Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Ricardo Takeshi Yamaguti

Combate ao spam: um estudo de caso

São Paulo

2006

Ricardo Takeshi Yamaguti

Combate ao spam: um estudo de caso

Dissertação apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT, para obtenção do título de Mestre em Engenharia de Computação.
Área de concentração: Redes de computadores

Orientador: Prof. Dr. Antonio Luiz Rigo

São Paulo
Março de 2006

Ficha Catalográfica
Elaborada pelo Centro de Informação Tecnológica do
Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT

Y19c Yamaguti, Ricardo Takeshi
Combate ao spam: um estudo de caso. / Ricardo Takeshi Yamaguti. São Paulo, 2006.
158p.

Dissertação (Mestrado em Engenharia de Computação) - Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Área de concentração: Redes de Computadores.

Orientador: Prof. Dr. Antonio Luiz Rigo

1. E-mail 2. Spam 3. Servidor de e-mail 4. Anti-spam 5. Antivírus 6. Código malicioso 7. SpamAssassin 8. Filtro bayesiano 9. Software livre 10. Correio eletrônico 11. Tese I. Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Centro de Aperfeiçoamento Tecnológico II.Título

06-39 CDU 004.49'773.3(043)

Dedicatória

Dedico este trabalho aos meus pais Mitsuko e Noriyoshi.

Agradecimentos

A todos que me apoiaram, ajudaram e vibraram a cada passo conquistado. Em especial aos meus pais, que forneceram toda a estrutura para que eu pudesse me dedicar ao trabalho e aos estudos.

Ao Prof. Dr. Antonio Luiz Rigo que aceitou o convite para ser meu orientador.

Aos meus colegas de trabalho, que fazem parte do meu dia-a-dia e de uma forma ou outra me ajudaram.

Resumo

E-mail tornou-se uma ferramenta de comunicação muito importante, pois permite a troca de informações com rapidez e a um custo relativamente baixo. Por essas características foi adotado por empresas e instituições públicas. No seu projeto inicial, apenas grupos restritos usavam esse sistema, razão pela qual exigia poucos recursos de segurança.

Nos últimos anos, a multiplicação de *e-mails* de propaganda, também conhecidos como spam, golpes e *e-mails* com código malicioso como vírus, *worms* e cavalos de tróia começaram a afetar os usuários de serviço de correio eletrônico, resultando em perda de produtividade e aumento na quantidade de chamados de suporte técnico para esclarecer dúvidas de usuários sobre esses *e-mails*.

Pretende-se então analisar “spam” no contexto de quem o envia e de quem o recebe para buscar a prevenção e o combate para mitigar os malefícios causados por essa prática.

Tendo em vista esse propósito, decidiu-se elaborar estudo prático para avaliar o funcionamento de um servidor de *e-mail* com sistema anti-spam e antivírus em ambiente de produção.

Os custos de aquisição e de renovação de licenças de *software* proprietários podem ser decisivos para a adoção de uma solução para filtragem de *e-mails* indesejados. Então, optou-se pela implantação de um sistema para filtragem de *e-mails* baseado em *software* livre: sistema operacional Linux, MTA Qmail, filtro anti-spam SpamAssassin e antivírus ClamAV.

Regras baseadas em palavras-chave podem ser usadas para filtrar spam. No entanto, características alteradas no spam implicam em gastar tempo para alterar ou criar novas regras. Para não onerar o administrador do servidor com essa atividade, o sistema anti-spam faz uso de regras por palavras-chave pré-definidas no *software* e

utiliza também outras técnicas de detecção de spam como o filtro bayesiano que não dependem do administrador para se manterem atualizadas e funcionais.

O servidor de *e-mail* com sistema anti-spam e antivírus traz benefícios aos técnicos de suporte técnico como redução na quantidade de atendimentos de casos de código malicioso recebido por *e-mail* e também esclarecimentos sobre *e-mails* forjados por *spammers* e *hackers*. Os técnicos avaliam esses itens e outros através de questionário.

Palavras-chave: *e-mail*, spam, anti-spam, SpamAssassin, filtro bayesiano, quarentena, antivírus, código malicioso, *software* livre.

Abstract

Combating spam: a case study

E-mail became an important communication tool, because it permits fast and relatively low cost information exchange. For these characteristics, it was adopted by enterprises and public institutions. In its initial project, only restrict groups used this system, and that's the reason why it demanded few security resources.

In the last years, the multiplication of commercial e-mails, also known as spam, scams, and e-mails with malicious code such as viruses, worms and trojans began to affect electronic mail service users, resulting in loss of productivity and raise in the amount of technical support requests to elucidate users' doubts about these e-mails.

The analysis of "spam" in the context of who sends and who receives it is intended to aim for prevention and combat to mitigate the effects of the damages caused by this problem.

For this purpose, it was decided to make a practical study to evaluate an e-mail server with an antispam system and antivirus in production environment.

The costs for acquiring and renewing proprietary software licenses can be decisive to adopt a solution for filtering undesirable e-mails. Then a free software solution for e-mail filtering was chosen: Linux operational system, Qmail MTA, antispam filter SpamAssassin, and antivirus ClamAV.

Rules based on keywords can be used to filter spam. However, characteristics altered in spam results in spending time to update or to create new rules. As the server administrator shouldn't be burdened by this activity, the antispam system uses the keyword rules predefined in the software and also other spam detection techniques that don't depend on the administrator to be up-to-date and functional.

The e-mail server with antispam system and antivirus brings benefits to the technical support staff such as reduction in the amount of requests to deal with malicious code received by e-mail and also elucidations about e-mails forged by spammers and hackers. The technicians evaluate these items and others through a questionnaire.

Keywords: e-mail, spam, anti-spam, SpamAssassin, bayesian filter, quarantine, antivirus, malicious code, free software.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ARPANET	*Advanced Research Projects Agency Network*
ASCII	*American Standard Code for Information Interchange*
BMC	*Bad MIME Characters*
CD	*Compact Disk*
CERT.br	*Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil*
CPAN	*Comprehensive Perl Archive Network*
DHA	*Directory Harvest Attack*
DNS	*Domain Name System*
DNSBL	*DNS-Based Blackhole List*
DoS	*Denial of Service*
DT/SIBi/USP	Departamento Técnico do Sistema Integrado de Bibliotecas da Universidade de São Paulo
E-mail	*Electronic mail*
FTC	*Federal Trade Commission*
GNU	*GNU's Not Unix*
HTML	*Hypertext Markup Language*
IDE	*Integrated Drive Electronics*
IMAP	*Internet Message Access Protocol*
IP	*Internet Protocol*
MIME	*Multi-Purpose Internet Mail Extensions*
MTA	*Mail Transfer Agent*
MUA	*Mail User Agent*
PERL	*Practical Extraction and Report Language*
PID	*Process ID*
POP	*Post Office Protocol*
RAID	*Redundant Array of Inexpensive Disks*
RHN	*Red Hat Network*
RFC	*Request For Comments*

SMTP	*Simple Mail Transfer Protocol*
TCP	*Transmission Control Protocol*
UBE	*Unsolicited Bulk E-mail*
UCE	*Unsolicited Commercial E-mail*
UOL	Universo *Online*
URL	*Uniform Resource Locator*
Wi-Fi	*Wireless Fidelity*
WLAN	*Wireless Local Area Network*

Sumário

Capítulo 1

1 Introdução

O sistema de correio eletrônico ou *e-mail* (*electronic mail*) é um meio de comunicação comum e importante nas empresas. A pesquisa intitulada “Perfil da Indústria Digital” (FAVA, 2002) realizada com empresas industriais brasileiras apresentou a seguinte informação:

> [D]estaca-se a grande utilização do correio eletrônico e o acesso a páginas Web através da Internet. Este resultado é significativo, pois indica que a grande maioria das pequenas e médias empresas e a totalidade das maiores já adota a Internet como um importante canal de comunicação para a realização de negócios.

Nos últimos anos, as caixas postais de correio eletrônico estão sendo invadidas por *e-mails* de propaganda (spam), golpes, correntes, boatos e vírus. Eles consomem recursos da infra-estrutura, incomodam, geram dúvidas, distraem os usuários e ainda podem ser danosos como os vírus e os golpes.

A mala-direta efetuada pelo correio convencional implica em gastos com envelope, papel, impressão e pagamento de taxa de envio. O custo total para o envio de poucas cartas dificilmente seria um grande impedimento para tal prática, mas e se fosse necessário enviar, por exemplo, três milhões de cartas?

Um usuário doméstico com um computador, uma linha telefônica e um provedor de acesso à Internet praticamente não teria mais nenhum custo para enviar milhares de *e-mails*. Além disso, dependendo da região em que o usuário mora, existem provedores de acesso à Internet gratuitos e taxas promocionais do impulso telefônico para dias e horários específicos.

A estatística do CERT.br (Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil) da Figura 1 indica que em 2003 e 2004 cerca de 4 milhões de notificações de spam oriundo do Brasil foram recebidas pelo centro e em

2005, foram cerca de 2 milhões e quatrocentas mil notificações (CERT.BR, 2006). Apesar da diminuição, a quantidade de notificações de spam ainda é significativa.

O relatório da Symantec (2005a) "*Internet Security Threat Report* – Relatório Regional América Latina" referente ao primeiro semestre de 2005 estima que o spam representou 61% de todo o tráfego de *e-mail*. A estatística ilustrada na Figura 2, de 08.02.2006, estima que cerca de 69,9% do tráfego de *e-mail* analisado pela Postini é spam (POSTINI, 2006).

A quantidade enorme de spam circulando na Internet é um indicador da necessidade de sistema anti-spam nas empresas.

