited character found. */
    char                *prohibited_char;

    /** An unwanted character found in begin or end of mail address. */
    char                unwanted_char;

    /** Pointer to a bad MAIL FROM found. */
    char                *bad_mail_from_domain;

    /** Pointer to SPF explanation, if it exists. */
    char                *spf_exp;

} v_mail_from_t;
```

Para uma melhor clareza dos códigos de retorno de checagem retornado no campo *return_value*, é listado o tipo estruturado *v_mail_from_return_t* extraído do fragmento de código do arquivo *mta-heuristics.h*:

```
/**
 * \typedef v_mail_from_return_t
 * \brief Antispam heuristics v_mail_from_return_t type definition.
 * Antispam heuristics return values for MAIL FROM command check in Sz_MTA_Heuristics_Check_Mail_From().
 * \note V_MAIL_FROM_LOCAL_BUT_CLIENT_NOT_LOCAL is deprecated, use the V_MAIL_FROM_MATCH_MY_DOMAINS instead.
 */
typedef enum
{
    V_MAIL_FROM_ERROR                                   = 0,
    V_MAIL_FROM_OK                                      = 1,
    V_MAIL_FROM_CHECK_IP_ADMIN_REJECT_IP_IN_ADDRESS     = 2,
    V_MAIL_FROM_CHECK_IP_NOT_IN_LITERAL_FORMAT          = 3,
    V_MAIL_FROM_CHECK_IP_FORMAT_INVALID                 = 4,
    V_MAIL_FROM_CHECK_IP_INVALID_IPV4                   = 5,
    V_MAIL_FROM_CHECK_IP_INVALID_IPV4_RFC1918           = 6,
    V_MAIL_FROM_CHECK_IP_INVALID_IPV6                   = 7,
    V_MAIL_FROM_CHECK_IP_IPV6_ADDR_IN_IPV4_CONNECTION   = 8,
    V_MAIL_FROM_CHECK_IP_IPV4_ADDR_IN_IPV6_CONNECTION   = 9,
    V_MAIL_FROM_CHECK_IP_NOT_RESOLVE_TO_FQDN            = 10,
    V_MAIL_FROM_NULL_REVERSE_PATH_NOT_LOCAL             = 11,
    V_MAIL_FROM_LOCAL_BUT_CLIENT_NOT_LOCAL              = 12,
    V_MAIL_FROM_MATCH_MY_DOMAINS                        = 12,
    V_MAIL_FROM_RESOLVE_INVALID_IPV4                    = 13,
    V_MAIL_FROM_RESOLVE_INVALID_IPV6                    = 14,
    V_MAIL_FROM_RESOLVE_MATCH_MY_IP_NETWORKS            = 15,
```

```
        V_MAIL_FROM_MX_MATCH_MY_DOMAINS                 = 16,
        V_MAIL_FROM_RESOLVE_MX_INVALID_IPV4             = 17,
        V_MAIL_FROM_RESOLVE_MX_INVALID_IPV4_RFC1918     = 18,
        V_MAIL_FROM_RESOLVE_MX_INVALID_IPV6             = 19,
        V_MAIL_FROM_MX_RESOLVE_MATCH_MY_IP_NETWORKS
    = 20,
        V_MAIL_FROM_DNS_TEMP_FAIL                       = 21,
        V_MAIL_FROM_NOT_RESOLVE_AND_NOT_HAS_MX          = 22,
  V_MAIL_FROM_RESOLVE_INVALID_IPV4_RFC1918_AND_NOT_HAS_MX
= 23,
        V_MAIL_FROM_BAD_MX_RECORD                       = 24,
        V_MAIL_FROM_HAS_PROHIBITED_CHARS                = 25,
        V_MAIL_FROM_HAS_BAD_ADDRESS                     = 26,
    V_MAIL_FROM_HAS_UNWANTED_CHAR_IN_BEGIN_OR_END_USEN
    AME  = 27,
        V_MAIL_FROM_SPF_NONE                            = 28,
        V_MAIL_FROM_SPF_PASS                            = 29,
        V_MAIL_FROM_SPF_FAIL                            = 30,
        V_MAIL_FROM_SPF_SOFTFAIL                        = 31,
        V_MAIL_FROM_SPF_NEUTRAL                         = 32,
        V_MAIL_FROM_SPF_TEMPERROR                       = 33,
        V_MAIL_FROM_SPF_PERMERROR                       = 34
    } v_mail_from_return_t;
```

É importante salientar que a função *Sz_MTA_Heuristics_Check_Mail_From()*, alem de servir de *front-end* para as sub-regras, permeia as informações passadas em seus argumentos para remover particularidades do protocolo SMTP que, apesar de pouco usadas podem atrapalhar as verificações caso venha a ser utilizadas, tais como remoção de informação de *host-relays*, *tags* especiais previstas no protocolo, conversão de IPs literais em FQDNs, etc.

4.4.5.1 - **Restrição ao uso de null-reverse path**

O *null-reverse path*, que no comando MAIL FROM é passado como o endereço <> (sinal de *menor que* seguido do sinal de *maior que*), foi definido originalmente na RFC-821 para ser usado em casos raros em que se necessita evitar a ocorrência interna de erros que possam causar *mail-loop*. O *mail-loop* é uma situação que ocorre quando um erro retorna ao endereço remetente de uma mensagem, porém, por algum motivo técnico, o remetente da mensagem é o próprio endereço destinatário da mesma. Assim gera-se uma situação de *loop* infinito que causa uma rápida degradação do desempenho do MTA particularmente e do sistema como todo no geral. Atualmente, os MTAs já possuem mecanismos eficientes que evitam o *mail-loop* e os servidores SMTP já vêm pré-configurados com endereços especiais como *MAILER-DAEMON* e *postmaster* para evitar tais situações. Infelizmente, por alguma má interpretação da RFC-821, os MTAs são programados para aceitarem endereços *null-reverse path* vindos de clientes, locais ou não, externos ao servidor SMTP, o que abre uma porta valiosa para o envio de SPAM, pois o *null-reverse path* não retorna erro para o remetente e o SPAM na pior das hipóteses é recebido pela conta administrativa do sistema, provocando degradando do espaço em disco em muitos casos.

Especificação: Nesta sub-regra, deve-se verificar se foi passado *null-reverse path* como argumento para o comando MAIL FROM e rejeitá-lo se o mesmo não provier do próprio servidor SMTP ou de clientes locais ou autorizados.

Implementação: A implementação desta regra foi bastante simples, pois o objetivo é verificar a presença da *string* "<>"[1] no comando MAIL FROM a verificação se o cliente está autorizado ou não a usar *null-reverse path* é feito no início da função *Sz_MTA_Heuristics_Check_Mail_From()*. Os fragmentos de código extraídos do arquivo *mta-heuristics-mail-from-vrfy.c* mostram como é feita a implementação correta da regra:

Notas: [1] *Em alguns MTAs, como o Sendmail, é possível passar o null-reverse path como uma string com um único caractere, o terminador '\0', que deve ser convertido para a string "<>" antes de quaisquer verificações.*

```
    // For restrict_null_reverse_path verification, if MTA accept
    // decommented addresses
    if((Mail_From[0] == '\0') || (strlen(Mail_From) == 0))
      string_size = strlen(NULL_REV_PATH);
    else
      string_size = strlen(Mail_From);

    // String size is too long...
    if(string_size > BUFFER_SIZE)
    {
      a_err_war->antispam_warning = MTA_HCMF_MFSTL;
      string_size = BUFFER_SIZE;
    }

    sender = (char *)malloc(string_size + MALLOC_SIZE_OFFSET);
    if(sender)
    {
      // Reset alloced memory with '\0' chars
      memset(sender, 0, (string_size + MALLOC_SIZE_OFFSET));

      // For restrict_null_reverse_path verification, if MTA accept
      // decommented addresses
      if((Mail_From[0] == '\0') || (strlen(Mail_From) == 0))
      {
        strncpy(sender, NULL_REV_PATH, string_size);
      }
      else if(strstr(Mail_From, NULL_REV_PATH) != NULL)
      {
        // If commented null reverse-path
        strncpy(sender, NULL_REV_PATH, string_size);
      }
...
    }
...
```

```
      // Verify if a null reverse-path "<>" was passed in MAIL FROM cmd. If
      // yes we may reject connection because only local friends IP/Networks
      // can declare null reverse-path in MAIL FROM cmd.
      if(restrict_null_reverse_path)
      {
        // Verify if user want this check
        if(Want_User_List)
        {
          if((!(user_opt->global_user_search_return_code&V_NO_USER)) &&
(!(user_opt->global_user_search_return_code&V_ERROR)) && (!(user_opt-
>global_user_search_return_code&V_USER_RESTRICT_NULL_REVERSE_
PATH_USE)))
            goto user_skip_null_rev_path_check;
        }

        if(strstr(sender, NULL_REV_PATH) != NULL)
        {
          free(sender);
          result.return_value                                          =
V_MAIL_FROM_NULL_REVERSE_PATH_NOT_LOCAL;
          in_libantispam_checking--;
          return(result);
        }
      }
user_skip_null_rev_path_check:
```

4.4.5.2 - Restrição ao uso do domínio local

Esta sub-regra restringe o uso dos domínios e subdomínios locais apenas às máquinas da rede local e às máquinas ou redes autorizadas. Esta regra se destina a barrar o método do uso de domínios locais no comando MAIL FROM, muito utilizado pelos *spammers* para enganar os usuários locais e algumas regras antispam (SPF, por exemplo).

Alguns exemplos de endereços utilizados em SPAMs que usaram esta técnica[1]:

MAILER-DAEMON@pads.ufrj.br
majordomo-owner@pompeia.pads.ufrj.br
hbot_ymyp_q_g_z@pads.ufrj.br
xisv_ufcs_j_t_z@pads.ufrj.br
dbvi_wpgb_l_u_f@pads.ufrj.br
srvw_hbbw_v_x_h@pads.ufrj.br

A desvantagem do uso deste tipo de regra consiste no fato que o usuário da rede local, caso esteja usando o seu ISP de casa ou uma outra rede, não poderá declarar o seu endereço local no campo *From:* de quaisquer mensagens que ele tente enviar para outros endereços no domínio local (geralmente, os MUA usam a informação do campo *From:* para criar o argumento do comando MAIL FROM). Porem, isto é um caso em que se deve orientar o usuário a mudar seus hábitos, dado que a quantidade de SPAMS recebidos com o uso de endereços locais é muito grande. Além disso, na regra de SPF este mesmo tipo de problema com o uso de domínios locais pode ser verificado caso a regra SPF seja muito restritiva, i.e. *-all*.

Especificação: A especificação para essa regra será: O uso de domínios e subdomínios locais será restrito apenas às máquinas da rede local ou de redes ou máquinas autorizadas para tal.

Implementação: A implementação desta regra foi bastante simples, pois o objetivo é verificar se o domínio local ou subdomínios locais estão sendo utilizados no comando MAIL FROM. A verificação se o cliente pertence a rede local ou a redes ou máquinas autorizadas é feito no início da função *Sz_MTA_Heuristics_Check_Mail_From()*. Podemos observar a rotina que faz a verificação do uso de domínios e subdomínios locais no fragmento de código abaixo, extraído do arquivo *mta-heuristics-mail-from-vrfy.c*. Notar que basta a utilização da função C strstr() para verificar a ocorrência tanto do uso do domínio local como de seus subdomínios.

Notas: [1] *Fonte: SPAMs recebidos pelo endereço szendro@pads.ufrj.br no ano de 2003.*

```
// Verify if the sender information give by remote client belongs to
// our(s) local(s) domain(s)
for(count = 0; count <= my_domains_counter; count++)
{
  // If local domain found in sender information, refuse
  // the connection because only local/friends network can
  // use our local domains!
  if(strstr(sender_domain, my_domains_stack[count]) != NULL)
  {
    free(sender_domain);
    free(sender);
    result.return_value = V_MAIL_FROM_MATCH_MY_DOMAINS;
    in_libantispam_checking--;
    return(result);
  }
}
```

4.4.5.3 - Sender Policy Framework (SPF)

Está regra foi a primeira a ter uma especificação formalizada em norma, que é regida pela RFC-4408. Ela se baseia na premissa de que a autorização para que uma ou mais máquinas possam enviar mensagens usando um domínio (declarando esse domínio no comando MAIL FROM, ou seja, MAIL FROM:<fulano@domínio.alguma.coisa>) deve ser da responsabilidade do detentor do domínio em questão.

Especificação: A especificação completa para esta regra pode ser obtida na RFC-4408. Resumindo a especificação: A forma de se passar a informação sobre como deve ser tratado o uso do domínio é feita realizando-se uma resolução DNS para o domínio usado no comando MAIL FROM, ou no comando HELO na falta de domínio naquele, buscando-se uma entrada SPF ou TXT para esse domínio e verificando as regras SPF contidas na *string* retornada na resolução. As regras SPF apresentam uma *string* inicial identificadora da entrada SPF e da versão, “v=spf1”, seguida de uma série de segmentos (*tokens*)

que contêm as diretivas de uso (mecanismos e modificadores) do domínio: *ip4*, *ip6*, *a*, *mx*, *all*, etc; e seus qualificadores: **+**, **-**, **~** e **?**. Como a RFC-4408 é muito extensa, não a detalharei aqui, porém recomendo um leitura da mesma para um melhor entendimento da implementação feita para a LibAntispam.

Implementação: A implementação do suporte a SPF na LibAntispam foi feito seguindo-se a RFC-4408. Primeiramente foram especificados os limites, códigos de retorno, a estrutura de retorno das verificações SPF e a estrutura para os mecanismos e modificadores SPF, que abaixo são mostrados no segmento de código do arquivo *spf.h*:

```
/*
 * The RFC-4408 recommends until 10 counts for the same mechanism, but
 * I found some guys that are using more than 10 counts (i.e. eleven
 * ip4 mechanisms).
 * I will limit in 20 counts for the same mechanism and I don't plan to
 * increase this limit!
 */
#define MAX_SAME_MECHANISM      20

// Qualifiers for SPF mechanisms
typedef enum
{
      FAIL              = 0,
      SOFTFAIL          = 1,
      PASS              = 2,
      NEUTRAL           = 3
} a_spf_qualifiers_t;

// Qualifiers and errors in return code format
// The return values are potency of 2 and caller SHOULD compare return values
// using AND (&) operation.
#define SPF_Q_FAIL                0x00000001
#define SPF_Q_SOFTFAIL            0x00000002
#define SPF_Q_PASS                0x00000004
```

```
#define SPF_Q_NEUTRAL           0x00000008
#define SPF_Q_NONE              0x00000010
#define SPF_Q_PERMERROR         0x00000020
#define SPF_Q_TEMPERROR         0x00000040

// Structure for SPF query return
typedef struct a_spf_query_return_s
{
        int spf_q_return;
        char *exp;
} a_spf_query_return_t;

// Structure for SPF mechanisms and modifiers
typedef struct a_spf_s
{
        /*
         * Internals variables for this structure (not part of real SPF
         * mechanisms or modifiers)
         */

        // Sender
        char *sender;

        // HELO/EHLO
        char *helo;

        // Remote Client IP number (IPv4 or IPv6)
        // The IP number of remote host that connect our SMTP server
        char *remote_client_ip;

        // The IP numbers type (IPv4 or IPv6)
        a_boolean_t ipv6_connection;
```

```
// Includes entries counter
uint recursive_include_counter;

// Redirects entries counter
uint recursive_redirect_counter;

// An index to save the actual index counter in spf_include() function
int include_ptr_index;

/*
 * Real SPF mechanisms or modifiers (including the version number)
 */

// Version
uint version;

// Mechanisms
char *all;
a_spf_qualifiers_t all_qlf;

char *a[MAX_SAME_MECHANISM];
a_spf_qualifiers_t a_qlf[MAX_SAME_MECHANISM];
uint a_counter;

char *mx[MAX_SAME_MECHANISM];
a_spf_qualifiers_t mx_qlf[MAX_SAME_MECHANISM];
uint mx_counter;

char *ptr[MAX_SAME_MECHANISM];
a_spf_qualifiers_t ptr_qlf[MAX_SAME_MECHANISM];
uint ptr_counter;

char *ip4[MAX_SAME_MECHANISM];
a_spf_qualifiers_t ip4_qlf[MAX_SAME_MECHANISM];
```

```
    uint ip4_counter;

    char *ip6[MAX_SAME_MECHANISM];
    a_spf_qualifiers_t ip6_qlf[MAX_SAME_MECHANISM];
    uint ip6_counter;

    char *include[MAX_SAME_MECHANISM];
    a_spf_qualifiers_t include_qlf[MAX_SAME_MECHANISM];
    uint include_counter;

    char *exists[MAX_SAME_MECHANISM];
    a_spf_qualifiers_t exists_qlf[MAX_SAME_MECHANISM];
    uint exists_counter;
    // Modifiers
    char *redirect;

    char *exp;
} a_spf_t;
```

Após isto, foram implementadas duas funções de vital importância antes da implementação das funções dos mecanismos e modificadores SPF:

spf_txt2struct(): Esta função transforma uma entrada SPF obtida por resolução DNS numa estrutura *a_spf_t*, com alocação de memória para a mesma. Isto é necessário para tornar mais dinâmica e rápida todas as verificações dos mecanismos SPF. Esta função já faz também a sinalização para todos os qualificadores encontrados na entrada SPF, junto a seus mecanismos.

spf_dealloc_struct(): Esta função é chamada no final da verificação de todos os mecanismos da entrada SPF, ela faz a desalocação de memória da estrutura *a_spf_t* alocada pela função anterior.

As funções para os mecanismos SPF foram implementadas em ordem crescente de complexidade. Sempre após a implementação da função de um

mecanismo, um teste real foi realizado para checar o correto funcionamento da implementação feita. As funções para os mecanismos e modificadores SPF são:

spf_all(): Função responsável pela verificação do mecanismo “all”.

spf_a(): Função responsável pela verificação do mecanismo “a”.

spf_mx(): Função responsável pela verificação do mecanismo “mx”.

spf_ptr(): Função responsável pela verificação do mecanismo “ptr”.

spf_ip4(): Função responsável pela verificação do mecanismo “ip4”.

spf_ip6(): Função responsável pela verificação do mecanismo “ip6”.

spf_exists(): Função responsável pela verificação do mecanismo “exists”.

spf_include(): Função responsável pela verificação do mecanismo “include”. Internamente ela chama a função *do_include_spf_check()*.

spf_redirect(): Função responsável pela verificação do modificador “redirect”.

spf_exp(): Função responsável pela verificação do modificador “exp”.

do_include_spf_check(): Esta função é chamada pela função *spf_include()* é realiza uma nova *query* SPF para o domínio declarado num mecanismo “include”.

Então, ao final da implementação das funções dos mecanismos e dos modificadores, foram implementadas as duas funções principais de busca SPF:

do_complete_spf_check(): Recebe uma estrutura a_spf_t e realiza a chamada de todas as demais funções de mecanismos e modificadores. É chamada pela função *do_spf_check()*.

do_spf_check(): E a função principal para as query SPF. Ela recebe como argumento os valores dos comandos MAIL FROM e HELO, além do número IP do cliente remoto, faz a verificação de consistência dos mesmos, faz a resolução da entrada SPF do domínio, chama as funções *spf_macro_expansion(), spf_txt2struct(), do_complete_spf_check()* e *spf_dealloc_struct()* e retorna o código de retorno para a verificação SPF.

A RFC-4408 ainda prevê a possibilidade de uso de macros nas *strings* SPF. Essa funcionalidade foi adicionada posteriormente ao suporte à SPF da LibAntispam num arquivo à parte denominado *spf_macro.c*. Há apenas uma função para a expansão das macros das entradas SPF.

spf_macro_expansion(): É a função que modifica a entrada SPF retornada na *query* DNS, de forma a expandir todas as macros que ela tiver, realocando memória para a *string*, se necessário.

***Comentários sobre as implementações das funções*:** Algumas funções apresentaram alguns complicadores para serem implementadas. Por exemplo, as entradas A, MX e PTR, que podem ter múltipas entradas.

Para o caso da entrada A não tem problema, pois o mecanismo "a" requer apenas que exista uma entrada no DNS. Porém para as entradas MX e PTR a dificuldade foi maior, pois elas podem ter múltiplas entradas e todas elas devem ser alvo de verificação. Para elas, foram criadas as funções internas *compare_ip_mx_rr()* e *compare_ptr_rr()* no arquivo *resolv_utils.c*, além de funções internas especiais para o mecanismo "ptr", estes mecanismos são *create_in_addr_arpa_ipv4_query_entry()* que está no arquivo *ip_utils.c* e *create_in_addr_arpa_ipv6_query_entry()* que está no arquivo *ipv6_utils.c*.

Como complicador para a implementação dessas funções, pouquíssima informação relevante foi extraída na Internet, pois são poucos que se dão ao trabalho de fazer esse tipo de checagem com essa exatidão, muitos aplicativos e MTAs na Internet utilizam erroneamente apenas a função *gethostbyname()* ou *gethostbyaddr()* porque partem do princípio que quase ninguém usa mais do que uma entrada PTR por FQDN, por exemplo, o que é uma premissa errônea. A abordagem usada para a implementação dessas funções foi semelhante a usada na função *search_and_compare_mx_rr()* da verificação do comando HELO, vista anteriormente.

A chamada da função *do_spf_check()* na LibAntiaspam é feita pelo *front-end* da mesma no arquivo *mta-heuristics-mail-from-vrfy.c*, onde ocorre uma depuração da informação a ser passada para a checagem SPF, a função *front-end* chamasse *Sz_MTA_Heuristics_SPF_Check()* e ela é chamada internamente pela função *Sz_MTA_Heuristics_Check_Mail_From()* após a

checagem de uso de domínio local e antes da checagem de uso de caracteres proibidos.

4.4.5.4 - Restrição ao uso de caracteres proibidos

Esta regra se baseia numa particularidade que a maioria dos *robots spammers* possuem, que é a de criar *usernames* aleatórios falsos de usuários utilizando todo o tipo de caractere ASCII de forma a criar *usernames* ininteligíveis em sua grande maioria. Alguns exemplos de *usernames* de endereços já criados por *robots spammers*[1]:

d_factory_@hotmail.com
muchachitas__2003@yahoo.com
m_kmourw@hotmail.com
CERTIFICADO_ISO@ZIPMAIL.COM.BR
no_repply@hotmail.com
uyu_ioy@yahoo.com.br
visite_site_gallastelecom.com.br@agius.com.br
cobran*a@embratel.com.br
owner-nolist-seg64267*szendro**pads*-ufrj*-br@ntls.digitalriver.com

Nota-se que nos exemplos anteriores, foram mostrados os endereços aleatórios que contem o caractere *undescore* (sublinhado). A maioria dos endereços aleatórios não contém caracteres do tipo _, #, !, *, etc, porém os que apresentam tais caracteres utilizam em sua maioria o *undescore* porque muitos usuários gostam de colocar esse caractere no seu *username*. Logo, este é o maior problema desta técnica quando se relaciona o caractere *undescore* na lista de caracteres proibidos, pois poderão ocorrer ocasionalmente *falsos-positivos*. A melhor solução neste caso é utilizar uma lista de endereços autênticos para *by-passar* esta checagem caso esses endereços contenham o caractere *undescore*.

Notas: [1] *Fonte: SPAMs recebidos pelo endereço szendro@pads.ufrj.br no ano de 2003.*

Especificação: A especificação para essa regra será: Rejeitar o uso de caracteres proibidos no comando MAIL FROM. Os caracteres proibidos *default* do sistema, no arquivo de configuração *mta-antispam.conf* são: | , \ , _ , ~ , ` , ! , # , $, % , ^ , & , * , (,) , { , } , [,] , " , ' , ; , : , ? , /

Os caracteres '**.**' , '**-**' , '**@**' , '**<**' , '**>**' , '**+**' e '**,**' nunca poderão ser aceitos como caracteres proibidos e serão rejeitados pela LibAntispam no momento do carregamento da lista. Demais caracteres não listados podem ou não serem adicionados a lista a critério do administrador ou do usuário, bem como caracteres *default* da lista podem ser removidos. É recomendado bom senso na elaboração de listas de caracteres proibidos!

Implementação: A implementação desta regra foi simples, porém trabalhosa, pois devemos considerar que a listagem de caracteres proibidos fornecida pelos usuários devem se sobrepor a listagem geral do administrador do sistema, isto foi feito para que a particularização da regra para um único usuário não afete os demais. O fragmento de código extraído do arquivo *mta-heuristics-mail-from-vrfy.c* demonstra como foi implementada esta verificação.

```
// Now we will verify for prohibited characters in sender information

if(restrict_prohibited_chars)
{
  // Verify if user want this check
  if(Want_User_List)
  {
    if((!(user_opt->global_user_search_return_code&V_NO_USER)) && (!(user_opt->global_user_search_return_code&V_ERROR)) && (!(user_opt->global_user_search_return_code&V_USER_RESTRICT_PROHIBITED_CHARS_USE)))
      goto user_skip_prohibited_chars_check;
  }

  // If machine was found in our accepted hosts lists, skip
  // prohibited hosts test
```

```
if(Sz_MTA_Heuristics_Search_In_Accepted_Hosts_List(RFC2821_Re
mote_Hostname_IP_Information, a_err_war))
  goto skip_prohibited_chars_check;

if(Want_User_List)
{
  if((!(user_opt->global_user_search_return_code&V_NO_USER))  &&
(!(user_opt->global_user_search_return_code&V_ERROR)))
  {
    if(user_opt->global_user_search_type == NORMAL_USER)
    {
      if(memory_words_control_users_stack[user_opt-
>global_user_search_position].user_prohibited_chars_counter != -1)
      {
        for(count = 0; count <=
memory_words_control_users_stack[user_opt-
>global_user_search_position].user_prohibited_chars_counter; count++)
        {
          // If found any user's prohibited characters...
          if(strstr(sender,  memory_words_control_users_stack[user_opt-
>global_user_search_position].user_prohibited_chars_stack[count]) != NULL)
          {
            free(sender);
            // Set prohibited_char to point to prohibited character
            result.prohibited_char =
memory_words_control_users_stack[user_opt-
>global_user_search_position].user_prohibited_chars_stack[count];
            result.return_value =
V_MAIL_FROM_HAS_PROHIBITED_CHARS;
            in_libantispam_checking--;
            return(result);
          }
        }
```

```
                // Skip system admin's prohibited characters check
                goto skip_prohibited_chars_check;
              }
            }
            else if(user_opt->global_user_search_type == DEFAULT_USER)
            {
              if(default_users_control_stack[user_opt-
>global_user_search_position].user_prohibited_chars_counter != -1)
              {
                for(count = 0; count <= default_users_control_stack[user_opt-
>global_user_search_position].user_prohibited_chars_counter; count++)
                {
                  // If found any user's prohibited characters...
                  if(strstr(sender, default_users_control_stack[user_opt-
>global_user_search_position].user_prohibited_chars_stack[count]) != NULL)
                  {
                    free(sender);
                    // Set prohibited_char to point to prohibited character
                    result.prohibited_char =
default_users_control_stack[user_opt-
>global_user_search_position].user_prohibited_chars_stack[count];
                    result.return_value =
V_MAIL_FROM_HAS_PROHIBITED_CHARS;
                    in_libantispam_checking--;
                    return(result);
                  }
                }
                // Skip system admin's prohibited characters check
                goto skip_prohibited_chars_check;
              }
            }
          }
        }
```

```
        for(count = 0; count <= prohibited_chars_counter; count++)
        {
          // If found any prohibited characters...
          if(strstr(sender, prohibited_chars_stack[count]) != NULL)
          {
            free(sender);