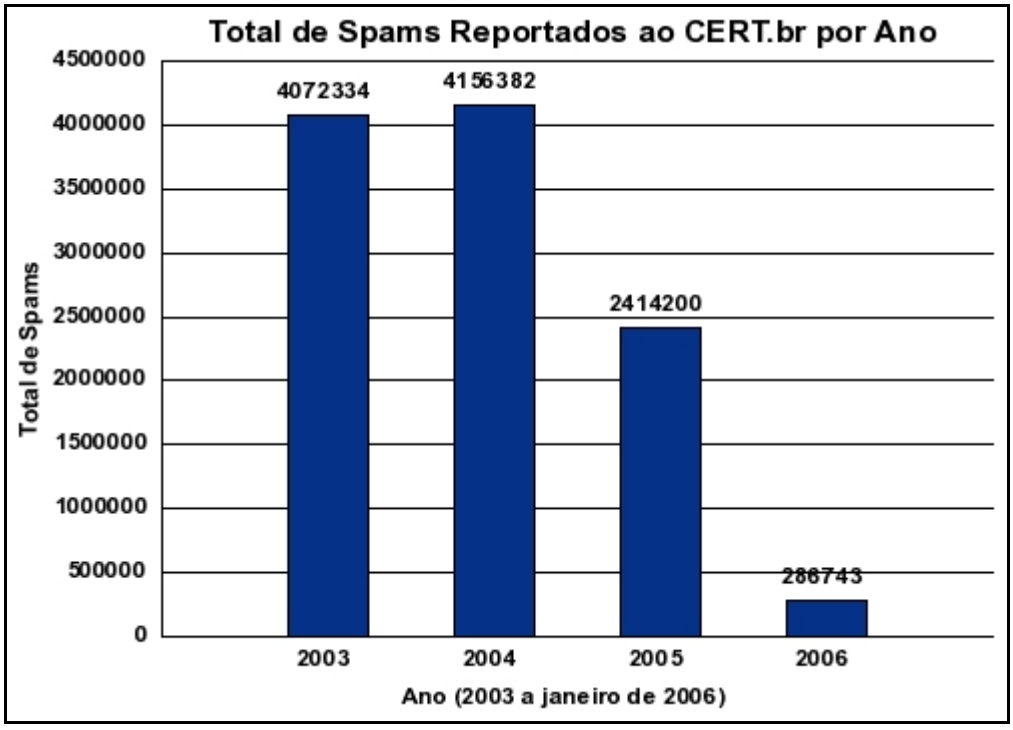

Figura 1 - Estatística de spam reportado ao CERT.BR anualmente.

Código malicioso, como vírus, *worm* e cavalo de tróia, não deve ser relegado ao segundo plano já que pode depender do sistema de *e-mail* para se propagar ou para chegar até as vítimas. Além de inutilizar computadores e interferir com o tráfego da rede, pode contribuir com o envio de spam ao transformar computadores de usuários em servidores de aplicações que são explorados por *hackers* e *spammers*.

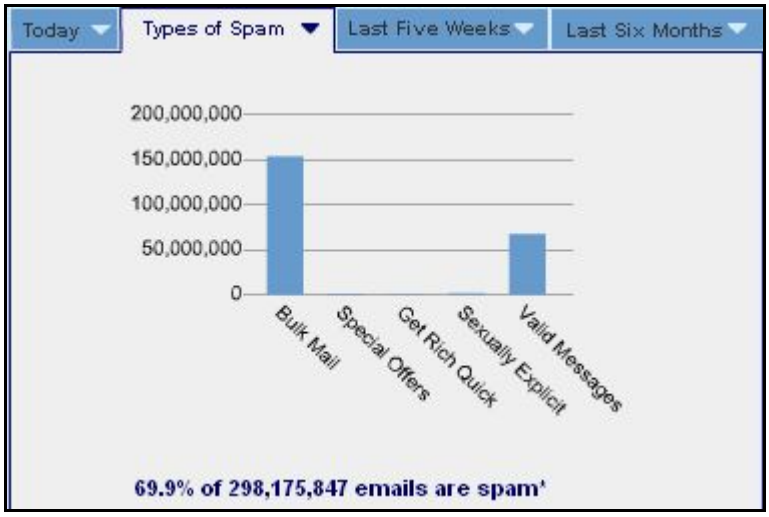

Figura 2 - Estatística de spam da Postini.

1.1 Objetivo

- Fazer levantamento sobre os elementos envolvidos com a disseminação do spam;
- Implantar um servidor de *e-mail* com sistema anti-spam e antivírus em ambiente de produção com a possibilidade de bloquear *e-mails* classificados como spam e código malicioso em área de quarentena. O servidor deve ser implantado de modo a não onerar o administrador do servidor com atividades como coleta e análise de spam para a criação e alteração de regras;
- Avaliar o sistema implantado;
- Propor mecanismo de prevenção e combate.

1.2 Justificativa

E-mails indesejados diminuem a produtividade das pessoas. Perde-se tempo para separar *e-mails* legítimos do lixo eletrônico das caixas postais. *E-mails* indesejados podem distrair as pessoas, ser considerados ofensivos por apresentar conteúdo pornográfico, ser usados para aplicar golpes e conter código malicioso.

Há alguns anos, funcionários do DT/SIBi/USP (Departamento Técnico do Sistema Integrado de Bibliotecas da Universidade de São Paulo) acusavam o recebimento de uma quantidade considerável de propagandas, vírus, *worms* e cavalos de tróia por *e-mail*, causando os problemas citados acima. Era evidente o consumo de tempo e desvio de objetivos dos membros da equipe de informática que atendiam aos usuários na análise e no esclarecimento de dúvidas relacionadas a esses *e-mails*.

1.3 Metodologia

Pesquisa bibliográfica sobre spam, comportamento dos *spammers* (responsáveis pelo envio de spam), técnicas anti-spam e recomendações para o uso do *e-mail*.

A parte prática do trabalho consiste em um estudo de caso sobre a implantação de sistema anti-spam baseado em software livre em regime de produção. Segundo YIN (2005) “O estudo de caso é a estratégia escolhida ao se examinarem acontecimentos contemporâneos, mas quando não se podem manipular comportamentos relevantes”.

Como exemplo de acontecimentos contemporâneos, YIN cita a “observação direta dos acontecimentos que estão sendo estudados e entrevistas das pessoas neles envolvidas”. Nesse estudo, há observação direta dos acontecimentos desde a implantação, administração e obtenção dos resultados. Além disso, um questionário foi aplicado aos técnicos de suporte técnico para obter informações que contribuíram na avaliação do sistema anti-spam e antivírus implantado.

A manipulação de comportamentos relevantes, segundo YIN, ocorre “quando o pesquisador pode manipular o comportamento direta, precisa e sistematicamente. Isso pode ocorrer em um laboratório, no qual o experimento pode focar uma ou duas variáveis isoladas (e presume que o ambiente de laboratório possa “controlar” todas as variáveis restantes além do escopo de interesse)”. O sistema anti-spam e antivírus

foi implantado em regime de produção, impossibilitando o controle sobre os e-mails indesejados recebidos, ou seja, para esse tipo de abordagem é indicado o estudo de caso.

No servidor de e-mail foram utilizados o sistema operacional Linux, sistema anti-spam SpamAssassin e antivírus ClamAV. Optou-se pelo *software* livre devido à liberdade de uso (GNU, 2005), sem pagamento de licenças, e também pela estabilidade do sistema operacional.

O sistema foi configurado com as técnicas anti-spam de verificação em DNSBL (*DNS-Based Blackhole List*), filtro bayesiano, verificação de assinatura e sistema de *auto-whitelist*. As regras de filtragem por palavras-chave pré-definidas no *software* também foram utilizadas.

O SpamAssassin atribui pontuações aos *e-mails* conforme características encontradas pelas técnicas anti-spam. Se a somatória das pontuações superar a nota de corte, definida na configuração, o *e-mail* é classificado como spam. Caso contrário, é classificado como legítimo. *E-mails* com até um ponto a mais que a nota de corte são identificados com “[SPAM]” no campo “assunto” e depois são encaminhados aos destinatários. Os demais com pontuação acima dessa são detidos em quarentena.

A rotulagem de *e-mails* com pontuação acima da nota de corte permite que usuários monitorem parte dos *e-mails* classificados como spam e notifiquem o administrador caso ocorram falsos positivos ou *e-mails* legítimos classificados como spam.

A implantação do sistema anti-spam e antivírus em ambiente de produção implicou na migração do serviço de *e-mail* já existente no DT/SIBi/USP. O servidor com sistema anti-spam e antivírus foi instalado e, inicialmente, foi testado por funcionários da área de informática. Os *e-mails* colocados em quarentena foram monitorados para ter certeza que não ocorreram falsos positivos ou *e-mails* legítimos classificados como spam.

Casos de falsos positivos geram insegurança aos usuários do correio eletrônico e atrapalham o fluxo de troca de informações. Seria difícil saber, por exemplo, se um *e-mail* não foi respondido ou se foi bloqueado no servidor. Como conseqüência, o administrador do serviço de correio eletrônico seria constantemente questionado pelos usuários para saber se há *e-mails* legítimos bloqueados.

Para diminuir as chances de ocorrerem falsos positivos, a nota de corte padrão do SpamAssassin foi inicialmente aumentada de 5 para 7.

Como não ocorreram problemas na fase de teste, todas as contas de *e-mail* foram migradas para o servidor com sistema anti-spam e antivírus. Para a realização desse estudo, o servidor foi monitorado por um ano. Por não ter tido casos confirmados de falso positivo, 10 meses após a implantação, a nota de corte foi baixada de 7 para 6.

A avaliação do servidor foi baseada no funcionamento adequando do sistema anti-spam e antivírus como a proporção de spam bloqueado em comparação ao resultado obtido por empresas do ramo e também com informações obtidas no questionário aplicado aos funcionários do suporte técnico. O custo, processo de instalação, estabilidade, desempenho e administração do servidor são parâmetros que também devem ser levados em consideração, pois podem inviabilizar a adoção de sistema semelhante por outras instituições.

Capítulo 2

2 Fundamentos do sistema de *e-mail*

Este capítulo tem o objetivo de apresentar os elementos envolvidos no funcionamento do sistema de *e-mail*.

Usuários interagem com o sistema de *e-mail* através de programa cliente do serviço de *e-mail* ou programa cliente de *e-mail* denominado “agente usuário” (*Mail User Agent* - MUA) como o Microsoft Outlook e o Netscape Messenger e o Eudora. O agente usuário permite criar, responder, encaminhar, apagar, visualizar e receber *e-mails*.

A composição de mensagem no agente usuário basicamente consiste em fornecer o endereço de *e-mail* do remetente e do destinatário, o assunto do *e-mail* e a mensagem.

O endereço de *e-mail* é composto por um nome de usuário ou *login*, seguido pelo símbolo “@” e um nome de domínio da Internet. O “@” em inglês é pronunciado “at”, preposição utilizada para indicar um lugar. Em comparação com o correio convencional, o domínio corresponde ao endereço completo e o *login*, ao nome do remetente ou destinatário. Por exemplo, o endereço de *e-mail* alice@zxc.com.br pode ser interpretado como a conta de *e-mail* com nome de usuário “alice” hospedada no servidor de *e-mail* do domínio “zxc.com.br”.

O agente usuário transmite o *e-mail* para o servidor SMTP (*Simple Mail Transfer Protocol*) de envio conhecido também como agente de transferência de mensagens (*Mail Transfer Agent* - MTA). Depois o *e-mail* é transmitido para o servidor SMTP de recebimento que o repassa ao agente de entrega de mensagens (*Mail Delivery Agent* - MDA), responsável pelo armazenamento da mensagem na caixa de entrada do destinatário.

O destinatário aciona o agente usuário, fornece o *login* e a senha de sua conta para receber os *e-mails* na caixa de correio ou caixa de entrada (*inbox*), através de protocolos como o POP (*Post Office Protocol*) e o IMAP (*Internet Message Access Protocol*).