            // Set prohibited_char to point to prohibited character
            result.prohibited_char = prohibited_chars_stack[count];
            result.return_value                                    =
V_MAIL_FROM_HAS_PROHIBITED_CHARS;
            in_libantispam_checking--;
            return(result);
          }
        }
      }

user_skip_prohibited_chars_check:
skip_prohibited_chars_check:
```

4.4.5.5 - Checagem de maus endereços passados no comando MAIL FROM

Esta regra é usada para rejeitar o envio de mensagens caso seja declarado no comando MAIL FROM algum endereço que esteja listado no arquivo *mail-addresses_or_domains.bad*. Esta regra é muito útil quando SPAMs estão sendo enviados de múltiplos servidores e estes estejam usando sempre o mesmo endereço no comando MAIL FROM. Por exemplo, um endereço muito usado para enviar SPAMs no Brasil é *ocarteiro@ocorreio.com.br*.

Especificação: A especificação desta regra será: Rejeitam-se todos os endereços e domínios listados no arquivo *mail-addresses_or_domains.bad*, podendo-se ainda utilizar o comando especial *cuwcbe()* para rejeitar caracteres estranhos no *início* ou no *final* do *username* do endereço, i.e. <_kdjfhh@xxx.x>

ou <mail-@someisp.faa>, isto é uma alternativa ao uso da checagem de caracteres proibidos, pois muitos SPAMs vem com caracteres estranhos ou de pouco uso logo no inicio ou no final do *username* do endereço.

Implementação: A implementação desta regra foi bastante simples, pois ela se baseia num simples checagem de strings, como podemos ver no fragmento de código extraído do arquivo *mta-heuristics-mail-from-vrfy.c*, também vemos como a implementação da checagem de comando especial foi simples pois tão somente é feita uma chamada a função *Sz_MTA_Heuristics_Execute_Special_Command()* que foi implementada no arquivo *mta-heuristics-internals.c*:

```
for(count = 0; count <= words_bad_mail_from_addresses_counter; count++)
{
  // A special command execution here!
  if(memory_words_bad_mail_from_addresses_stack[count][0] == '!')
  {
    // Set the default value
    special_command_return.return_value = A_FALSE;
    special_command_return.uw_char = 0;
    special_command_return = Sz_MTA_Heuristics_Execute_Special_Command(CHECK_MAIL_FROM, memory_words_bad_mail_from_addresses_stack[count], NULL, NULL, sender, a_err_war);

    if(special_command_return.return_value)
    {
      free(sender);

      // Set bad mail_from unwanted character
      result.unwanted_char = *special_command_return.uw_char;
      result.return_value = V_MAIL_FROM_HAS_UNWANTED_CHAR_IN_BEGIN_OR_END_USER NAME;
```

```
            in_libantispam_checking--;
            return(result);
          }

          continue;
        }

        // If found any bad mail_from, i.e., user@, user@FQDN or @FQDN...
        if(strstr(sender,
memory_words_bad_mail_from_addresses_stack[count]) != NULL)
        {
          free(sender);

          // Set bad mail from ptr to point to bad mail from
          result.bad_mail_from_domain                                    =
memory_words_bad_mail_from_addresses_stack[count];
          result.return_value = V_MAIL_FROM_HAS_BAD_ADDRESS;
          in_libantispam_checking--;
          return(result);
        }
```

4.4.6 - Checagem de cliente remoto usando DNSBLs

Esta regra se baseia na consulta de servidores DNS de entidades conhecidas pelo seu empenho no combate às ações *spammers* em âmbito global da Internet, i.e. MAPS RBL, SPAMHAUS, etc.

Especificação: A especificação desta regra segue a risca a técnica de consulta aos DNSBLs já vista no item 3.3 deste trabalho.

Implementação: A implementação da checagem usando DNSBLs da LibAntispam foi feita partindo do principio que poderão ser consultados vários servidores DNSBLs para garantir uma maior segurança. A função que faz essa checagem chamasse *Sz_MTA_Heuristics_BlackList()* que se encontra no arquivo *mta-heuristics-blacklist.c*, essa função é um *front-end* para a função

Sz_Check_FQDN_or_IPv6_in_Multiples_RSBL() e para a função *Sz_Check_FQDN_or_IP_in_Multiples_RSBL()*, ambas que se encontram no arquivo *antispam-check-dnsbl.c*. A função *Sz_MTA_Heuristics_BlackList()* retorna o tipo estruturado *multi_dnsbl_t*, que pode ser visto no fragmento de código do arquivo *mta-heuristics.h*:

```
/**
 * \struct multi_dnsbl_s
 * \brief Store the results for a multi-dnsbl search.
 * The structure for antispam heuristics mta multi_dnsbl search result.
 * \sa Sz_MTA_Heuristics_BlackList()
 */
typedef struct multi_dnsbl_s
{
	/** Result of search
	 *  \sa dnsbl_r_t
	 */
	dnsbl_r_t	result;

	/** message from DNSBL IN TXT entry */
	char		*message;

	/** message from DNSBL IN TXT entry to write in MTA logs */
	char		*mta_log_message;

} multi_dnsbl_t;
```

Os possíveis códigos de retorno para uma busca DNSBLs são listados a seguir no fragmento do arquivo *mta-heuristics.h*:

```
/**
 * \typedef dnsbl_r_t
 * \brief Antispam heuristics dnsbl result type definition.
 */
```

```
typedef enum
{
    DNSBL_NOT          = 0,
    DNSBL_IN           = 1,
    DNSBL_TEMP_FAIL    = 2,
    DNSBL_ERROR        = 3
} dnsbl_r_t;
```

Este tipo estruturado retorna o resultado das buscas nos servidores DNSBLs, em caso de um resultado positivo, pode ser retornada também uma mensagem explicativa do DNSBL, que resultou numa busca positiva, sobre o motivo do cliente remoto estar indexado naquela lista. É retornada também uma mensagem padronizada que poderá ser adicionada nos logs do MTA que solicitou a pesquisa nos DNSBLs.

Internamente dentro do arquivo *antispam-check-dnsbl.c*, as funções de checagem de múltiplos DNSBLs chamam as funções *Sz_Check_FQDN_or_IPv6_in_RSBL()* e *Sz_Check_FQDN_or_IP_in_RSBL()*, respectivamente. Essas funções fazem buscas individualizadas para por DNSBL passado em seus argumentos. Como a checagem de DNSBLs é uma simples consulta a DNS diretos, essas funções utilizam as funções de resolução DNS padrão da linguagem C, ou seja, a função *gethostbyname2()* para resoluções de endereços IPv6 e *gethostbyname()* para resoluções de endereços IPv4.

Para que um cliente remoto seja considerado listado num DNSBL, o servidor DNS do mesmo deve retornar um IP da classe A 127.0.0.0/8, com exceção do IP 127.0.0.1, no caso de uma consulta IPv4 ou então um IP do prefixo 0:0:0:0:0:0:0:x, com exceção do IP 0:0:0:0:0:0:0:1, no caso de IPv6.

Opcionalmente, mas ainda em testes e não habilitado atualmente para uso pelos administradores de rede, a função *Sz_MTA_Heuristics_BlackList()* permite o retorno do código DNSBL_TEMP_FAIL, que indica um erro temporário de consulta ao DNSBL, e que permite que a mensagem seja temporariamente rejeitada até que a comunicação com o DNSBL seja restabelecida, evitando casos de *falsos-negativos*, caso fosse dado apenas um código de retorno neutro em seu lugar. Essa opção ainda não foi habilitada para uso porque se esta conjecturando se isso não poderia criar um excessivo *delay* na entrega das mensagens, pois erros temporários de DNS são bastante comuns atualmente.

4.4.7 - Checagem por reciprocidade mútua

Esta regra é bastante popular atualmente, apesar de poder gerar *falsos-positivos* com certa freqüência. Ela se baseia na premissa que se o cliente remoto envia uma mensagem usando um endereço, esse mesmo endereço deve também aceitar receber mensagens. O complicador neste tipo de teste é que existem servidores administrativos legais que são configurados para apenas enviar mensagens, alguns ligados às instituições publicas brasileiras, como a CAPES, fazem isso. Além disso, *robots* de domínios *spammers* podem ser programados para aceitarem conexões SMTP, apenas na parte de envelopamento, para ludibriar este teste. Na LibAntispam esta regra recebeu a denominação técnica de *Remote Domain SMTP Sender Checking* (RDSSC) e ela interage com a regra de SPF, pois esta pode desabilitá-la se o cliente remoto for autenticado pela mesma, e com a regra de GrayListing.

Especificação: A especificação desta regra foi baseada na descrição desta técnica no item 3.3 deste trabalho. Algumas modificações foram introduzidas nela, como a possibilidade de usar o *username postchecker*, que é um endereço aceito internamente pela implementação dos *patches* da LibAntispam nos MTAs, mas que não existe realmente e que por isso não é aceito para nada além da verificação no fase de envelopamento. Uma rotina de *flachback* foi feita para o *username postmaster*, usando na maioria das implementações desta regra, de forma a manter a compatibilidade com o sistema legado. Algumas melhorias, como a utilização de um sistema de *ranking* e de *punições* foi adotado para melhor gerenciar o tempo que os endereços ficam listados como endereços bons e maus, evitando ter que repetir o teste toda vez que uma nova mensagem vinda daquele endereço é enviada. A interação entre esta técnica e as técnicas de checagem por SPF e GrayListing foi também especificada para evitar testes desnecessários e para contornar problemas compatibilidades de implementações mal feitas desta regra.

Implementação: A implementação desta regra não foi trivial porque teve que se considerar uma série de fatores e premissas como, por exemplo, que esta regra realiza uma conexão SMTP que deve ter tempos de resposta consideravelmente curtos para evitar que a conexão do cliente remoto caia

devido a *timeouts* longos de resposta. Teve-se que levar-se em consideração ainda que as listagens internas de memorização de testes bem ou mal sucedidos deveriam tirar o máximo proveito das características de funcionamento dos MTAs como, por exemplo, *multithreading*.

Na LibAntispam, a função principal de chamada desta regra é *Sz_MTA_Heuristics_Check_Remote_SMTP_Sender()*, que retorna o tipo enumerativo *remote_smtp_sender_return_t*, que abaixo é mostrado no fragmento de código do arquivo *mta-heuristics.h*:

```
/**
 * \typedef remote_smtp_sender_return_t
 * \brief Antispam heuristics remote_smtp_sender_return_t type definition
 */
typedef enum
{
    REMOTE_SMTP_SENDER_DISABLED                         = 0,
    REMOTE_SMTP_SENDER_SKIP                             = 1,
    REMOTE_SMTP_SENDER_MTA_GREETINGS_FAIL               = 2,
    REMOTE_SMTP_SENDER_PERMANENT_FAIL                   = 3,
    REMOTE_SMTP_SENDER_TRANSIENT_FAIL                   = 4,
    REMOTE_SMTP_SENDER_OK                               = 5,
    REMOTE_SMTP_SENDER_SENDER_AUTH                      = 6,
REMOTE_SMTP_SENDER_CHECK_IP_ADMIN_REJECT_IP_IN_ADDRES
S       = 7,
    REMOTE_SMTP_SENDER_CHECK_IP_NOT_IN_LITERAL_FORMAT
= 8,
    REMOTE_SMTP_SENDER_CHECK_IP_FORMAT_INVALID    = 9,
    REMOTE_SMTP_SENDER_CHECK_IP_INVALID_IPV4          = 10,
    REMOTE_SMTP_SENDER_CHECK_IP_INVALID_IPV4_RFC1918
= 11,
    REMOTE_SMTP_SENDER_CHECK_IP_INVALID_IPV6          = 12,
REMOTE_SMTP_SENDER_CHECK_IP_IPV6_ADDR_IN_IPV4_CONNECT
ION                                                 =
13,
```

```
REMOTE_SMTP_SENDER_CHECK_IP_IPV4_ADDR_IN_IPV6_CONNECT
ION      = 14,
    REMOTE_SMTP_SENDER_CHECK_IP_NOT_RESOLVE_TO_FQDN
= 15
} remote_smtp_sender_return_t;
```

Deve-se dar destaque para o código de retorno REMOTE_SMTP_SENDER_SENDER_AUTH, que denota a interação desta regra com a regra de SPF, pois não faz sentido um teste de autenticação de reciprocidade mútua se o cliente remoto já foi autenticado.

Como já dito, a função *Sz_MTA_Heuristics_Check_Remote_SMTP_Sender()* faz uma listagem interna de endereços que passaram ou não pelo teste desta regra. Como é um teste de curta duração e relativamente rápido, foi adotada a abordagem de se armazenar uma pequena lista temporária em memória RAM, tanto para os endereços bem sucedidos como os endereços mal sucedidos nos testes no caso de MTAs *multithreads*, porém, também optou-se por uma lista em arquivo pois nos MTAs do tipo *Parent-Child*, a lista em memória desapareceria com o final da execução do processo filho, além disso com a listagem em arquivo o MTA teria uma memória não-volátil, de onde ele poderia recuperar essa informação caso o sistema seja reinicializado. Tanto a lista em memória RAM como a lista em arquivo possuem um espaço para até 100 endereços.

O controle de acesso a tais listas é feito por sinalizadores, sendo que no caso dos MTAs *multithreads* o acesso a lista em arquivo é feita apenas uma única vez para leitura e, todas às vezes necessárias, apenas para atualização da mesma, no caso de acesso para escrita.

Internamente, a função *Sz_MTA_Heuristics_Check_Remote_SMTP_Sender()* faz uso da função *Sz_MTA_Heuristics_Check_Remote_SMTP_Sender_Do_Vrfy()*, cuja responsabilidade é abrir uma conexão de *socket* TCP/IP, seja em IPv4 como em IPV6, se necessário, e a partir dela interpretar sua resposta e repassar um dos códigos de retorno para a função que a chamou.

Os *timeouts* do tempo de estabelecimento de conexão e do tempo de transação de comandos são, respectivamente, 20 e 15 segundos. Porém, caso seja necessário se fazer ajustes nestes valores de *timeouts*, é permitido somente ao administrador fazer alterações dentro de intervalos de tempos pré-definidos pela

implementação desta regra na LibAntispam, que segue as recomendações da RFC-5321.

4.4.8 - Checagem de cliente remoto usando GrayListing

Esta regra antispam não tem uma norma oficial que dite a forma de seu uso, porém ela foi muito bem formalizada no "*GreyListing: Whitepaper*" escrito por seu idealizador, Evan Harris [38], e deste *whitepaper* derivam todas as implementações usadas para GrayListing. Conforme descrito no item 3.3, esta regra possui alguns problemas de compatibilidade com algumas implementações e configurações de alguns poucos servidores SMTP, além de atrasar o recebimento da mensagem, porém os benefícios dela superam esses problemas pontuais. Na implementação da LibAntispam, esta regra possui interação com as regras de SPF e de RDSSC vistas anteriormente.

Especificação: A especificação desta regra é simples, quando um servidor MTA recebe uma conexão SMTP, sua primeira ação deverá ser guardar o trio *IP do Cliente Remoto, Endereço do Remetente* e o *Endereço do Destinatário*, retornando para o cliente remoto um código de erro temporário, i.e. 4.5.0, e continuar retornando esse código de erro em conexões sucessivas deste mesmo trio até que certo tempo de *delay* tenha decorrido após o qual a mensagem será aceita normalmente.

Implementação: A implementação desta regra na LibAntispam é feita pela função *Sz_MTA_Heuristics_GrayList()*, implementada dentro do arquivo *mta-heuristics-graylist.c*, juntamente com a função de limpeza e manutenção *Sz_MTA_Heuristics_GrayList_Clean()*, encarregada de manter a listagem da GrayListing em disco sempre enxuta. Na implementação dessa listagem em disco, já que o número de entradas e os tempos são relativamente muito maiores do que os empregados na regra de reciprocidade mútua e não dá para manter essa listagem em memória RAM, optou-se por aproveitar o fato que o *filesystem* de um sistema operacional é em essência um banco de dados e implementou-se um sistema de listagem em que o IP do cliente remoto é a denominação do arquivo texto que contem o endereço do remetente, o endereço do destinatário e a estampa de tempo representando a data, em segundos desde 1º de Janeiro de

1970, quando a primeira conexão foi feita. A partir daí foram definidos os tempos máximos de existência desses arquivos e das entradas dentro deles, de forma a deixar a listagem sempre enxuta o máximo possível.

A função *Sz_MTA_Heuristics_GrayList()* retorna códigos de retorno definidos no tipo enumerativo graylist_return_t, mostrado a seguir no fragmento de código do arquivo *mta-heuristics.h*:

```
/**
 * \typedef graylist_return_t
 * \brief Antispam heuristics graylist_return_t type definition.
 */
typedef enum
{
    GRAYLIST_DISABLED       = 0,
    GRAYLIST_SKIP           = 1,
    GRAYLIST_IN             = 2,
    GRAYLIST_OUT            = 3,
    GRAYLIST_SENDER_AUTH    = 4
} graylist_return_t;
```

Nos códigos de retorno acima podemos notar que o código GRAYLIST_SENDER_AUTH se refere à interatividade com a regra do SPF, pois uma vez que o cliente já tenha tido sua identidade autenticada, não é necessário se proceder uma nova verificação de autenticidade.

A interatividade com a regra de RDSSC não é tão evidente, pois está embutida dentro da função *Sz_MTA_Heuristics_GrayList()*, porém pode ser observada no fragmento de código do arquivo *mta-heuristics-graylist.c*:

```
// If in the RDSSC test we get a temporarily deferred in greetings, i.e., the idiot Yahoo antispam "technique",
// we will multiply graylist delay time by a factor of X to punish the idiot that did it!
// We will respect the upper limit of 2 * GRAYLIST_MAX_DELAY!
```

```
            if(user_opt.global_user_search_return_code&V_SENDER_RDSSC_GR
            EETINGS_FAIL)
              {
                // If graylist delay time is between 1 and 15, multiply by factor of 3
                if((local_graylist_delay >= 1) && (local_graylist_delay <= 15
        ))
                {
                  local_graylist_delay *= 3;
                }
                else
                {
                  // If graylist delay time is between 16 and 20, multiply by factor of
2.5
                  if((local_graylist_delay >= 16) && (local_graylist_delay <= 20))
                  {
                    local_graylist_delay = (int)(local_graylist_delay * 2.5);
                  }
                  else
                  {
                    // If graylist delay time is greater than 21, multiply by factor of 2
                    if(local_graylist_delay >= 21)
                      local_graylist_delay *= 2;
                  }
                }

                if(local_graylist_delay > (2 * GRAYLIST_MAX_DELAY))
                  local_graylist_delay = (2 * GRAYLIST_MAX_DELAY);
              }
```

Podemos observar que, quando do teste de RDSSC, caso ocorram problemas de verificação como, por exemplo, implementações mal-feitas do protocolo SMTP, é acionado um mecanismo de punição que aumenta o tempo de *delay* do GrayListing por um fator de até três vezes, de forma a evitar casos de *falsos-negativos* devido a

implementações em *robots spammers* que tentem ludibriar a verificação por reciprocidade mútua.

Como última observação sobre a implementação desta regra, a notação de tempo usada em segundos necessitou de um cuidado especial para evitar truncamentos que atrapalhassem o teste e resultassem em *falsos-negativos*. Para isto foi usado o tipo inteiro sem sinal de 64 bits, *uint64_t*, cuja definição formal para sistemas que não possuem esse tipo pode ser vista no fragmento do arquivo *mta-heuristics.h*, mostrado a seguir:

```
#ifdef DEFINE_UINT64_T
typedef unsigned long long int uint64_t;
#endif
```

Capítulo 5

A Interface LibAntispam-GUI

5.1 – Por que configurar usando interface gráfica?

A LibAntispam possui uma interface gráfica *WEB* que pode ser instalada e utilizada opcionalmente. Num primeiro momento podemos questionar qual seria a utilidade do uso de uma interface gráfica pelo administrador se o mesmo poderia simplesmente configurar a LibAntispam editando diretamente os arquivos de configuração no sistema. A resposta para este questionamento é que a edição através da interface *WEB* torna mais fácil e agradável a editoração das regras, tanto à nível das regras de sistema com das regras particularizadas dos usuários, simplificando a seleção das regras a serem habilitadas ou desabilitadas, facilitando a criação de perfis/contas antispam personalizada de usuários e ainda aumentando a segurança do sistema, pois o administrador não precisará logar no mesmo usando contas de privilégio elevado e não precisará instalar programas especiais de acesso remoto na máquina em que esteja logado naquele momento, como o SSH, pois bastará fazer uso de um navegador *WEB* para a editoração das regras. A LibAntispam-GUI foi implementada fazendo-se uso das linguagens de programação C e C++ e da biblioteca VBMcgi [39], que é usada no desenvolvimento de aplicativos CGI de alta performance.

5.2 – A conceituação técnica de segurança da LibAntispam-GUI

A conceituação técnica da LibAntispam-GUI prevê dois níveis de acesso para editoração das regras: O primeiro nível é do administrador do sistema, que tem acesso às regras antispam globais do sistema e às regras particularizadas de cada perfil de usuário cadastrado no sistema antispam. O segundo nível é do usuário do sistema, que tem acesso somente às regras particularizadas do seu perfil.

Para um aumento do nível de segurança, a LibAntispam-GUI foi implementada para a leitura e a edição dos arquivos de configuração da LibAntispam usando o mesmo GID de privilégio restrito, por *default* com a denominação *antispam*. A copia dos

arquivos de configuração editados pela LibAntispam-GUI é realizado por um programa *wrapper* que faz os isolamento entre o pseudo-usuário que roda o servidor HTTP, i.e. *WWW*, *nobody*, *etc*, e que executa as CGI da LibAntispam-GUI e quaisquer rotinas de mais baixo nível do sistema, evitando-se riscos de invasão por *buffer overrun* como, por exemplo, o envio de *strings* muito grandes contendo código malicioso.

5.3 – O sistema perfis da LibAntispam-GUI

O sistema de perfis do administrador e dos usuários do sistema é de conceituação simples na LibAntispam. Todos os perfis ficam armazenados dentro do diretório de páginas WWW da LibAntispam-GUI do servidor HTTP, que neste caso é o *Apache*. Cada perfil corresponde a um subdiretório ou pasta e seu conteúdo é um link simbólico para as CGIs que esse perfil tem direito de executar. Os arquivos de configuração ficam armazenadas em */etc/antispam* para o perfil do administrador do sistema e em */etc/antispam/users/<username>* para os perfis de cada usuário da interface da LibAntispam-GUI e da LibAntispam.

A autenticação de acesso a esses perfis, na versão atual da LibAntispam-GUI, 1.1.0, é feito através de *HTTP Authentication* [1] fornecido pelo próprio servidor HTTP. Parte-se do princípio (e recomenda-se fortemente) que o servidor HTTP esteja configurado com suporte a HTTPS para uma maior proteção dos dados transmitidos via rede, principalmente as senhas do administrador e dos usuários.

Notas: [1] *É intenção se modificar este procedimento futuramente de forma a utilizar um sistema de autenticação próprio baseado em cookies e sessões.*

5.4 – A interface de acesso

A interface de acesso da LibAntispam-GUI é um CGI que lista o nome dos perfis dos usuários cadastrados no sistema antispam, conforme mostra a figura 5.1:

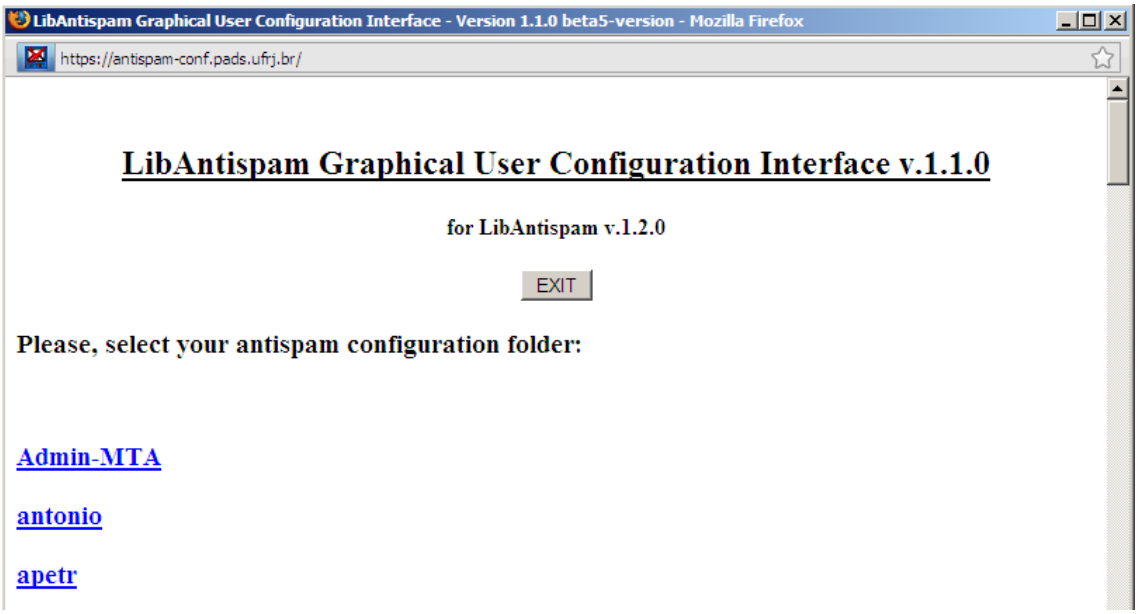

Fig. 5.1 – Interface de acesso da configuração da LibAntispam

Fonte: O autor do Trabalho

Note que o perfil Admin-MTA é criado na instalação da LibAntispam-GUI e é obrigatório, pois é esse o perfil utilizado pelo administrador para modificar as configurações do sistema.

Além do perfil Admin-MTA, podem existir um ou mais perfis de usuários, cada qual representará um diretório separado de CGIs e configurações personalizadas.

Quando é escolhido um perfil para se logar e é clicado no mesmo, aparece um *prompt* de autenticação gerado pelo servidor HTTP. Nele deve-se escrever o nome do perfil utilizado e a senha correspondente a esse perfil:

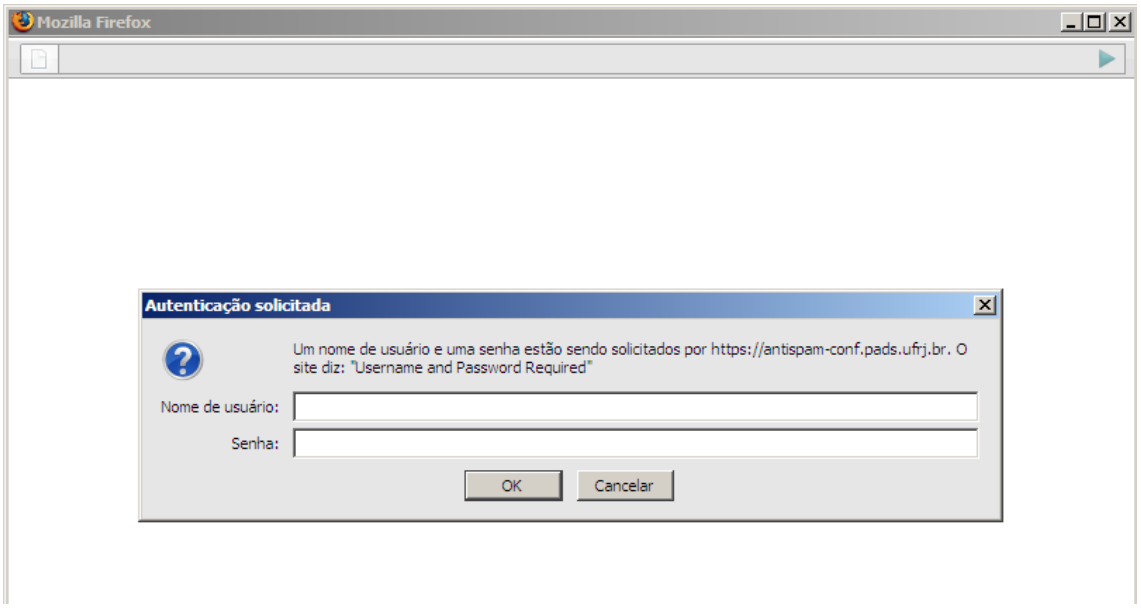

Fig. 5.2 – Interface de acesso de autenticação gerada pelo servidor HTTP

Fonte: O autor do Trabalho

5.5 – A interface de configuração do administrador

A interface de configuração do administrador pode ser logicamente dividida em três partes:

- A primeira com as configurações mais básicas e gerais que identificam o domínio e a rede em que o sistema antispam opera, as redes amigáveis que podem "*by-passar*" as regras antispam, a configuração das regras antispam, etc.
- A segunda que permite a criar e apagar perfis de usuários do sistema antispam, além de poder acessar as configurações pessoais deles.
- A terceira permite a edição das diversas listas utilizadas pela LibAntispam (*strings* proibidas, máquinas aceitáveis, endereços não tolerados, etc).

As figuras a seguir mostram os fragmentos de algumas dessas partes descritas:

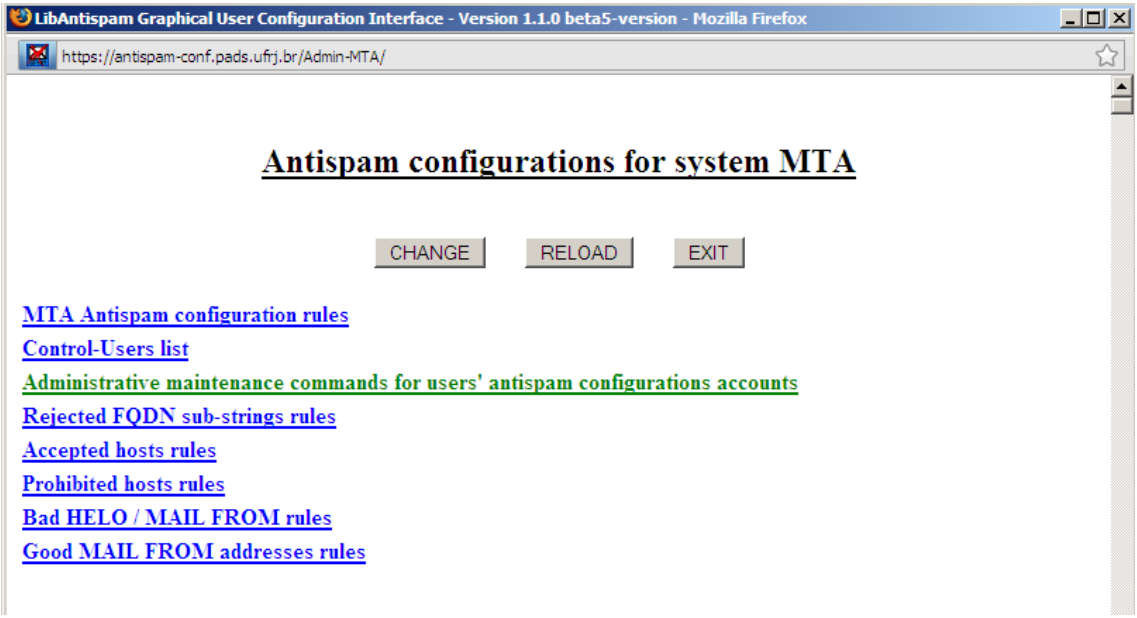

Fig. 5.3 – Interface do administrador (menu inicial)

Fonte: O autor do Trabalho

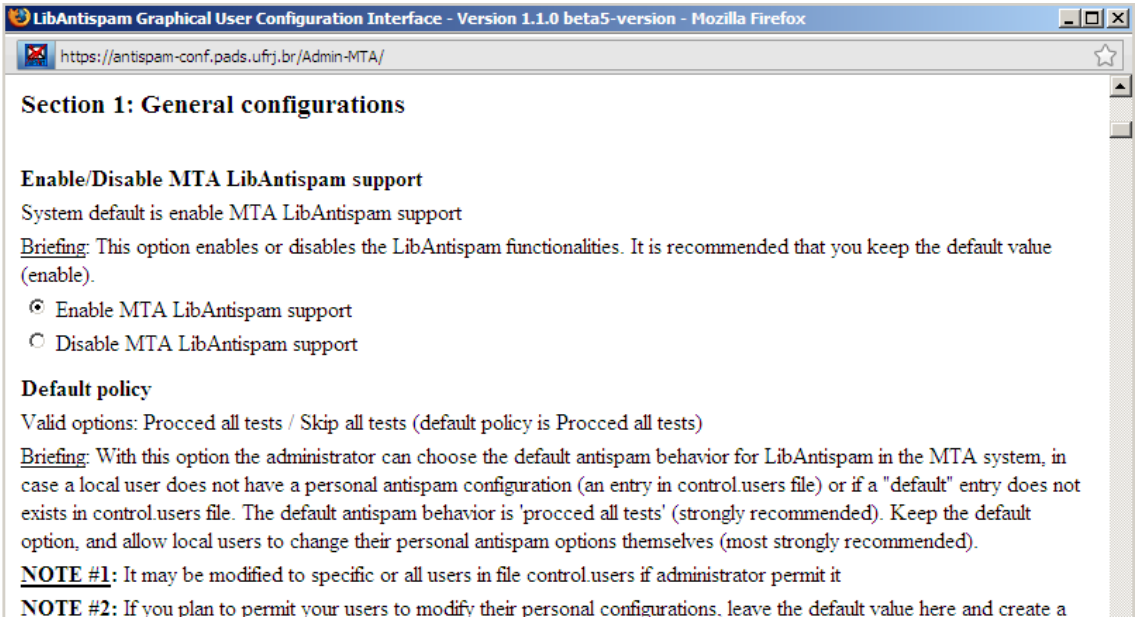

Fig. 5.4 – Interface do administrador (configuração geral do sistema)

Fonte: O autor do Trabalho

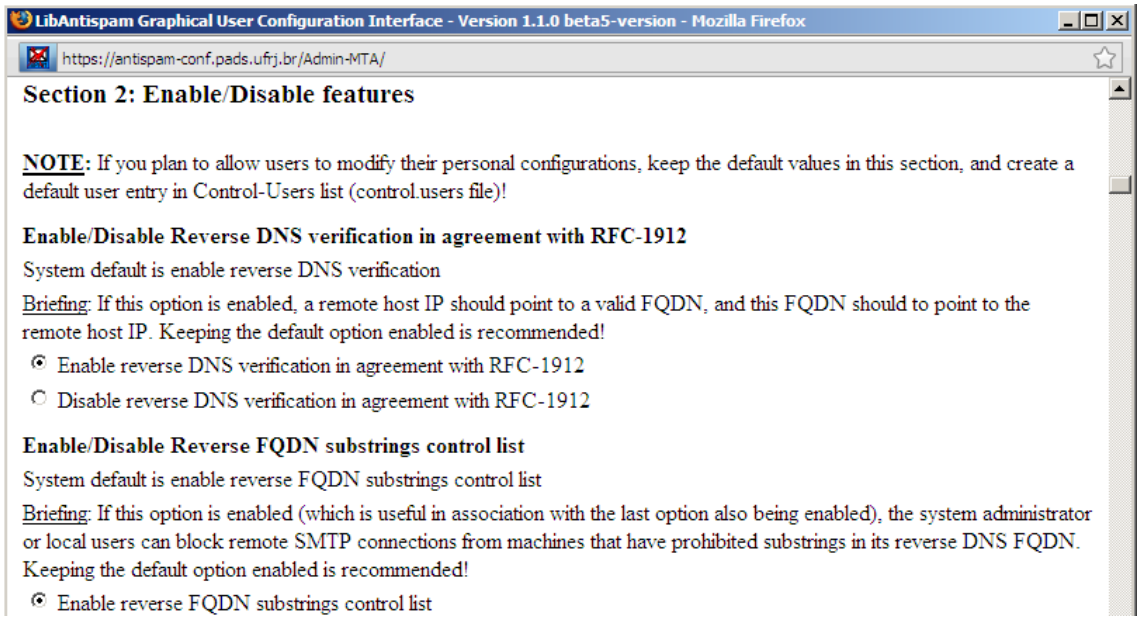

Fig. 5.5 – Interface do administrador (ativação e desativação global das regras)

Fonte: O autor do Trabalho

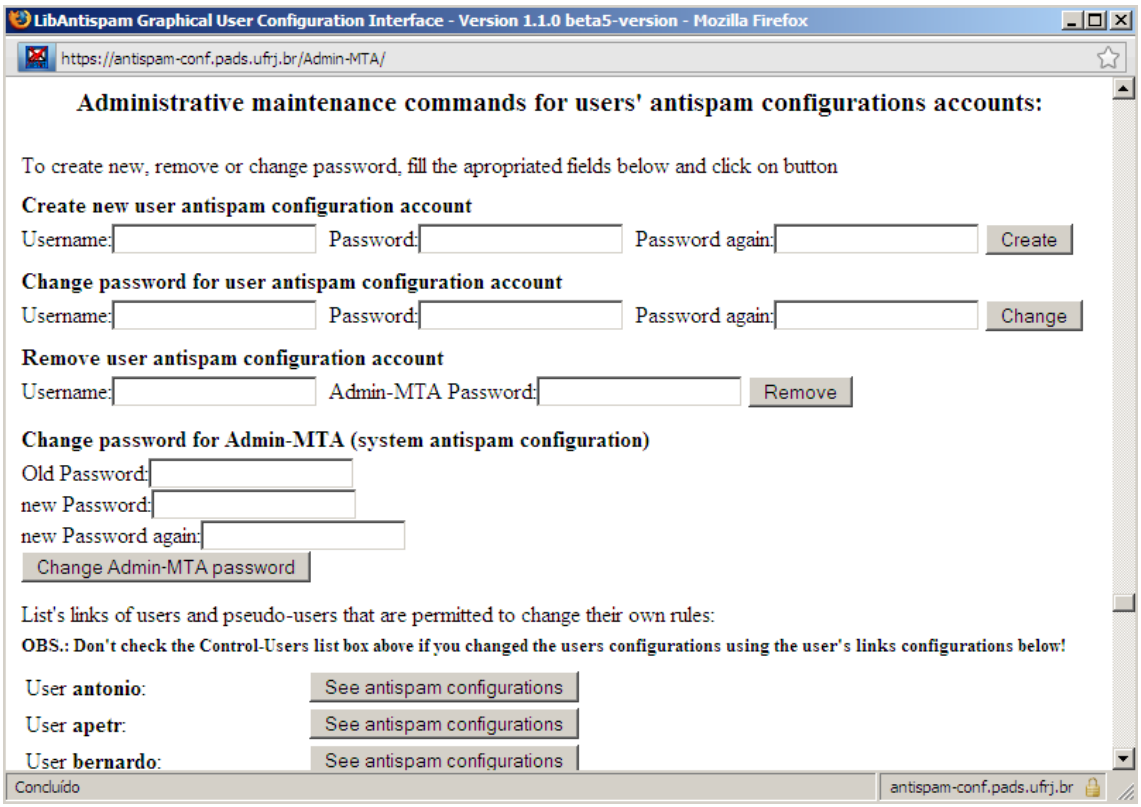

Fig. 5.6 – Interface do administrador (manutenção de perfis de usuários)

Fonte: O autor do Trabalho

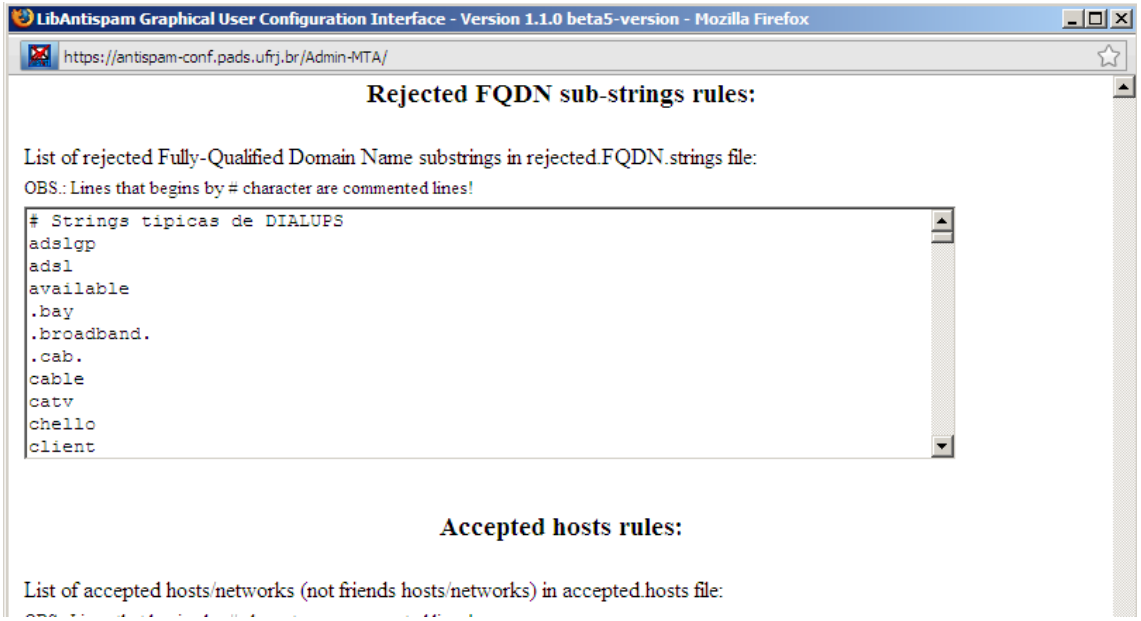

Fig. 5.7 – Interface do administrador (configuração geral das listagens)

Fonte: O autor do Trabalho

5.6 – A interface de configuração do usuário

A interface de configuração do usuário pode ser também logicamente dividida em três partes:

- A primeira parte que permite a ativação e desativação das regras antispam de forma personalizada.
- A segunda parte que corresponde às suas listas pessoais personalizadas e à mudança opcional de sua senha de acesso.
- A terceira parte corresponde a um pequeno glossário explicativo sobre como cada regra antispam funciona e as recomendações para cada uma delas.

As figuras a seguir mostram os fragmentos de algumas dessas partes descritas:

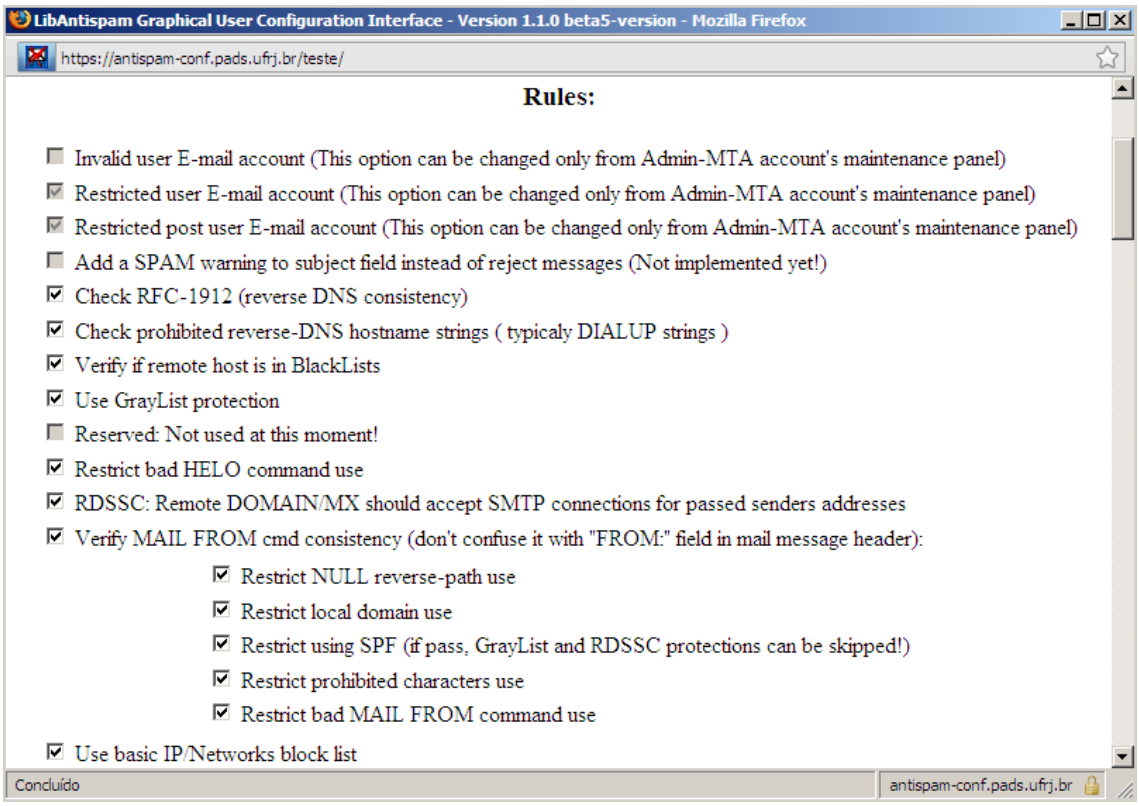

Fig. 5.8 – Interface do usuário (ativação e desativação de regras)

Fonte: O autor do Trabalho

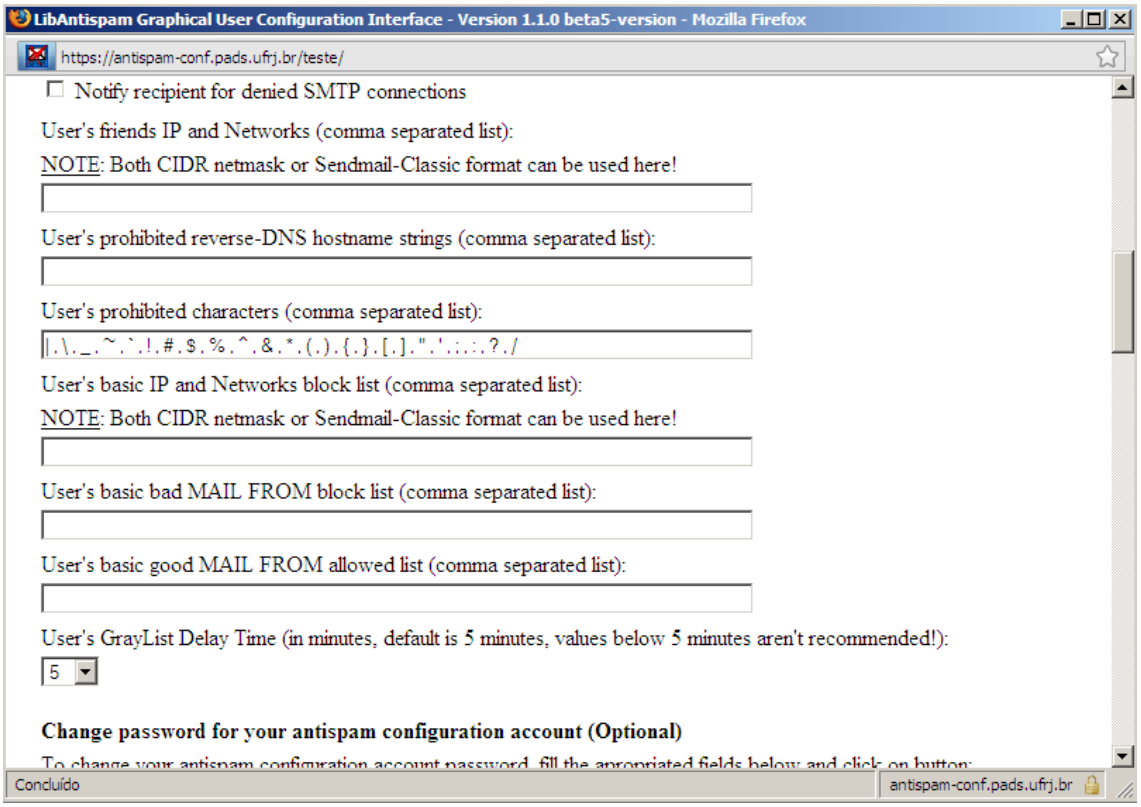

Fig. 5.9 – Interface do usuário (configuração geral das listagens)

Fonte: O autor do Trabalho

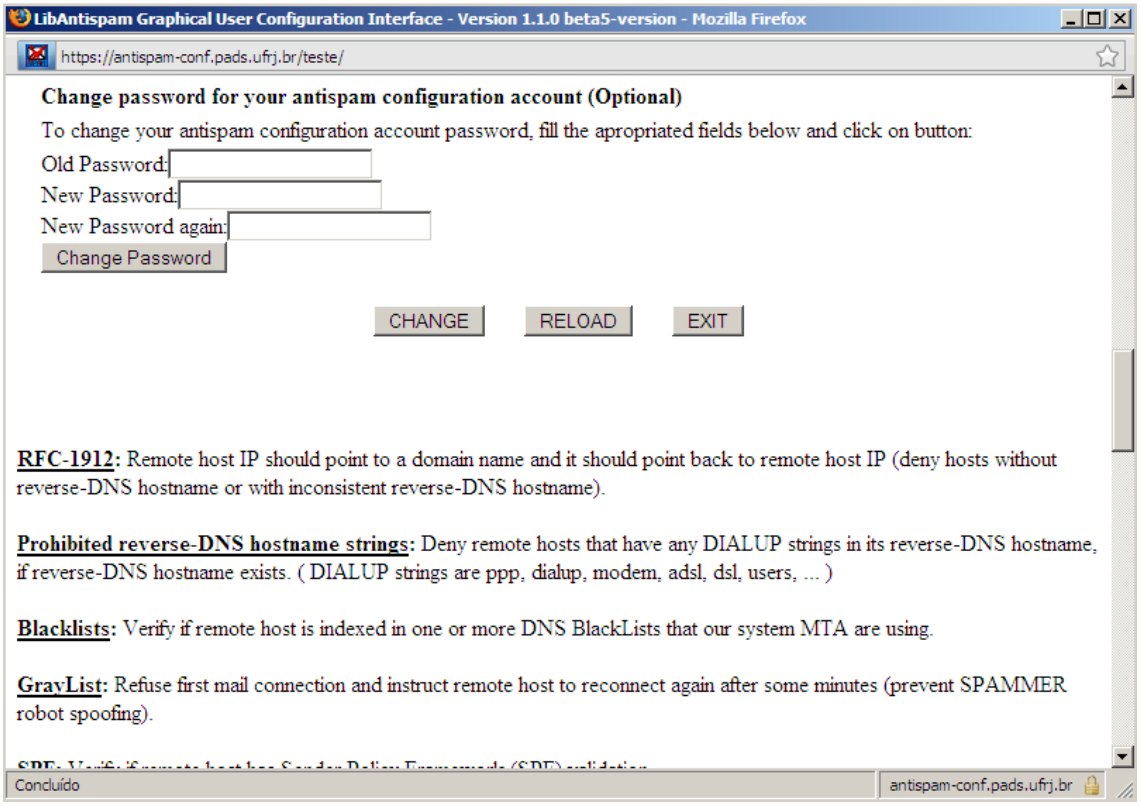

Fig. 5.10 – Interface do usuário (glossário explicativo)

Fonte: O autor do Trabalho

Capítulo 6

Análise de Desempenho

6.1 – Motivação