2.1 SMTP (Simple Mail Transfer Protocol)

O SMTP (*Simple Mail Transfer Protocol*) é o protocolo cliente-servidor para transferência de *e-mail*. Utiliza o serviço orientado à conexão fornecido pelo TCP (*Transmission Control Protocol*) e usa a porta 25. Atualmente está definido na RFC 2821, tornando a RFC 821 obsoleta.

O protocolo funciona com comandos em texto ASCII (*American Standard Code for Information Interchange*) conforme apresentado na Tabela 1.

Tabela 1 - Comandos SMTP.

Comando	**Função**
HELO <servidor SMTP>	Identifica o cliente SMTP
MAIL FROM: <endereço de *e-mail*>	Identifica o endereço de *e-mail* do remetente
RCPT TO: <endereço de *e-mail*>	Identifica o endereço de *e-mail* do destinatário
DATA	Inicia a transmissão da mensagem
HELP <comando>	Exibe uma explicação sobre o comando
QUIT	Termina a conexão SMTP

A Figura 3 apresenta um exemplo de utilização desses comandos através de conexão ao servidor de correio com o programa “Telnet”, nativo no Windows, Unix e Linux. Os comandos digitados pelo usuário estão destacados em negrito para diferenciar das respostas do servidor.

Na primeira linha, o Telnet estabelece uma conexão com o servidor de *e-mail* na porta 25. O primeiro comando a ser executado por um cliente SMTP é o “HELO”. Serve para identificar o seu nome de domínio totalmente qualificado (*Fully Qualified Domain Name* – FQDN) ou endereço IP. Em seguida são informados o endereço de *e-mail* do remetente com o comando “MAIL FROM” e o endereço de *e-mail* do destinatário com o comando “RCTP TO”. A composição da mensagem começa com o comando “DATA” e termina com uma linha que contém apenas um “.”. A sessão é terminada com o comando “QUIT”.

As três linhas inseridas pelo cliente após o comando “DATA” contêm o endereço de *e-mail* e o nome do destinatário (“To:”) e do remetente (“From:”) e o assunto da mensagem (“Subject:”). Principalmente o endereço e o nome do remetente são freqüentemente forjados em *e-mails* de *spammers* e de código malicioso para dificultar a identificação da origem da mensagem.

```
telnet smtp.dominio.com.br 25
220 alfa.dominio.com.br ESMTP
helo dominio.com.br
250 alfa.dominio.com.br
mail from: remetente@dominio.com.br
250 ok
rcpt to: destinatario@dominio.com.br
250 ok
data
354 go ahead
To: Destinatario <destinatario@dominio.com.br>
From: Remetente <remetente@dominio.com.br>
Subject: Envio de e-mail

Este é o corpo da mensagem
Observações:
Entre o corpo da mensagem e o assunto é preciso deixar uma linha em branco.
Para finalizar uma mensagem é preciso uma linha com apenas um “.”, como na linha
abaixo.
.
250 ok 1129843425 qp 27862
quit
```

Figura 3 - Exemplo de transmissão de *e-mail* com comandos SMTP.

2.2 Formato das mensagens

O formato das mensagens é atualmente definido na RFC 2822, tornando a RFC 822 obsoleta. *E-mail* consiste em campos de cabeçalho, uma linha em branco e corpo da mensagem.

O cabeçalho é composto por várias linhas de texto ASCII com o nome do campo, um sinal de dois-pontos e um valor. As linhas que começam com espaço ou "TAB" são continuações da linha anterior.

A Tabela 2 apresenta campos de cabeçalho (TANENBAUM, 1997). Os campos "To:", "Cc:", "Bcc:", "From:" e "Received:" são relacionados com o transporte de mensagens.

Endereços de *e-mail*, nos campos "From:", "To:", "Cc:" e "Bcc:", podem ser especificados da seguinte forma:

- Apenas o endereço de *e-mail*. Ex.: beto@zxc.com.br
- Nome e sobrenome seguido pelo endereço de *e-mail* entre "<" e ">". Ex.: Roberto Otrebor <beto@zxc.com.br>
- Nome e sobrenome entre aspas seguido pelo endereço de *e-mail* entre "<" e ">". Ex.: "Roberto Otrebor" <beto@zxc.com.br>

O campo "Received:" é incluído em cada agente de transferência durante o percurso até o destinatário. A linha contém a identidade do agente, a data e a hora em que a mensagem foi recebida. Essas informações permitem reconstituir o caminho percorrido pelo *e-mail* desde o servidor de envio até o de recebimento.

Tabela 2 - Campos de cabeçalho.

Cabeçalho	**Significado**
To:	O(s) endereço(s) de correio eletrônico do(s) destinatário(s) principal(is)
Cc:	O(s) endereço(s) de correio eletrônico do(s) destinatário(s) secundário(s)
Bcc:	O(s) endereço(s) de correio eletrônico para cópias carbono cegas. Essa linha é eliminada de todas as cópias enviadas aos endereços do campo “To:” e “Cc:”.
From:	A(s) pessoa(s) que criou(aram) a mensagem
Received:	A linha que é incluída em cada agente de transferência durante o percurso
Reply-To:	O endereço de correio eletrônico para onde as respostas podem ser enviadas
Date:	A data e a hora em que a mensagem foi enviada
Subject:	Pequeno resumo da mensagem apresentado em apenas uma linha

Cabeçalhos personalizados podem ser inseridos também, desde que comecem com “X-” para não haver conflitos entre os oficiais e os particulares. O sistema anti-spam implantado inclui linhas de cabeçalho “X-Qmail-Scanner” com informações sobre a classificação do *e-mail*.

2.3 MIME (Multipurpose Internet Mail Extensions)

O SMTP suporta apenas texto ASCII de 7 bits e linhas de comprimento máximo de 1000 caracteres, resultando nas seguintes limitações:

- Não suporta conjuntos de caracteres maiores de outros idiomas;
- Não suporta transferência de dados binários.

O MIME (*Multipurpose Internet Mail Extensions*) continua a usar o formato padrão das mensagens e resolve esses problemas com técnicas de codificação que permitem o transporte de vários tipos de dados e também vários objetos numa única mensagem.

O MIME define cinco campos de cabeçalho: “MIME-Version:”, “Content-Description:”, “Content-ID:”, “Content-Transfer-Encoding:” e “Content-Type:”.

Além da versão, o cabeçalho “MIME-Version:” indica que se trata de uma mensagem MIME e não uma mensagem em texto simples escrita no idioma inglês, por exemplo, sem acentuação. “Content-Description:” é uma descrição do conteúdo codificado e “Content-ID”: a identificação do conteúdo.

“Content-Transfer-Encoding:” informa como o corpo da mensagem está codificado para transmissão. Os tipos são:

- “7 bit”: não é codificado;
- “8 bit”: não é codificado, mas as linhas não superam 1000 caracteres.
- “Binary”: não é codificado e o limite de linha de 1000 caracteres não é respeitado.
- “Quoted-printable”: trata-se apenas de um código ASCII de 7 bits, com todos os caracteres acima de 127 codificados como um sinal de igual seguido pelo valor do caractere em hexadecimal (dois dígitos).
- “Base64”: grupos de 24 bits são divididos em até quatro unidades de 6 bits, com cada unidade representada como um caractere ASCII válido. Os caracteres de 6 bits são subconjuntos do ASCII de 7 bits. A codificação é “A” para 0, “B” para 1, e assim por diante, seguida das 26 letras minúsculas, dos dez dígitos e finalmente + e / para 62 e 63, respectivamente. As seqüências == e = são usadas para indicar que o último grupo continha apenas 8 ou 16 bits, respectivamente. O comprimento máximo de cada linha deve ser de 76 caracteres.

“Content-Type:” identifica o tipo e subtipo de dado carregado na mensagem.

Os tipos são:

- “Text”: com subtipos “plain”, “richtext” e outros.
- “Image”: com subtipos “jpeg”, “gif” e outros.
- “Audio”: com subtipo “basic” codificado em “Pulse Code Modulation” (PCM)
- “Video”: com subtipo “mpeg”.
- “Application”: dado binário como “octet-stream”, “PostScript” e outros.

- “Message”: indica a compatibilidade do conteúdo da mensagem com a RFC de formato de mensagem. O subtipo “partial” permite que mensagens encapsuladas sejam divididas em várias mensagens MIME. “External-body” aponta para uma fonte externa como um objeto em um servidor FTP.
- “Multipart”: com subtipos “mixed”, “alternative”, “parallel”, “digest” e outros.

Para obter mais detalhes e informações sobre o MIME, consultar as RFCs:

- RFC 2045 - *Multipurpose Internet Mail Extensions* (MIME) *Part One: Format of Internet Message Bodies*
- RFC 2046 - *Multipurpose Internet Mail Extensions* (MIME) *Part Two: Media Types*
- RFC 2047 - MIME (*Multipurpose Internet Mail Extensions*) *Part Three: Message Header Extensions for Non-ASCII Text*
- RFC 2048 - *Multipurpose Internet Mail Extensions* (MIME) *Part Four: Registration Procedures*
- RFC 2049 - *Multipurpose Internet Mail Extensions* (MIME) *Part Five: Conformance Criteria and Examples*

2.4 POP (Post Office Protocol)

O POP versão 3, definido na RFC 1939, é um protocolo cliente-servidor baseado em TCP, cujo número de porta convencional é 110. É utilizado para transferir os *e-mails* que estão no servidor de correio para o computador local de usuários do sistema.

Dispõe de comandos, em texto ASCII como o SMTP, para o usuário se autenticar, listar, obter e excluir mensagens.

A Figura 4 apresenta um exemplo de utilização desses comandos através de conexão ao servidor de correio com o programa “Telnet” e representa o recebimento do *e-mail* enviado no item 2.1 sobre SMTP. Os comandos digitados pelo usuário estão destacados em negrito para diferenciar das respostas do servidor.

Na primeira linha, o comando telnet estabelece uma conexão TCP com o servidor POP na porta 110. O comando “user” identifica o nome do usuário. O

comando “pass” seguido da senha estabelece a autenticação do usuário. Com o comando “list”, o servidor exibe as mensagens recebidas pelo usuário. Para ler uma mensagem, basta usar o comando “retr” seguido pelo número da mensagem obtido pelo comando “list”. A conexão é finalizada pelo comando “quit”.

```
telnet pop.dominio.com.br 110
+OK <28423.1129843978@pop.dominio.com.br>
user beto
+OK
pass ********
+OK
list
+OK
1 390
.
retr 1
+OK
Return-Path: <remetente@dominio.com.br>
Delivered-To: destinatario@dominio.com.br
Received: (qmail 28416 invoked from network); 20 Oct 2005 21:31:58 -0000
Received: from maq.dominio.com.br (HELO dominio.com.br) (xxxx.xxx.xxx.xxx)
  by serv.dominio.com.br with SMTP; 20 Oct 2005 21:31:58 -0000
To: Destinatario <destinatario@dominio.com.br>
From: Remetente <remetente@dominio.com.br>
subject: Envio de e-mail

Este é o corpo da mensagem
Observações:
Entre o corpo da mensagem e o assunto é preciso deixar uma linha em branco.
Para finalizar uma mensagem é preciso uma linha com apenas um “.”, como na linha
abaixo.
.
quit
```

Figura 4 - Exemplo de recebimento de *e-mail* com o uso de comandos do protocolo POP.

Esse protocolo está presente em clientes de *e-mail* como o Microsoft Outlook, Netscape Messenger e Eudora. Os clientes de *e-mail* devem ser configurados com o endereço do servidor POP, que geralmente está no formato pop.domínio. Mediante a apresentação do nome do usuário e da senha, é possível fazer o *download* das novas mensagens.

O *software* cliente pode ser configurado para procurar por novas mensagens no servidor periodicamente ou quando o usuário solicitar. Outra opção é a possibilidade de manter ou excluir os *e-mails* do servidor após o *download*. As mensagens mantidas no servidor podem ser acessadas do local de trabalho e da residência do usuário.

2.5 IMAP (Internet Message Access Protocol)

O IMAP versão 4, 1ª revisão, definido na RFC 3501, é um protocolo cliente-servidor baseado em TCP, cujo número de porta convencional é 143. Esse protocolo, assim como o POP, é implementado nos programas clientes mais populares para acessar os *e-mails* dos usuários contidos em suas caixas de entrada do servidor de *e-mail*.

O IMAP é recomendado aos usuários que acessam os *e-mails* de diversos computadores, pois os *e-mails* ficam armazenados como num servidor de arquivos, permitindo o gerenciamento de *e-mails* de qualquer computador conectado na Internet. O usuário pode administrar os *e-mails* localmente e depois sincronizar com o servidor. O POP, por outro lado, é bem mais limitado, pois só permite fazer o *download* de todas as mensagens e depois apagá-las ou mantê-las no servidor.

2.6 Relay

O sistema de correio eletrônico surgiu na década de 70 e foi utilizado, inicialmente, por pesquisadores de universidades norte-americanas. Nessa época, o endereço era composto pela rota completa até o servidor de destino. Nessa arquitetura cada nó fazia o encaminhamento (*relay*) da mensagem para o próximo até a mensagem chegar ao seu destinatário.

O endereço de *e-mail* seguia a "*bang path notation*". Os nomes das máquinas eram separados pelo sinal de exclamação, pois "*bang*" significa exclamação no inglês coloquial. Por exemplo, o endereço bigsite!foovax!barbox!bob envia o *e-mail* para uma máquina conhecida, nesse caso a "bigsite", que depois encaminha para "foovax", que encaminha para a máquina de destino "barbox" colocar a mensagem na caixa de entrada do usuário "bob".

Com o crescimento da Internet, os servidores de *e-mail* passaram a ser explorados para o envio de spam, pois aceitavam *e-mails* de qualquer lugar do mundo e os encaminhavam para os servidores de *e-mail* de destino.

Essa situação criou a necessidade de restringir as pessoas autorizadas a enviar um *e-mail* para outro domínio através de um determinado servidor de *e-mail*. Antes de aceitar um *e-mail* para entrega remota, o servidor deve verificar, por exemplo, se o endereço IP do computador cliente está autorizado. Os servidores que não restringem o encaminhamento de *e-mails* para servidores remotos são denominados *open relays* ou servidores com o *relay* aberto.

Os servidores com o *relay* aberto devem ser atualizados ou configurados para encaminhar apenas os *e-mails* de computadores autorizados. Normalmente a restrição é feita por endereços IP, mas pode ser feita também ao exigir autenticação do usuário pelo SMTP-AUTH, apresentado no próximo item, para permitir o envio de *e-mails*.

2.7 SMTP-AUTH

O SMTP-AUTH, definido na RFC 2554, é uma extensão do SMTP criada para incluir a autenticação de usuário no processo de envio de *e-mail*, permitindo que apenas usuários autenticados enviem *e-mail* pelo servidor.

Esse sistema, apesar de parecer mais seguro, tem uma fraqueza. Se usuários autenticados possuírem permissão para enviar *e-mails* a partir de qualquer endereço IP, então um *hacker* ou *spammer* pode realizar ataques de força bruta para tentar descobrir o *login* e a senha de algum usuário. Portanto, para tornar o sistema mais seguro, deve existir uma política de senhas de *e-mail* para dificultar esse tipo de ataque.

2.8 Armazenamento das mensagens no servidor

Existem dois métodos para armazenar os *e-mails* dos usuários no servidor:

- Mbox;
- Maildir.

O formato “mbox” é bastante conhecido, pois é utilizado pelo Sendmail, um dos MTAs mais populares. Todos os *e-mails* de cada usuário são armazenados, em suas respectivas contas de *e-mail*, num único arquivo.

O formato “maildir” é relativamente novo, mas já está presente em alguns MTAs como o Qmail e o Postifix. Nesse formato, cada *e-mail* é armazenado como um único arquivo.

Nesse formato, na área do usuário, há um diretório “Maildir” com os subdiretórios “new”, “cur” e “tmp”. Um *e-mail* é primeiro armazenado temporariamente no subdiretório “tmp”. Depois que o sistema consegue escrever a mensagem com sucesso dentro desse subdiretório, copia a mensagem para o “new” e depois a exclui do “tmp”. Mensagens já lidas e mantidas no servidor são movidas para o diretório “cur” (TELLES, 2005).

A desvantagem do formato mbox é o risco de se perder mensagens ou corromper o arquivo se houver uma falha do servidor durante a manipulação do único arquivo da caixa de entrada. É necessário também um sistema de *lock* para o acesso concorrente, ou seja, enquanto o MTA manipula o arquivo o servidor POP não pode acessá-lo.

As desvantagens do maildir são o consumo de mais inodes[1] no sistema de arquivos e a necessidade de se obter *software*s capazes de acessar esse formato. O programa cliente de *e-mail* Pine, por exemplo, precisa ser recompilado para reconhecer esse formato.

[1] Num sistema operacional baseado em UNIX, um inode é uma descrição de um arquivo individual do sistema de arquivo UNIX (TECHTARGET, 2003).

Capítulo 3

3 Spam

Acredita-se que o nome spam foi dado aos *e-mails* de propagadas de produtos e serviços a partir de um episódio de um esquete do grupo Monty Python (HORMEL FOODS CORPORATION, 200?) de dezembro de 1970. Nesse episódio (LOWE, 200?b), um cliente está num restaurante e pergunta a uma garçonete quais são as opções do cardápio. Todas as opções incluíam o spam, comida enlatada muito famosa da Hormel Foods Corporation, principalmente, pelo seu consumo por soldados durante a segunda guerra mundial. O cliente não queria spam, mas mesmo assim tudo tinha spam. Analogamente, estamos nos acostumando a ter sempre *e-mail* com spam.

Acredita-se que o primeiro spam (LOWE, 200?a) foi enviado por um funcionário da Digital Equipment Corporation, em maio de 1978, para usuários de *e-mail* da ARPANET (*Advanced Research Projects Agency Network*). Tratava-se de uma apresentação da nova linha de equipamentos da família DECSYSTEM-20.

3.1 O que é spam?

Propagandas de produtos para melhora do desempenho sexual e perda de peso, produtos de beleza, relógios, *softwares* mais baratos, cursos, diplomas universitários e esquemas para ganhar dinheiro fácil são exemplos de spam.

UCE (*Unsolicited Commercial E-mail*), UBE (*Unsolicited Bulk E-mail*) e *junk e-mail* são outros nomes dados ao spam. A sigla UCE refere se aos *e-mails* comerciais; UBE, ao envio de *e-mails* em massa e *junk e-mail*, aos *e-mails* inúteis ou lixo eletrônico.

O Grupo Brasil Anti-Spam (200?) descreve o spam da seguinte forma:

> a) Não existe a identificação do remetente, ou ele é falso;
> b) Você não autorizou que aquele remetente lhe enviasse mensagens;
> c) Não lhe foi oferecida a opção de não receber mais mensagens do mesmo remetente;
> d) Abordagem enganosa - tema do assunto da mensagem é distinto de seu conteúdo de modo a induzir o destinatário em erro de abertura na mensagem;
> e) Ausência da sigla "NS" (Não Solicitado) no campo Assunto, quando a mensagem não houver sido previamente solicitada;
> f) Envio de mensagens com o mesmo conteúdo e para o mesmo destinatário, variando apenas o remetente ou o assunto, em prazo inferior a dez dias.

Segundo o Movimento Anti-Spam brasileiro (200?) “spam” é:

> “A definição de spam do ponto de vista prático é o envio abusivo de correio eletrônico não solicitado em grande quantidade distribuindo propaganda, correntes e esquemas de ‘ganhe dinheiro fácil’. É o envio de correio tentando forçar a leitura pela pessoa que recebe que outroura [sic] sequer optou por este recebimento. Um desperdício de recursos da rede pago por quem recebe”.

PAIVA (2002) classifica o spam como um “Abuso no Correio Eletrônico” (ACE) e o define da seguinte forma:

> "[O] correio eletrônico não solicitado ou não desejado encaminhado a um grande número de usuários com o objetivo de divulgar promoções comerciais ou a proposição das mais diversas idéias"

A TechTarget (2005b) compara o “spam” da seguinte forma:

> “Spam é quase equivalente a ligações telefônicas de marketing não solicitadas, exceto que o usuário paga parte da mensagem já que todos compartilham o custo da manutenção da Internet”. (tradução)

Com as informações apresentadas anteriormente podemos descrever o spam como:

- Um meio de fazer propaganda;
- Geralmente enviando em massa (uso abusivo do sistema de correio eletrônico);
- Não foi solicitado ou autorizado previamente e o remetente é uma pessoa desconhecida;
- Geralmente não há identificação completa do remetente ou as informações são falsas;
- O assunto do *e-mail* induz o usuário a abrir a mensagem, porém o conteúdo não possui relação com o que está descrito.

3.2 Problemas causados pelo spam

A perda de produtividade é um dos grandes problemas criados pelo spam. As pessoas perdem tempo para separar as mensagens legítimas do lixo eletrônico. Algumas mensagens são facilmente identificadas como irrelevantes e são descartadas imediatamente. Outras, através de técnicas de engenharia social, convencem as pessoas a abri-las, resultando em distração e mais tempo despendido.

O aumento de *e-mails* de origem ou de conteúdo duvidoso, como propagandas, golpes, boatos e vírus, prejudica a equipe de informática. Cada vez mais, o suporte técnico desperdiça tempo atendendo a solicitações de usuários para esclarecer dúvidas ou resolver problemas relacionados a essas mensagens.