Neste capitulo será discorrido uma analise de desempenho das regras de 1º nível implementadas na LibAntispam. Isto é necessário para que tenhamos uma idéia exata do grau de eficácia do grau de proteção que as regras podem oferecer, seja individualmente ou em grupo. Nesta fase, também foram feitas as verificações finais para eliminação de possíveis *BUGs* nas implementações das regras.

6.2 – Metodologia

Para que a análise de desempenho fosse correta e representasse a realidade dos fatos, foi utilizada a seguinte metodologia na análise de desempenho:

- Foram utilizados três endereços de E-mail antigos da rede do PADS, todos com mais de 10 anos de uso pelos seus respectivos usuários, e que notoriamente já estão, há bastante tempo, indexados em listas *spammers* de modo irremediável.
- Os três endereços de E-mail deviam estar sob o mesmo domínio (pads.ufrj.br) e possuir regras antispam habilitadas em três diferentes níveis: Baixo, Médio e Alto. Neste caso, o nível Baixo significa sem regras antispam e ele é tomado como um ponto de comparação de desempenho para com os demais, pois como foi já visto neste trabalho, um mesmo SPAM pode ser muitas vezes enviado para vários endereços dentro de um mesmo domínio.
- Foi considerado um período de tempo de um mês (dezembro de 2008) para a análise de desempenho dos três endereços.
- A análise consistiu em verificar os logs do MTA utilizado como *mailhost* do domínio para a verificação das máquinas clientes de origem das

mensagens (IP e DNS-reverso), dos argumentos utilizados no comando MAIL FROM e o resultado da verificação feita pela LibAntispam.

- Quando a mensagem, em processo de recebimento, era sabidamente julgada como SPAM, ela recebia essa classificação, caso fosse julgada em princípio como não-SPAM ela era classificada como LEGAL, caso não pudesse ser classificada, recebia a denominação DUVIDOSA.
- Para cada mensagem em processo de recebimento, foi analisado o resultado da verificação antispam, qual regra a rejeitou (isto quando ocorria uma rejeição), se havia ocorrido um *falso-positivo* ou um *falso-negativo* e qual era a classificação da mensagem (SPAM, LEGAL ou DUVIDOSA).
- É importante frisar que, apesar de uma mensagem ser classificada como LEGAL, não significa que ela seja LEGITIMA. Isto é, a mensagem é classificada como LEGAL por ter passado pelas regras antispam habilitadas para aquele endereço e por não apresentar as características clássicas de um SPAM, como já visto neste trabalho. Porém, por razões óbvias, eu só tinha acesso a um dos mailboxes dos endereços testados, o meu mailbox, e não os dos outros dois endereços utilizados no teste, logo eu não tinha como verificar o conteúdo dos mesmos para constatar se uma determinada mensagem classificada como LEGAL era LEGITIMA. Por exemplo, ela poderia ser uma mensagem de FRAUDE (*PHISHING*) que, como já visto neste trabalho, é o tipo de mensagem que se utiliza de credenciais verdadeiras “roubadas” para *by-passar* regras antispam de 1º nível e que só podem ser verificadas por regras antispam de 2º ou 3º níveis.

6.3 – As Configurações Antispam dos Endereços Verificados

Durante a análise de desempenho, foram verificados os resultados com três endereços do domínio pads.ufrj.br, de níveis de proteção Baixo, Médio e Alto, que são, respectivamente: *mariane@pads.ufrj.br*, *petra@pads.ufrj.br* e *szendro@pads.ufrj.br*.

Suas configurações antispam são visualizadas nas figuras a seguir:

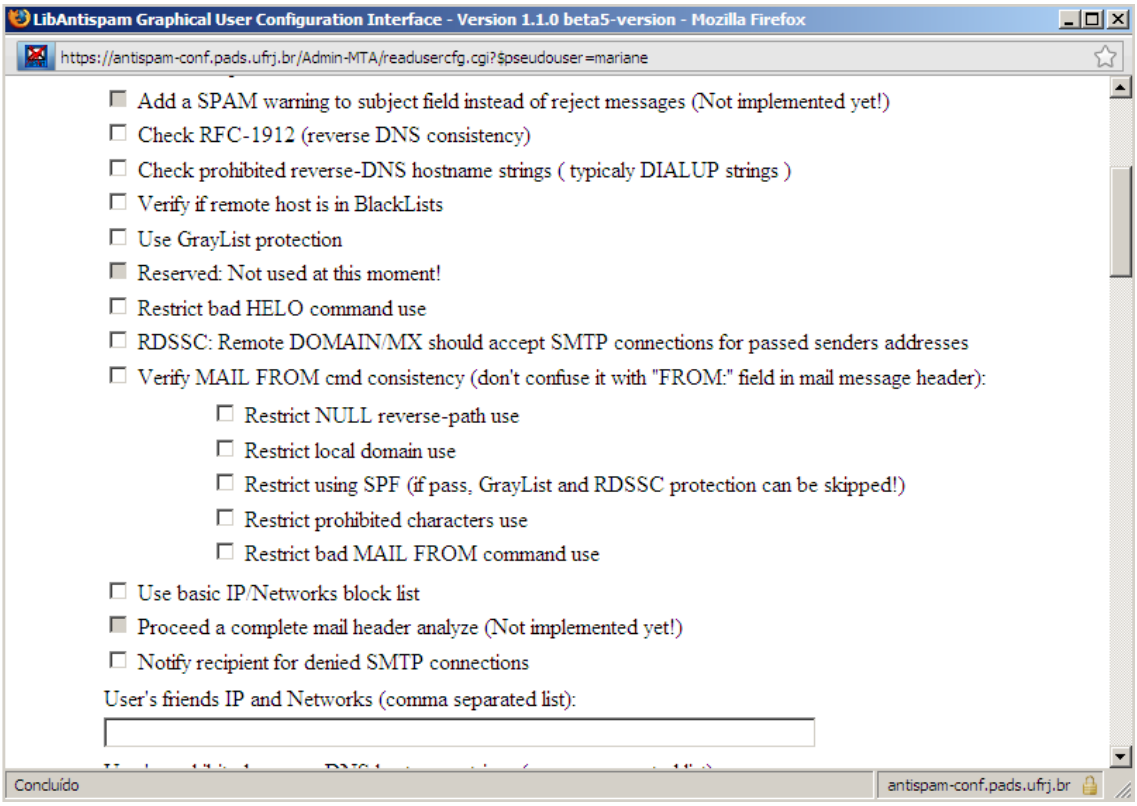

Fig. 6.1 – Configuração antispam endereço mariane@pads.ufrj.br

Fonte: O autor do Trabalho

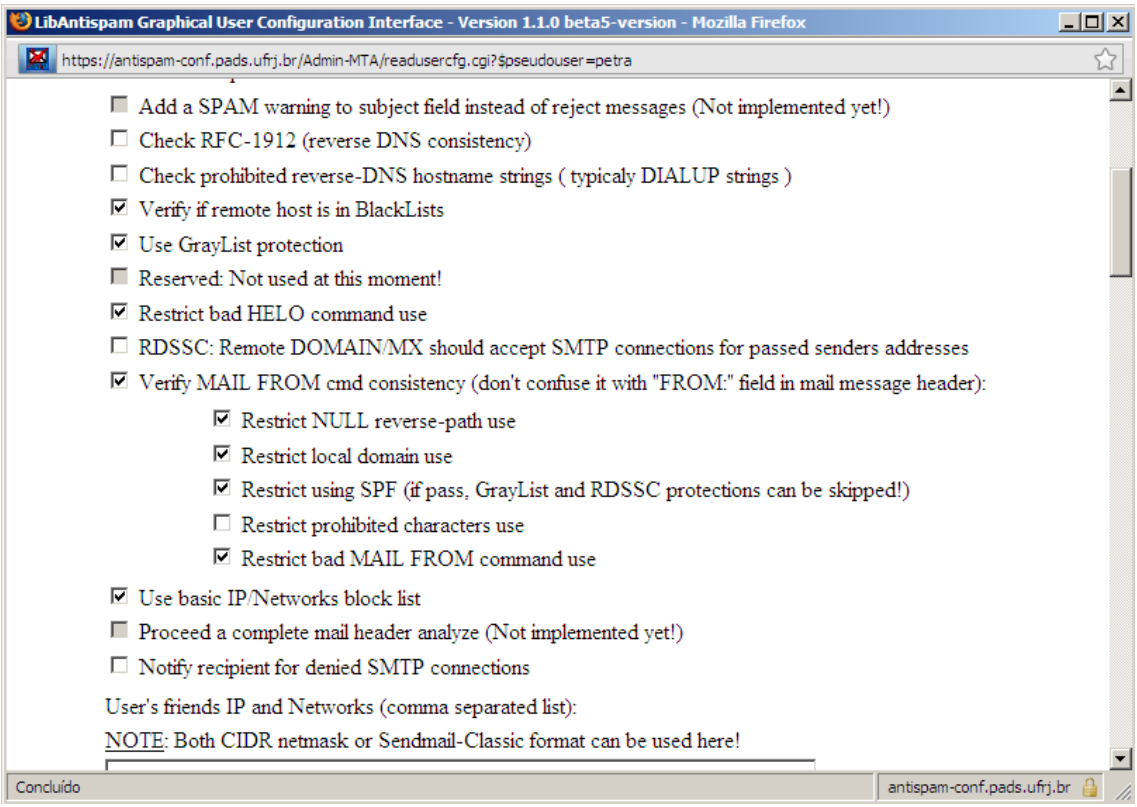

Fig. 6.2 – Configuração antispam endereço petra@pads.ufrj.br

Fonte: O autor do Trabalho

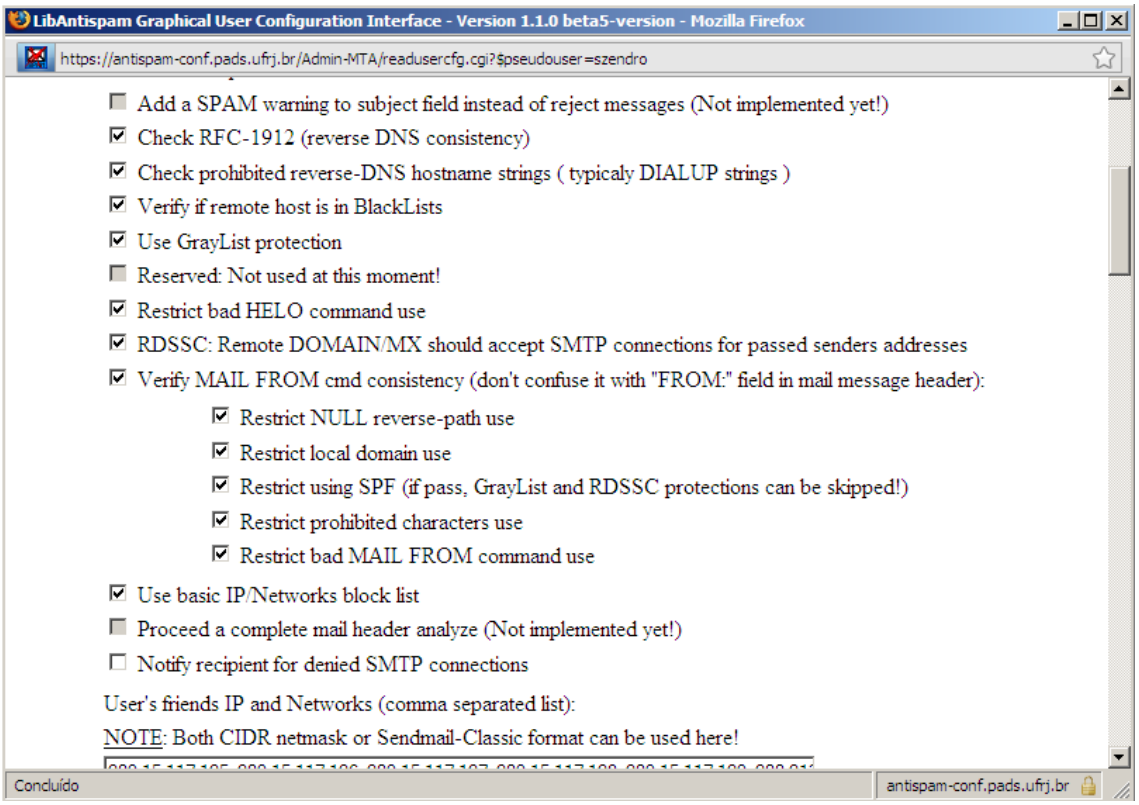

Fig. 6.3 – Configuração antispam endereço szendro@pads.ufrj.br

Fonte: O autor do Trabalho

Podemos observar nas figuras 6.1, 6.2 e 6.3 que o endereço *mariane@pads.ufrj.br* não possui nenhuma regra antispam de 1º nível habilitada, o endereço *petra@pads.ufrj.br* possui apenas as regras consideradas menos agressivas habilitadas, ou seja, como menores probabilidades de gerarem *falsos-positivos* para SPAM. Finalmente, o endereço *szendro@pads.ufrj.br* possui todas as regras antispam de 1º nível habilitadas. Como informação adicional, os três endereços citados não possuem outros tipos de filtros no MTA como, por exemplo, SpamAssassin, que poderia ser normalmente utilizado em conjunto com a LibAntispam.

6.4 – Resultados da Análise dos Logs do Período Considerado

A tabela 6.1 mostra o resultado da análise dos logs do período de um mês considerado (dezembro de 2008). A primeira coluna da tabela mostra os parâmetros utilizados na análise, a segunda, a terceira e a quarta coluna mostram as contas e os seus totais apurados para cada parâmetro analisado.

Foram considerados apenas os E-mails recebidos de fora da UFRJ. As redes internas da UFRJ são consideradas globalmente, nas configurações de LibAntispam do PADS, como redes amigas e não é feita nenhuma verificação antispam para E-mails provenientes das mesmas.

Os critérios utilizados para classificar uma mensagem como LEGAL, SPAM ou DUVIDOSA foram vistos nos capítulos anteriores deste trabalho e se baseiam, como dito anteriormente, na análise do IP, do DNS-reverso e do argumento no comando MAIL FROM. Em muitos casos, um E-mail classificado como DUVIDOSO ou LEGAL pode ser um caso de *Phishing*.

Os campos "Tentativas Totais" significa o número de vezes que se tentou conectar para enviar uma mensagem, com exceção das vezes em que a regra de GrayListing é acionada para negar a tentativa temporariamente. Foi feita esta distinção porque uma das contas não possui nenhuma regra antispam habilitada e se fossem contados os acionamento do GrayListing no total de tentativas de envio, iria ocorrer um mascaramento para melhor do desempenho da LibAntispam, ademais, regra de GrayListing não é uma regra de rejeição, mas de atraso de envio de mensagem.

Para verificar a consistência dos dados obtidos, observe que:

1. O número de TENTATIVAS é a soma dos totais de mensagens LEGAIS, SPAMs e DUVIDOSAS.
2. O número de mensagens LEGAIS é igual ao número de mensagens ACEITAS menos o número de *falsos-negativos*.
3. O número de SPAMs é igual ao número de REJEITADAS mais o número de *falsos-negativos* menos o número de *falsos-positivos*.
4. O número de REJEITADAS corresponde à soma de seus respectivos subitens, excluindo GRAYLIST.
5. Os números entre parênteses nos campos e sub-campos de ACEITAS significam:
    a. Nos campos ACEITAS, o número mensagens de FRAUDE (*Phishing*) que passaram, sendo indicadas pela sigla F.
    b. Nos sub-campos SPF de ACEITAS, o número de mensagens de FRAUDE (*Phishing*) que foram aceitas devido às regras de SPF, sendo indicadas pela sigla F.
    c. Nos sub-campos SPF de ACEITAS, o número de mensagens que são SPAMs que deram falso-negativos, indicadas pela sigla FN.

Endereços	mariane@pads.ufrj.br	petra@pads.ufrj.br	szendro@pads.ufrj.br
Tentativas Totais	892	828	419
LEGAIS	143	271	59
SPAMs	662	557	360
DUVIDOSAS	87	0	0
REJEITADAS	0	543	358
Falso-Positivos	0	1	0
RFC-1912	0	0	255
PROHIBITED STRINGS	0	0	83
BLACKLIST	0	0	0
GRAYLIST [1]	0	919	53
BAD HELLO	0	220	3
RDSSC	0	0	7
MF: NULL REVERSE-PATH	0	3	0
MF: LOCAL DOMAIN	0	281	0
MF: SPF	0	31	0
MF:PROHIBITED CHARS	0	0	3
MF: BAD MAIL FROM	0	7	7
BASIC IP LIST	0	0	0
ACEITAS	892	286	61 (6 F)