A capacidade de armazenamento de informações em computadores é limitada. Por isso, normalmente é definida a quantidade máxima de espaço em disco rígido que os arquivos de usuários podem ocupar no servidor. As pessoas que não acessam a caixa postal freqüentemente podem ficar com essa área em disco, definida por uma quota, esgotada. Ou seja, além de o spam ocupar o espaço em disco do servidor pago pela instituição, ainda pode impedir que mensagens legítimas sejam recebidas.

A enxurrada de spam que passa pelas redes dos provedores aumenta a latência da rede, e prejudica os seus usuários. Conseqüentemente, o provedor pode ser

forçado a ampliar a capacidade dos equipamentos para aumentar a largura de banda de rede e repassar o custo do investimento aos usuários. Semelhantemente, as entidades que possuem servidores de *e-mail* também podem ser forçadas a aumentar a capacidade dos servidores e da rede.

Além do prejuízo material, o spam também pode expor crianças e adolescentes a materiais impróprios como propagandas para o público adulto com imagens eróticas e pornográficas. Para algumas pessoas, dependendo do país, religião ou cultura, esse material pode ser considerado ofensivo.

E-mails forjados com técnicas de engenharia social podem convencer as pessoas a fazerem *download* e a executarem arquivo com código malicioso ou a inserirem informações confidenciais como número de conta bancária e senha em *sites* controlados por golpistas.

Para piorar a situação, código malicioso utiliza o serviço de *e-mail* para se multiplicar e infectar outros computadores, onerando ainda mais a rede e o servidor.

3.3 Razão para enviar spam

O spam serve para promover produtos e serviços e conseguir clientes. Como apenas uma pequena porcentagem dos *e-mails* atinge o objetivo, quanto maior a quantidade, maior o retorno.

Uma reportagem da revista Wired News (MCWILLIAMS, 2003) informou que num período de quatro semanas, 6000 pessoas responderam a anúncios enviados por *e-mail* sobre a venda de produtos para aumentar o pênis. A maioria dos clientes pediu dois frascos pelo preço de $50 cada. Essas informações foram obtidas através de um *log* de pedidos de um servidor *web* com falhas de segurança. Com essas informações pode-se ter uma idéia de como o envio de spam pode ser lucrativo.

3.4 Tipos de e-mails indesejados

Além do spam, existem outros tipos de *e-mail* indesejados:

- Golpe (*phishing scam*);
- Corrente;
- *E-mail* com código malicioso;
- Boato (*hoax*);
- *Mail bomb*;
- Retorno de spam enviado a endereços inexistentes;
- Notificações de bloqueio de *e-mail* com código malicioso.

3.4.1 Phishing scam

Os *phishing scams* são *e-mails* forjados para parecer que foram enviados por instituições financeiras (ANEXO A1) para obter informações pessoais, número de conta bancária e senha. "*Scam*" significa golpe e "*phishing*", segundo a TechTarget (2005a), é uma corruptela de "*fishing*", palavra em inglês que significa pescaria. Metaforicamente, o *e-mail* do golpe é a isca e as pessoas que o recebem, os peixes. Segundo o relatório do *Anti-phishing working group* (APWG, 2004) 5% das pessoas que recebem o "*phishing scam*" acabam respondendo a esse tipo de *e-mail*.

Os golpes podem ser aplicados da seguinte forma:

- O *Site* de uma instituição financeira é copiado e alocado num computador controlado pelo golpista;
- Um *e-mail* forjado para se parecer com um comunicado oficial da empresa é enviado para milhares de pessoas;
- A vítima, convencida por técnicas de engenharia social, entra no *site* controlado pelo golpista e coloca informações pessoais, o número da conta e a senha.

Nessa categoria encontra-se também o "*Nigerian 419*[1] *scam*" (ANEXO A2). Existem versões diferentes, mas muito semelhantes entre si.

[1] 419 se refere à seção relevante do código criminal nigeriano.

O golpista envia milhares de *e-mails*. Ele finge que é representante de uma instituição financeira da Nigéria e afirma que um correntista milionário faleceu sem deixar herdeiros. Para conseguir sacar o dinheiro, precisa de uma pessoa que se apresente como parente do correntista. Como motivação, ele oferece uma porcentagem da herança. A vítima convencida de que conseguirá um bom retorno se oferece como voluntária. Mas, o golpista alega que para efetuar a transação, precisa de apoio financeiro. A vítima fornece dinheiro para o golpista até perceber que jamais receberá parte da herança.

3.4.2 Corrente

As correntes ou *chain letters* (ANEXO A3) influenciam o usuário supersticioso a enviar várias cópias da mensagem recebida para os seus conhecidos para ter sorte ou para que nada de ruim aconteça.

Quando os *e-mails* de correntes são encaminhados, os endereços de *e-mail* das pessoas que haviam recebido essa mensagem anteriormente são repassados também, tornando-se uma rica fonte de endereços de *e-mail* para vírus, *worms* e *spammers*.

3.4.3 *E-mail* com código malicioso

Programas especificamente desenvolvidos para executar ações maliciosas em um computador são conhecidos como código malicioso ou *Malware* (***Mal****icious Soft****ware***) (CERT.BR, 2005). Vírus, *worms* e cavalos de tróia (*trojan*) são exemplos de código malicioso.

Vírus de computador é um programa que, ao infectar um computador, insere ou anexa o seu código em programas, setores de boot, setores de partição ou documentos que suportam macros. Alguns vírus apenas exibem mensagens ou

imagens inofensivas, mas podem também destruir arquivos, formatar o disco rígido ou causar outros danos.

Os vírus, por muitos anos, foram disseminados por disquetes com o setor de boot ou arquivos contaminados. No entanto, nos últimos anos, passaram a atingir outros computadores usando as redes de computadores, por exemplo, explorando compartilhamentos da rede local e o serviço de *e-mail*.

Worm é um programa que facilita a propagação de cópias funcionais de si mesmo para outros computadores ao explorar vulnerabilidades presentes nos *softwares* de computadores.

Vírus e *worms* podem ser diferenciados pela capacidade do vírus inserir código malicioso em arquivos do usuário ou do sistema operacional. O *worm*, por outro lado, não é capaz de infectar arquivos. Além disso, os vírus não são capazes de replicar sem a ação humana, por exemplo, ao executar um arquivo com código malicioso (WEBOPEDIA, 2004). Os *worms*, diferentemente dos vírus, são capazes de se propagar sem o auxílio de pessoas ao explorar falhas, por exemplo, do sistema operacional.

A replicação de um vírus ou *worm* freqüentemente implica em centenas ou até milhares de cópias espalhadas na Internet, podendo atingir o mundo todo rapidamente. O desempenho da rede de instituições é afetado e os servidores de *e-mail* podem ficar congestionados, caso a replicação dependa desse meio.

Vírus e *worms*, que se replicam por *e-mail*, enviam cópias de si mesmos na forma de arquivo em anexo para atingir outros computadores. Os campos remetente e destinatário são freqüentemente preenchidos com endereços de *e-mail* contidos no catálogo de endereço do programa cliente de *e-mail* ou em *e-mails* recebidos anteriormente pelo usuário do computador.

Como o campo remetente não é necessariamente preenchido com o endereço de *e-mail* do usuário do computador, é difícil de identificar o computador que está realmente infectado e tomar as devidas providências. Nessa situação, as informações de cabeçalho são muito importantes porque contêm o endereço IP de origem.

No exemplo apresentado na Figura 5, o computador de Eva está infectado. O vírus ou *worm* enviou uma cópia de si mesmo para Alice, pois o seu endereço consta no catálogo de endereços de Eva. O endereço de *e-mail* de Beto, que também está no catálogo de endereços de Eva, foi colocado como remetente. Ao abrir o *e-mail*, o antivírus do computador de Alice acusa a presença de código malicioso. Alice pensa que o *e-mail* foi enviado pelo computador de Beto e imediatamente solicita que ele tome providências. Enquanto isso, o computador de Eva continua a disseminar o vírus ou *worm* e o suporte técnico da empresa em que o Beto trabalha recebe um chamado técnico.

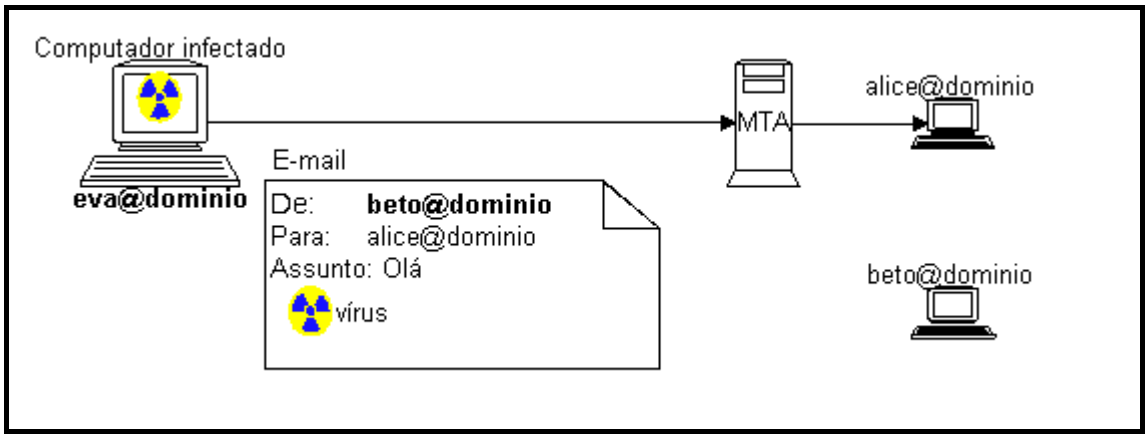

Figura 5 - Técnica de engenharia social aplicada em código malicioso enviado por *e-mail* para enganar os usuários.

O vírus Klez.H@mm (SYMANTEC, 2004b), por exemplo, procura por endereços de *e-mail* no catálogo de endereços do Windows e em arquivos com as extensões TXT, HTM, HTML, WAB, ASP, DOC, RTF, XLS, JPG, CPP, C, PAS, MPG, MPEG, BAK, MP3 e PDF. Os endereços encontrados servem para forjar remetentes e destinatários. Outros vírus que agem dessa maneira são: Sobig.C, Bugbear.B, Sobig.B, Fizzer.A, Ganda.A, Gibe.B e Yaha.P. (SIMPSON, 2003)

Em abril de 2000, o *worm* "*LoveLetter*" se espalhou em seis horas em escala global, por *e-mail*, causando danos estimados em sete bilhões de dólares (MCAFEE, 2001). Com o uso de técnica de engenharia social, os usuários foram encorajados a executar o código malicioso encontrado em anexo. Quando era ativado, o *worm* sobrescrevia com o seu código todos os arquivos com a extensão .doc, .txt, .jpeg, .gif, .wav, .avi, .mpg, .mp3. Para replicar, *e-mails* com o *worm* eram enviados para endereços contidos no catálogo de endereços do computador infectado.

O cavalo de tróia se disfarça em programas de computador aparentemente inofensivos como jogos, animações e protetores de tela para roubar informações, destruir arquivos ou tornar o computador vulnerável a ataque de *hackers*. Diferentemente dos vírus e *worms*, o cavalo de tróia não é capaz de se replicar.

PWSteal é o nome genérico dado pela Symantec para os vários tipos de cavalos de tróia que tentam coletar nomes de usuário e senhas do computador infectado. Geralmente, um *e-mail* criado com técnica de engenharia social é enviado para uma grande quantidade de pessoas. Através de um *link* presente no corpo do *e-mail*, o cavalo de tróia é salvo no computador para depois ser executado. Como ele não é anexado ao *e-mail*, os computadores de usuários precisam de *software* antivírus instalado e atualizado para detectá-lo.

A detecção e o tratamento de vírus, *worms* e cavalos de tróia pelo *software* antivírus depende do desenvolvimento de vacinas específicas para cada tipo de ameaça. Essas vacinas são distribuídas em arquivos transferidos durante atualizações, que devem ser realizadas regularmente para manter o *software* antivírus capaz de lidar com as ameaças mais recentes.

3.4.4 Boato

Boato ou *hoax* (ANEXO A4) explora a falta de conhecimento dos usuários para se alastrar na Internet. Geralmente se trata de um *e-mail* sobre um novo vírus

que não pode ser detectado por nenhum antivírus, capaz de destruir o computador. Sem saber que é um boato, os usuários encaminham o *e-mail* para todos os seus conhecidos, acreditando que estão fazendo algo útil.

Assim como as correntes, os boatos carregam os endereços de *e-mail* das pessoas que os receberam, tornando-se também uma grande fonte de endereços de *e-mail* para vírus, *worms* e *spammers*.

3.4.5 *Mail bomb*

Entende-se por *mail bomb* o envio de grande quantidade de *e-mails* ou de poucos *e-mails* com anexos relativamente grandes para uma pessoa específica com a finalidade de ocupar todo o espaço em disco da cota. Isso impede que o destinatário receba novas mensagens.

Em alguns casos pode até sobrecarregar o servidor, impedindo o seu funcionamento como num ataque de negação de serviço. No passado, as *mail bombs* eram usadas para punir usuários da Internet que violavam as regras de “etiqueta da Internet”, como os *spammers*.

Uma forma mais recente de *mail bomb* é o cadastramento do *e-mail* da vítima em listas de distribuição de spam.

3.4.6 Retorno de spam enviado a endereços inexistentes

Para dificultar a sua identificação ou rastreamento, os *spammers* normalmente enviam os *e-mails* com o endereço de remetente forjado. Quando o endereço do destinatário não existe, o *e-mail* retorna para o endereço de remetente forjado que pode até ser de um domínio existente.

A Figura 6 exibe uma simulação em que *spammer* envia um *e-mail* cujo destinatário não existe, e que retorna para um endereço forjado. Nesse caso, o domínio desse endereço existe, porém o usuário não. Então o administrador do servidor de *e-mail* desse domínio recebe a mensagem de erro.

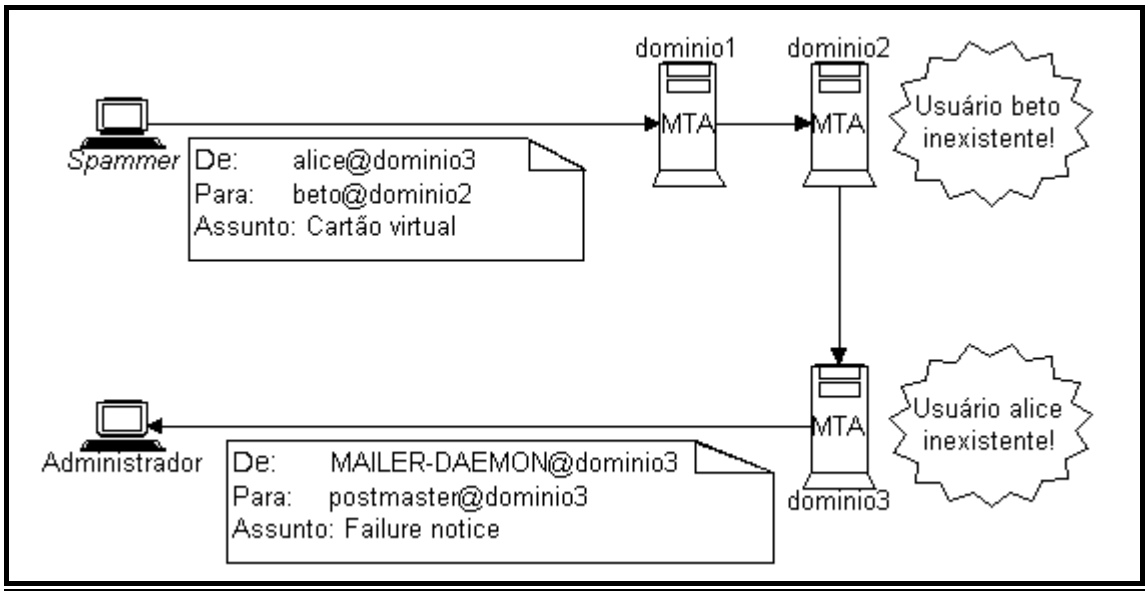

Figura 6 - Exemplo de falha na entrega de spam enviado para endereço de remetente e de destinatário forjados.

Eventualmente, o endereço de remetente forjado realmente existe e o usuário recebe a mensagem de erro, informando que o endereço do destinatário não existe.

3.4.7 Notificação de bloqueio de *e-mail* com código malicioso

O último tipo de *e-mail* indesejado, porém não menos importante, é a notificação de bloqueio de *e-mail* com código malicioso (ANEXO A5). A propagação de código malicioso por *e-mail* tem forçado as empresas a investirem em *software* antivírus no servidor de *e-mail*. Ao bloquear algum *e-mail* com código malicioso, o remetente é informado sobre o ocorrido. Infelizmente, conforme visto no item 3.4.3 sobre *e-mail* com código malicioso; geralmente o endereço do

remetente não é o do usuário do computador, mas é obtido do catálogo de endereços do computador infectado. Então as notificações são geralmente enviadas para as pessoas erradas, que acabam recebendo mais lixo eletrônico nas caixas postais.

No exemplo da Figura 7, o computador de Eva está infectado. O vírus envia um *e-mail* com código malicioso para Alice, que está cadastrada no catálogo de endereços de Eva, mas no campo "remetente" coloca o endereço de Beto e não o de Eva. Ao receber a mensagem, o servidor de *e-mail* identifica que a mensagem possui um vírus e envia um *e-mail* de notificação de bloqueio de *e-mail* com vírus para o remetente. Beto recebe a notificação e acha que o seu computador está infectado. Por isso, chama o suporte técnico para ajudá-lo. Além de a notificação confundir o Beto, acaba gerando mais tráfego de *e-mail* desnecessário.

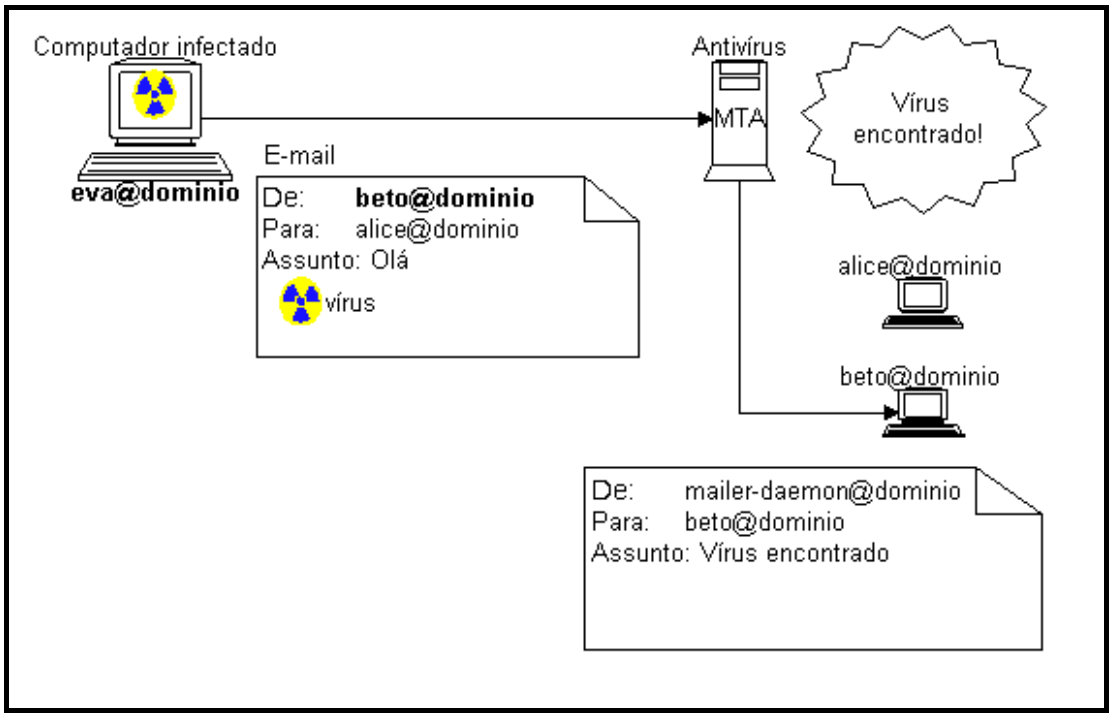

Figura 7 - Notificação de bloqueio de *e-mail* com código malicioso enviada para o usuário errado.

Nas empresas em que o servidor não possui *software* antivírus, os usuários recebem código malicioso por *e-mail* e ficam na total dependência de um antivírus instalado e atualizado em seu computador de trabalho.

3.5 Táticas para o envio de spam

O envio de spam pode ser realizado com o uso de:

- Contas de *e-mail* gratuitas;
- Servidor de *e-mail* com o *relay* aberto;
- Servidor *proxy* aberto;
- Computadores de *cyber*-cafés, bibliotecas e hotéis;
- Redes locais sem fio mal-configuradas;
- Computadores comprometidos por código malicioso.

3.5.1 Contas de *e-mail* gratuitas

A criação de contas de *e-mail* gratuitas, atualmente, é extremamente simples. O cadastro é feito pela Internet com informações sobre o solicitante que nem sempre são verdadeiras. As contas são usadas para o envio de spam até que o provedor receba reclamações e as cancele.