Falso-Negativos	0	15	2
SPF	0	128 (6 FN)	29 (1 F) (1 FN)

Tabela 6.1 – Análise dos Resultados da LibAntispam no período de um mês

Fonte: O autor do Trabalho

Notas: [1] *GrayListing não é regra de rejeição, é regra de atraso de entrega. Foi aqui totalizado o número de vezes que a regra foi acionada, que pode ser maior do que o número de tentativas recebidas. Se a regra provocar rejeição no cliente que envia, o ERRO é dele.*

Endereços	mariane@pads.ufrj.br	petra@pads.ufrj.br	szendro@pads.ufrj.br
Tentativas Totais	100%	100%	100%
LEGAIS	16,0% [1]	32,7%	14,1%
SPAMs	74,2%	67,3%	85,9%
DUVIDOSAS	9,8%	0%	0%
REJEITADAS	0%	65,6% [2]	85,4% [2]
Falso-Positivos	0%	0,1%	0%
RFC-1912	0%	0%	60,9%
PROHIBITED STRINGS	0%	0%	19,8%
BAD HELLO	0%	26,6%	0,7%
RDSSC	0%	0%	1,7%
MF: NULL REVERSE-PATH	0%	0,4%	0%
MF: LOCAL DOMAIN	0%	33,9%	0%
MF: SPF	0%	3,7%	0%
MF:PROHIBITED CHARS	0%	0%	0,7%
MF: BAD MAIL FROM	0%	0,8%	1,7%
ACEITAS	100%	34,5%	14,6% (1,4% F)
Falso-Negativos	0%	1,7%	0,5%
SPF	0%	15,1% (0,7% FN)	6,9% (0,2% F) (0,2% FN)

Tabela 6.2 – Resultados em percentuais aproximados (excluindo GrayList e linhas com 0% nos três endereços)

Fonte: O autor do Trabalho

Notas: [1] *A prof. Mariane recebe cerca de metade de suas mensagens legais pelo seu endereço no LPS, que aqui não foram considerados, por isso essa diferença acentuada em relação as mensagens legais do prof. Petraglia.*

[2] *A relação entre mensagens Rejeitadas e SPAMs ficou em ≈ 97,5% e ≈ 99,5% para os endereços petra@pads.ufrj.br e szendro@pads.ufrj.br, respectivamente.*

6.5 – Interpretação dos Resultados Obtidos

Dos resultados obtidos, podemos extrair algumas informações importantes:

- Haja visto os resultados obtidos pelas filtragens nos endereços *szendro@pads.ufrj.br* e *petra@pads.ufrj.br*, neste último principalmente, pois o total de tentativas nele foram semelhantes a do endereço *mariane@pads.ufrj.br*. Podemos concluir, pelo altos percentuais de SPAMs filtrados nesses dois endereços (≈97,5% e ≈99,5%), que não fazer uso de quaisquer filtros antispams não é uma boa prática.
- O uso de regras antispam de nível de proteção médio, sem as regras mais agressivas de checagem de DNS reverso e verificação de RDSSC, provoca uma alta utilização da regra de GrayListing, o que se traduz por um maior número de conexões SMTP que o MTA terá que atender. Neste caso, a maior parte da filtragem é realizada pelas regras de checagem do uso do domínio local nos endereços dos remetentes em conjunto com a regra de GrayListing seguido da checagem da consistência dos FQDN passados nos comandos HELO durante o envelopamento da mensagem.
- Ainda sobre o uso de regras antispam de nível de proteção médio, foi verificado durante a análise dos *logs* que os engenhos *robots* geralmente fazem três tentativas de envio de uma mensagem, sempre utilizando endereços aleatórios de domínios coletados na Internet. Observe que o percentual de *falsos-negativos* para SPAMs gerados por SPF (que representou quase 5% das verificações SPF) se deveram a domínios com as entradas SPF mal-configuradas. Isto desde já demonstra que o SPF realmente pode ser considerado uma "*faca-de-dois-gumes*" e que se futuramente os *spammers* perceberem esse fato, eles utilizarão maciçamente domínios com entradas SPF mal-configuradas para comporem o endereço do remetente de suas mensagens, pois passando na verificação de SPF consegue-se geralmente *by-passar* outras regras de autenticação como RDSSC e GrayListing.

- O uso de regras antispam de nível de proteção alto, ao contrário do que se esperaria, não ocasionou um numero maior de *falsos-positivos* para SPAM, conforme sempre se acreditou. E pelo percentual de *falsos-negativos* para mensagens SPAM aceitas como LEGAIS ter sido mais de 3 vezes menor do que com o nível de proteção médio, conclui-se que este nível de proteção é mais eficiente para barrar o recebimento de mensagens não-solicitadas.
- Ainda sobre o uso de regras antispam de nível de proteção alto, verificamos que das mensagens SPAM rejeitadas, cerca de 94% delas vieram de máquinas que apresentavam algum tipo de problema relacionado a consistência do DNS reverso, seja pela falta do mesmo (cerca de 71% dos caso) ou por provirem de máquinas suspeitas de pertencerem a faixas de usuários ou dinâmicas (cerca de 23% dos casos).
- Verificamos que o fluxo de E-mails para a conta *szendro*, principalmente SPAMs, é cerca de metade do apurado para as contas *petra* e *mariane*, mas no passado antes da LibAntispam ser usada na conta *szendro* (em uso desde o início de 2006), esse fluxo era comparável nas três contas. Creditamos isto à LibAntispam ter tornado o E-mail *szendro* mais protegido, o que levou a uma retirada do mesmo de muitas listas *spammers*. A conta *petra* começou a usar o suporte da LibAntispam apenas a partir de setembro de 2008.
- Podemos verificar ainda que a LibAntispam apresentou resultados bastante satisfatórios se considerarmos os baixos percentuais de *falso-negativos* (entre 0,5% e 1,7%) e, principalmente, de *falsos-positivos* (menos de 1%). A passagem de um ou outro *falso-negativo* é perfeitamente aceitável para a grande maioria dos usuários, porém a perda de uma mensagem legítima devido a um *falso-positivo* pode representar até a perda de um grande negócio. Porém aqui novamente neste ponto posso ser incisivo: Do que adianta não ter filtros antispam se, no desespero ou pressa de apagar todas as mensagens indesejadas o usuário por engano apagar uma mensagem legítima e que representava um grande negócio. Isto pode ser interpretado como o equivalente a um *falso-positivo*, mas no ato de se apagar mensagens não-solicitadas.

Capítulo 7

Conclusões Finais

A prática do SPAM (envio de mensagens não-solicitadas) é nos dias atuais uma das maiores pragas que assolam a Internet. Vimos que não se trata de um fenômeno recente, como muitos supõem, mas tem sua origem nos primórdios da Internet, embora a sua explosão maciça tenha ocorrido apenas a partir da metade dos anos 90.

Ao contrario do senso-comum, vimos que o maior prejuízo causado pelo SPAM não se resume apenas ao recebimento de mensagens indesejadas pelos destinatários finais das mesmas, mas engloba toda a cadeia de recursos da Internet, sejam os provedores de acesso que pagam pela banda-passante que utilizam, sejam os provedores e os usuários, que gastam recursos em termos de tempo de CPU e de armazenamento físico dos dados em disco, sejam dos roteadores de Internet, que tem o seus trabalho consideravelmente aumentado, sejam todos em conjunto em termos da energia, seja ela humana ou elétrica, gasta durante todo esse processo.

A utilização de regras antispam de 1º nível, contrariando o senso-comum de muitos administradores de rede, não é prejudicial aos usuários se os mesmos tiverem atenção e procurarem sempre estar em contato com os administradores de rede para sanar eventuais problemas ocasionados por *falsos-positivos* e por *falsos-negativos*. É sempre melhor a utilização dos recursos antispam do que a ausência total deles.

Verificamos também, pelos resultados obtidos, que o uso de regras de 1º, 2º e 3º níveis pode e deve ser feita, pois contrariando o senso-comum, elas não são auto-excludentes, mas complementares entre si. Deixar a filtragem de SPAMs a cargo apenas do MUA (3º nível) é uma política errada e que em muito sacrifica os recursos da Internet de modo geral.