Para mitigar esse problema a Microsoft, no MSN Hotmail, criou um esquema de cotas permitindo que cada usuário envie até 100 mensagens diariamente (MICROSOFT, 2003).

3.5.2 Servidores de *e-mail* com o *relay* aberto

Servidores de *e-mail* com o *relay* aberto são alvos freqüentes de *spammers*. Os servidores explorados podem ficar sobrecarregados e consomem a largura de banda da rede. Além disso, a utilização de servidores de terceiros torna o processo de identificação do responsável pelo envio do spam mais difícil.

A tendência é que os servidores com os *softwares* desatualizados ou mal configurados diminuam. Mas, há vírus, *worms* e cavalos de tróia que podem transformar computadores de usuários em servidores de *e-mail* com o *relay* aberto. O

código malicioso associado ao crescente acesso à Internet por banda larga ainda mantém esse meio de envio de spam bastante presente nos dias atuais.

3.5.3 Servidor *proxy* aberto

Um servidor *proxy* atua como intermediário entre usuários e a *web* para tornar a navegação mais eficiente. As páginas acessadas pelos usuários são armazenadas no servidor para que possam ser acessadas localmente caso sejam requeridas novamente. Esse mecanismo, além de melhorar o tempo de resposta de páginas freqüentemente acessadas, diminui o tráfego na Internet.

Os servidores *proxy* corporativos devem ser configurados para atender apenas as requisições oriundas de computadores da própria organização para evitar o mau uso do servidor por terceiros e também pelo uso de recursos da empresa como o servidor de *e-mail* e a largura de banda de rede. Um servidor *proxy* mal-configurado, que atende requisições de qualquer pessoa conectada na Internet é conhecido como *open proxy* ou servidor *proxy* aberto.

Um servidor *proxy* aberto pode, por exemplo, ser explorado por um *spammer* para se conectar anonimamente a um servidor de correio. Então, qualquer *e-mail* enviado pelo *spammer* parecerá ter sido originado pelo servidor *proxy*.

A reputação de empresas pode ser afetada por possuir servidor *proxy* aberto e estar relacionada com o envio de spam. Além disso, *e-mails* da empresa poderão ser bloqueados se o servidor de *e-mail* for cadastrado em listas negras.

3.5.4 Computadores de *cyber*-cafés, bibliotecas e hotéis

Locais com bastante circulação de pessoas e que fornecem acesso à Internet, estão sujeitos à má-utilização de seus equipamentos. Os *cyber*-cafés, bibliotecas e hotéis, por exemplo, devem aplicar políticas para impedir que pessoas da rede interna utilizem os equipamentos para o envio de spam.

3.5.5 Redes locais sem fio mal-configuradas

As redes locais sem fio "*Wireless Local Area Network*" (WLAN), conhecidas também como Wi-Fi ("*Wireless Fidelity*" - fidelidade sem fios) estão se tornando cada vez mais populares.

A mobilidade propiciada pela tecnologia resulta na necessidade de aumentar a segurança para impedir o abuso de *hackers* e *spammers*.

3.5.6 Computadores comprometidos por código malicioso

Adolescentes aficionados por informática criavam vírus para exibir as suas habilidades e ganhar fama entre os seus colegas. O envio de spam, por outro lado, esteve sempre ligado ao lado material, principalmente para a obtenção de recursos financeiros. Tratava-se de problemas separados.

Atualmente, existe código malicioso que transforma computadores de usuários em servidores de *e-mail* com o *relay* aberto ou servidores *proxy* abertos. Os computadores comprometidos também podem ser usados para executar ataques de negação de serviço (*Denial of Service* - DoS).

Paul Wood (2004?, p. 4), analista de segurança da informação, da MessaLabs, afirma que existe uma correlação entre o spam e os vírus interceptados pela empresa,

pois cerca de dois terços dos *e-mails* analisados são enviados através de servidores *proxy* abertos. Vírus como o Bugbear e o SoBig foram desenvolvidos para funcionar como servidores *proxy* abertos e permitem o acesso aos computadores infectados por *backdoor*.

Nas enciclopédias de vírus encontradas nos *sites* de empresas de antivírus é possível fazer a busca por vírus relacionados com o spam. Segundo a Symantec no dia 18.05.2004 foi descoberto o *worm* W32.Bobax.D ou Bloodhound.Packed (SYMANTEC, 2005b) capaz de tornar o computador infectado em um servidor de *e-mail* com o *relay* aberto.

Existem relatos na literatura também de vírus e *worms* que tornam computadores comprometidos em servidores *proxy* abertos como os *worms* W32.Mydoom.BI@mm (SYMANTEC, 2005c) e W32.Sobig.F@mm (SYMANTEC, 2004a).

3.6 Criação e manutenção da base de e-mails

A base de *e-mails* utilizada pelos *spammers* pode ser obtida ou construída da seguinte forma:

- Aquisição de kit com *software* para envio de *e-mail* em massa e milhares de endereços de *e-mail*
- *Spiders*, *crawlers* e *bots* na *web*, *chats* e *newsgroups*
- *Sites* que oferecem serviços por *e-mail*
- Endereços de *e-mail* encontrados em correntes e boatos
- DHA (*Directory Harvest Attack*)
- Formulários com a opção para aceitar propagandas pré-selecionada

3.6.1 Aquisição de kit com milhares de endereços de *e-mail*

A obtenção dos endereços de *e-mail* não é mais uma tarefa árdua para os iniciantes no “negócio” já que existem kits prontos em CDs piratas ou vendidos por *spammers*. Esses kits já fornecem milhares de *e-mails*, assim como o *software*.

3.6.2 *Spiders*, *crawlers* e *bots* na *web*, *chats* e *newsgroups*

Os serviços disponíveis na Internet são uma fonte inesgotável de endereços de *e-mail*. Baseando-se nisso, os *spammers* utilizam *spiders* ou *bots* (robôs) para vasculhar as páginas da *web*, *chats* e *newsgroups* em busca de endereços de *e-mail*.

Uma pesquisa da *Federal Trade Commission* (FTC, 2003) constatou que 86% dos endereços publicados na *web*, durante o estudo, receberam spam.

3.6.3 *Sites* que oferecem serviços por *e-mail*

Alguns *sites* oferecem serviços por *e-mail* como é o caso do envio de cartões virtuais e de imagens de pornografia. De modo geral é difícil determinar se o uso do endereço é somente para aquele solicitado pelo usuário ou se é utilizado para o envio spam.

3.6.4 Endereços de *e-mail* encontrados em correntes e boatos

Muitos usuários, ao encaminhar as correntes (*chain letter*) e boatos (*hoax*), não removem o cabeçalho presente no *e-mail*. Esse processo pode se repetir por inúmeras vezes, fazendo com que esses *e-mails* circulem pela Internet com os endereços de *e-*

mail de centenas de pessoas. Eventualmente, esses *e-mails* podem chegar até os *spammers* com todos os endereços de *e-mail*.

3.6.5 DHA

O DHA ("*Directory Harvest Attack*") é parecido com o ataque de dicionário. O *spammer* cria uma listagem de possíveis nomes de usuários e de domínios. Ao juntar cada nome de usuário com o domínio, obtém-se um endereço de *e-mail*. Para descobrir quais endereços são válidos, o *spammer* envia mensagens para os endereços de *e-mail* gerados. Se receber mensagem de erro, significa que o endereço não existe. Os demais são os endereços de *e-mail* válidos.

3.6.6 Formulários com a opção para permitir o envio de propaganda pré-selecionada

A atenção deve estar redobrada ao preencher formulários on-line. Em algumas ocasiões existe a possibilidade de aceitar ou não o recebimento de propagandas. Essa opção já pode estar, por padrão, configurada para consentir o envio de mala-direta e geralmente está no final do formulário de modo que muitas pessoas acabam passando por ela sem perceber.

3.7 Controle das contas ativas

Um endereço de *e-mail* verdadeiro não garante que o usuário acesse a conta freqüentemente ou que ele abra o spam. Para fazer o controle de contas ativas, os *spammers* utilizam os seguintes recursos:

- Solicitação para o descadastramento da lista de distribuição;
- Compra de produtos divulgados por spam;
- *Web bugs*.

3.7.1 Solicitação para o descadastramento da lista de distribuição

No rodapé do spam, às vezes aparecem instruções para o usuário se descadastrar da lista de distribuição. Freqüentemente é solicitado ao usuário para enviar um *e-mail* com o assunto “remover”. Mas, o endereço colocado no spam nem sempre é verdadeiro ou o *e-mail* enviado serve apenas como um indício de que o usuário acessa a conta com freqüência.

Outro recurso até mais eficiente, baseado na solicitação de descadastramento, é a inserção de um *link* com parâmetros sobre o endereço de *e-mail*. Quando o usuário clica no *link*, o seu acesso é cadastrado num banco de dados.

3.7.2 Compra de produtos divulgados por spam

Pior do que o pedido de descadastramento é a compra de produtos divulgados por spam. Isso incentiva o envio de spam e também aumenta as informações que os *spammers* possuem sobre a pessoa.

3.7.3 *Web bugs*

O controle via *web bug*, um tipo de *spyware*, é bastante eficiente, pois basta que o usuário abra o spam para que o *spammer* saiba que o *e-mail* foi acessado. No corpo da mensagem é adicionada uma referência a uma imagem de 1x1 pixel hospedada em um servidor *web* controlado pelo *spammer*. Quando o usuário acessa a mensagem, a imagem de tamanho praticamente imperceptível é carregada pelo cliente e uma entrada nos *logs* do servidor é adicionada. Na referência à imagem pode haver um código identificando o *e-mail* da vítima. Veja um exemplo de *web bug*:

```
<img width=”1” height=”1”
src="http://www.spammerserver.domain/img.jpg?email=vitima@domain.com">
```

3.8 Técnica de engenharia social para dificultar a identificação do spam

Os *spammers* sabem que não são todas as pessoas que abrem o spam. Ao forjar o remetente e o assunto do *e-mail* (Tabela 3) como se a mensagem fosse um cartão virtual, uma piada ou de uma empresa bastante conhecida, eles garantem que mais pessoas abram os seus *e-mails* antes de excluí-los.

Tabela 3 - Exemplos de *e-mails* forjados com técnicas de engenharia social.

De:	**Assunto:**
lovecards	De Alguém Para Voce!!
OCarteiro.com.Br	Felicidades!!!
Cartoesbol	Voce recebeu um cartão BOL
Humor Tadela	baixe e leia com atencao!!!
ReceitaFederal	CPF Cancelado ou Pendente de Regularização
SERASA!!!!	SERASA!!!!
embratel21	Debito com a Embratel21!!!
Security	Symantec-Atualização de segurança

3.9 Marketing por e-mail

Um assunto bastante discutido é a possibilidade de se fazer marketing por *e-mail*. Apesar de ainda não existir um código de conduta unificado, algumas empresas oferecem a possibilidade de o usuário optar por receber ou não mala-direta. Essa prática é conhecida como "*opt-in/opt-out*".