É sempre bom lembrar, entretanto, que nunca existirá a ferramenta antispam 100% perfeita, sempre se deverá ter em mente que algum SPAM sempre conseguirá acabar chegando aos mailboxes dos usuários. A melhor prática é não se apoiar em apenas uma ou outra regra ou ferramenta antispam, mas procurar utilizar um grupo delas para se ter uma proteção mais efetiva.

Finalmente, podemos verificar que a LibAntispam provou sua eficiência como uma ferramenta que congrega diversas regras de 1º nível, facilitando a sua incorporação nos mais diversos MTAs, universalizando as sintaxes das regras de configuração e facilitando a sua intercambialidade entre os mais diversos sistemas e separando a implementação dos MTAs da implementação das proteções antispam, conferindo dessa forma uma eficácia maior na filtragem de mensagens indesejadas ao mesmo tempo em que procura deixar os níveis de casos de *falsos-positivos* dentro de patamares aceitáveis.

Ainda sobre a LibAntispam, a implementação e o design de sua interface gráfica, LibAntispam-GUI, permite uma maior facilidade e rapidez para a configuração antispam em um sistema, simplificando tanto para o administrador de rede quanto para o usuário da mesma o acesso e alteração de suas regras, até independentemente da interferência daquele.

Embora seja eficiente, acredito que a adoção da LibAntispam como a ferramenta antispam de 1º nível *defacto* seja algo para ser vislumbrado para daqui a alguns anos, pois é difícil modificar a mentalidade e as práticas já há muito tempo enraizadas no seio dos administradores de redes da Internet. Porem, a necessidade faz a ocasião, e acredito que um dia a mesma venha a ganhar o destaque que realmente deva ter.

Bibliografia

[1] WIKIPÉDIA, *"SPAM"*, http://pt.wikipedia.org/wiki/Spam, 2009, (Acesso em 25 de Janeiro de 2009).

[2] GRAHAM, P., *"Filtro Bayesiano"*, http://pt.wikipedia.org/wiki/Filtro_Bayesian, 2009, (Acesso em 25 de Janeiro de 2009).

[3] KLENSIN, J., *"Simple Mail Transport Protocol"*, http://www.ietf.org/rfc/rfc5321.txt?number=5321, 2008 (Acesso em 25 de Fevereiro de 2009).

[4] SENDMAIL, *"Sendmail Home"*, http://www.sendmail.org, 2008, (Acesso em 25 de Janeiro de 2009).

[5] META1, *"MeTA1 – Message Transfer Agent"*, http://www.meta1.org, 2008, (Acesso em 25 de Janeiro de 2009).

[6] EXIM, *"The Exim Home Page"*, http://www.exim.org, 2008, (Acesso em 25 de Janeiro de 2009).

[7] POSTFIX, *"The Postfix Home Page"*, http://www.postfix.org, 2008, (Acesso em 25 de Janeiro de 2009).

[8] WONG, M.; SCHLITT, W., *"Sender Policy Framework for Authorizing Use of Domains in E-mail, Version 1"*, http://www.ietf.org/rfc/rfc4408.txt, 2006, (Acesso em 25 de Janeiro de 2009).

[9] INTERNET ENGINEERING TASTK FORCE, *"IETF Home"*, http://www.ietf.org, 2008, (Acesso em 25 de Janeiro de 2009).

[10] WIKIPÉDIA, *"Monty Python"*, http://pt.wikipedia.org/wiki/Monty_Python, 2009, (Acesso em 31 de Janeiro de 2009).

[11] WIKIPÉDIA, *"ARPAnet"*, http://pt.wikipedia.org/wiki/ARPANET, 2009, (Acessado em 31 de Janeiro de 2009).

[12] POSTEL, J. B., *"Simple Mail Transfer Protocol"*, http://www.ietf.org/rfc/rfc0821.txt, 1982, (Acesso em 31 de Janeiro de 2009).

[13] KELSIN, J., *"Simple Mail Transfer Protocol"*, http://www.ietf.org/rfc/rfc2821.txt?number=2821, 2001, (Acesso em 31 de Janeiro de 2009).

[14] RESNICK, P., *"Internet Message Format"*, http://www.ietf.org/rfc/rfc5322.txt?number=5322, 2008, (Acesso em 25 de Fevereiro de 2009).

[15] CROCKER, D. H., *"Standard for the Format of ARPA Internet Text Messages"*, http://www.ietf.org/rfc/rfc0822.txt?number=822, 1982, (Acesso em 01 de Fevereiro de 2009).

[16] RESNICK, P., *"Internet Message Format"*, http://www.ietf.org/rfc/rfc2822.txt?number=2822, 2001, (Acesso em 01 de Fevereiro de 2009).

[17] TEMPLETON, B., *"Origin of the term "Spam" to mean net abuse"*, http://www.templetons.com/brad/spamterm.html, 2003, (Acesso em 31 de Janeiro de 2009).

[18] INFOSECUTIRY TASK FORCE, *"Fake Mail"*, http://www.istf.com.br/vb/penetration-tests/2396-fake-mail.html, 2002, (Acesso em 01 de Feveiro de 2009).

[19] Szendrodi, R. J. C., *"O SPAM – Suas origens, sua evolução e as técnicas para evitá-lo!"*, http://www.gta.ufrj.br/grad/05_1/spam/szendro/index.html, 2005, (Acesso em 01 de Fevereiro de 2009).

[20] VLECK, T.V., *"The History of Electronic Mail"*, http://www.multicians.org/thvv/mail-history.html, 2001, (Acesso em 31 de Janeiro de 2009).

[21] WIKIPÉDIA, *"Phishing"*, http://pt.wikipedia.org/wiki/Phishing, 2009, (Acesso em 01 de Fevereiro de 2009).

[22] WIKIPÉDIA, *"Engenharia Social (Segurança da Informação)"*, http://pt.wikipedia.org/wiki/Engenharia_social_(segurança_da_informação), 2009, (Acesso em 01 de Fevereiro de 2009).

[23] TAVEIRA, D. M., *Mecanismo Anti-Spam Baseado em Autenticação e Reputação*. M.Sc. dissertation, Universidade Federal do Rio de Janeiro, Março 2008.

[24] REKHTER, Y., LI, T., A Border Gateway Protocol 4 (BGP-4), março de 1995. RFC 1771.

[25] RAMACHANDRAN, A., FEAMSTER, N., Understanding the network-level behavior of spammers. Em *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications* (2006), ACM Press, pág. 291–302.

[26] WITTEL, G. L., WU, S. F., On attacking statistical spam filters. Em *First Conference on Email and Anti-Spam (CEAS2004)* (julho de 2004).

[27] WIKIPÉDIA, *"SpamAssassin"*, http://en.wikipedia.org/wiki/SpamAssassin, 2009, (Acesso em 08 de Fevereiro de 2009).

[28] TREND MICRO, *"MAPS"*, http://www.mail-abuse.com, 2005, (Acesso em 15 de Fevereiro de 2009).

[29] SPAMHAUS, *"The Spamhaus Project"*, http://www.spamhaus.org, 2009, (Acesso em 21 de Fevereiro de 2009).

[30] WIKIPÉDIA, *"GreyListing"*, http://en.wikipedia.org/wiki/Greylisting, 2009, (Acesso em 15 de Fevereiro de 2009).

[31] BARR, D., *"Common DNS Operational and Configuration Errors"*, http://www.ietf.org/rfc/rfc1912.txt?number=1912, 1996, (Acessado em 15 de Fevereiro de 2009).

[32] WIKIPÉDIA, *"BIND"*, http://pt.wikipedia.org/wiki/BIND, 2008, (Acesso em 21 de Fevereiro de 2009).

[33] HARRENSTIEN, K., STAHL, M., FEINLER, E., *"DOD Internet Host Table Specification"*, http://www.ietf.org/rfc/rfc0952.txt?number=952, 1985, (Acesso em 21 de Fevereiro de 2009).

[34] BRADEN, R. *"Requiriments for Internet Hosts – Application and Support"*, http://www.ietf.org/rfc/rfc1123.txt?number=1123, 1989, (Acesso em 01 de Março de 2009).

[35] REKHTER, Y., MOSCOWITZ, B., KARRENBERG, D., GROOT, G.J.DE, LEAR, E., *"Address Allocation for Private Internets"*, http://www.ietf.org/rfc/rfc1918.txt?number=1918, 1996 (Acesso em 01 de Março de 2009).

[36] MOCKAPETRIS, P., *"Domain Names – Concept and Facilities"*, http://www.ietf.org/rfc/rfc1034.txt?number=1034, 1987, (Acesso em 25 de Fevereiro de 2009).

[37] MOCKAPETRIS, P., *"Domain Names – Implementation and Specification"*, http://www.ietf.org/rfc/rfc1035.txt?number=1035, 1987, (Acesso em 25 de Fevereiro de 2009).

[38] HARRIS, E., *"The Next Step in the Spam Control War: GreyListing"*, http://projects.puremagic.com/greylisting/whitepaper.html, 2003, (Acesso em 15 de Março de 2009).

[39] VILLAS-BOAS, S.B., MARTINS, A., *"VBMcgi home page"*, http://www.sbvb.com.br/cgi-bin/index_vbmcgi.cgi?p=0, 2007, (Acesso em 22 de Março de 2009).