Para a regulamentação do marketing por *e-mail*, foram propostas as seguintes medidas:

- Alterar o assunto dos *e-mails* para facilitar a identificação;
- Proibição de informações falsas ou vagas a respeito do conteúdo do *e-mail*;
- Incluir informações para o *opt-out*.

Capítulo 4

4 Recomendações para o uso do *e-mail*

Enquanto não há uma solução definitiva para manter o spam afastado das caixas postais, é preciso conscientizar os usuários como método de prevenção. “Álvaro Teófilo, *Security Officer* da Caixa Seguros, diz que campanhas de divulgação, políticas de bom uso do correio eletrônico e treinamento são algumas das armas para evitar a proliferação do spam” (MÓDULO SECURITY, 2003).

As seguintes recomendações devem ser passadas aos usuários para minimizar as chances de receber *e-mails* indesejados.

Usar o *e-mail* profissional apenas para assuntos relacionados ao trabalho. Para o uso pessoal e outras atividades, contas de *e-mail* descartáveis devem ser criadas. É muito mais problemático e desagradável trocar um endereço de *e-mail* profissional e avisar todos os clientes do que trocar um descartável.

Fazer cadastro na Internet somente em *sites* de empresas conhecidas e que se comprometem com a privacidade do usuário. Informações sobre os clientes, como o endereço de *e-mail*, podem ser comercializadas.

Os nomes de usuário dos endereços de *e-mail* não podem ser simples como, por exemplo, apenas o primeiro nome. Isso facilita a descoberta do endereço por DHA.

Não tentar descadastrar o endereço de *e-mail* da lista de distribuição de spam. De modo geral, isso serve apenas para confirmar que o endereço é válido e utilizado. Segundo um relatório da FTC (2003), 63% das solicitações de remoção do *e-mail* da lista de distribuição não foram honradas.

Jamais adquirir um produto ou serviço anunciado por spam. Essa atitude incentiva os *spammers* a continuarem no negócio e só contribui para que eles consigam mais informações sobre o comprador, aumentando ainda mais a quantidade de spam. Indiretamente isso incentiva também o desenvolvimento de vírus e *worms* para controlar computadores de terceiros para o envio de spam.

Ao preencher formulários on-line, certificar-se de que as opções que autorizam o envio de propagandas estejam desmarcadas. Além disso, é importante conhecer a política de privacidade da instituição.

Configurar os programas clientes de *e-mail* para não abrir automaticamente o conteúdo dos *e-mails*, desabilitando o painel de exibição. Ao clicar num *e-mail* mesmo que seja para excluí-lo, os clientes de *e-mail*, por padrão, abrem a mensagem automaticamente. Com o painel desabilitado será necessário um clique duplo para abrir um *e-mail*. Dessa forma, é possível manipular as mensagens suspeitas sem abri-las, evitando possível contato desnecessário com código malicioso e *web bugs* presentes em spam.

Desconfiar de *links* contidos em *e-mails* de instituições financeiras. *E-mails* são facilmente forjados e podem conter *links* que direcionam os usuários a *sites* clonados dos oficiais para capturar informações como o número da conta e a senha, resultando em roubo de identidade. Em caso de dúvida, a instituição em questão deve ser consultada.

Não encaminhar correntes e boatos para evitar que o endereço de *e-mail* fique circulando pela Internet. Se o *e-mail* contiver informações interessantes que não pareçam com boato, verificar a veracidade, por exemplo, na Internet com o auxílio de um mecanismo de busca.

Minimizar a publicação de endereços de *e-mail* em páginas da Internet. Segundo um estudo realizado pelo CDT (CENTER FOR DEMOCRACY & TECHNOLOGY, 2003), endereços de *e-mail* divulgados na *web* ou *newsgroups*

recebem mais spam. Nesse estudo, endereços que foram escritos “literalmente” ou que tiveram os caracteres substituídos pelo equivalente em HTML (*Hypertext Markup Language*) não receberam spam. Por exemplo, o endereço “example@domain.com” publicado como “example arroba domain ponto com” ou codificado em HTML como “exampl e@domain.co m”. Endereços de *e-mail* também podem ser divulgados em imagens. Essas técnicas visam dificultar a obtenção dos endereços de *e-mail* por robôs controlados por *spammers*.

O processo de educação de usuários pode ter como base a “Cartilha de Segurança para Internet” do CERT.br (2005), que contém informações sobre aspectos de segurança. A parte VI desse material é dedicada ao tema “spam”.

O livro “Combatendo o Spam” de Renata Cicilini Teixeira (2004) é outra fonte de informações sobre esse assunto.

Capítulo 5

5 Sistema anti-spam

Infelizmente, ainda não existe uma solução tecnológica simples e direta para erradicar o spam. O ser humano consegue identificar um spam, mas tornar o computador capaz de realizar a classificação de *e-mails* sem erros não é uma tarefa trivial. *Spammers* freqüentemente forjam *e-mails* para dificultar a classificação dos *e-mails* feita pelos sistemas anti-spam.

5.1 Técnicas anti-spam

O SpamAssassin implementa diversas técnicas que auxiliam na classificação de *e-mails* em legítimos ou spam.

5.1.1 Verificação em DNSBL

Nessa técnica, listagens de endereços IP de computadores relacionados com o envio de spam são disponibilizadas na Internet. Servidores de *e-mail* consultam essas informações ao receber cada *e-mail*. Se o endereço IP do remetente estiver cadastrado na lista, o *e-mail* é rejeitado. Caso contrário, o *e-mail* é entregue ao destinatário.

Para facilitar a publicação e consulta a essas listagens, as DNSBLs (*DNS-Based Blackhole List*) foram construídas sobre um serviço muito importante para o funcionamento atual da Internet: o DNS (*Domain Name System*).

As listas podem ser de servidores de *e-mail* com o *relay* aberto, de servidores *proxy* abertos, de computadores relacionados com o envio de spam ou uma

combinação desses itens. Por exemplo, as listas da DSBL (*Distributed Sender Blackhole List*) contêm os endereços IP de servidores que encaminham mensagens de teste especiais para “listme@listme.dsbl.org”; isso ocorre se o servidor possui o *relay* aberto, é um *proxy* aberto ou possui qualquer vulnerabilidade que permita que qualquer pessoa envie *e-mail* para qualquer lugar, por meio desse servidor. (DSBL, 200?).

Endereços IP relacionados com o envio de spam são denunciados por administradores de servidores de *e-mail* ou mesmo por usuários. Testes automatizados executados por *software* fornecido pelo mantenedor da lista negra permitem confirmar facilmente a suspeita de *relay* aberto ou *proxy* aberto. Os responsáveis são notificados e até que o problema seja solucionado, o IP continua cadastrado na lista.

A maioria dos MTAs suportam o sistema de verificação em listas negras, podendo rejeitar ou marcar os *e-mails* enviados por IPs listados. A consulta em listas negras também é implementada em sistemas anti-spam. O SpamAssassin, por exemplo, consulta as listas negras da Tabela 4.

Tabela 4 - Exemplos de DNSBL.

Nome da DNSBL	***Site* da DNSBL**
Distributed Sender Blackhole List	http://dsbl.org/
Not Just Another Bogus List	http://www.njabl.org/
Spam and Open Relay Blocking System	http://www.us.sorbs.net
SpamCop	http://www.spamcop.net/

A consulta em listas negras é ineficaz quando os usuários, por exemplo, comunicam-se com empresas ou instituições que possuem servidores de *e-mail* com o *relay* aberto. Isso resultaria numa porcentagem alta de falsos positivos. Por outro lado, se computadores infectados por código malicioso se tornarem servidores de *e-mail* com o *relay* aberto, apenas os *e-mails* enviados pelos usuários desses computadores seriam bloqueados.

A existência de listas negras pode incentivar a atualização ou a correta configuração de servidores de *e-mail*. Empresas podem ter a imagem denegrida por terem os seus *e-mails* bloqueados em sistemas anti-spam que implementam a verificação em listas negras.

5.1.2 Filtragem por palavras-chave

Uma das primeiras técnicas desenvolvidas para bloquear spam foi a filtragem por palavras-chave. Inicialmente, era relativamente fácil bloquear spam, vírus e *worms*, por exemplo, pelo assunto do *e-mail*.

O *worm* "*LoveLetter*" no ano 2000 se espalhou rapidamente pelo mundo todo, trazendo grande destruição. Arquivos dos tipos .vbs, .vbe, .js, .jse, .css, .wsh, .sct, .hta, .jpg, .jpeg, .wav, .txt, .gif, .doc, .htm, .html, .xls, .ini, .bat, .com, .avi, .qt, .mpg, .mpeg, .cpp, .c, .h, .swd, .psd, .wri, .mp3, e .mp2 dos computadores infectados eram sobrescritos pelo código malicioso do *worm* (CHIEN; EWELL, 2001?). O CERT publicou uma notificação sobre essa ameaça, que trouxe também informações de como bloqueá-la pelo assunto do *e-mail* em servidores implementados com o *software* Postfix e Procmail. (CERT, 2000).

Atualmente, spam, vírus e *worms* são forjados de tal maneira que são difíceis de serem bloqueados em filtros por palavras-chave. Os vírus e *worms* se propagam encaminhando *e-mails* do usuário do computador com uma cópia do código malicioso anexada. Alguns tipos de arquivos são freqüentemente associados aos vírus e *worms* e podem ser bloqueados no servidor de *e-mail*. Um *software* antivírus também é bastante eficiente, mas precisa ser atualizado diariamente.

Propagandas de medicamentos como o Viagra são velhas conhecidas dos usuários do correio eletrônico. A filtragem de *e-mails* que contêm a palavra "viagra" seria muito simples se os *spammers* não mascarassem a grafia de diversas formas: v_i_a_g_r_a, v1agra, vi@gr@, \/iagra (note que não foi usada a letra "V"), etc.

As regras podem ser feitas para filtrar *e-mails* com determinado endereço IP de origem, pelo conteúdo do campo assunto ou de palavras encontradas no corpo da mensagem. O SpamAssassin possui regras de filtragem que analisam se o *e-mail* é: composto por texto codificado em HTML, se o texto contém palavras em caixa-alta, se o tamanho da fonte é grande, se o cabeçalho foi forjado, etc.

O uso das regras pré-definidas que acompanham o *software* anti-spam pode ajudar no processo de filtragem de spam. No entanto, fazer a manutenção delas é um processo demorado e complexo. O administrador precisa coletar e analisar o spam que passa pelo filtro (falsos negativos) e depois, atualizar ou criar novas regras. Além disso, as alterações requerem muita atenção, pois o resultado seria catastrófico se fossem mal-elaboradas.

5.1.3 Filtro bayesiano

Thomas Bayes (1702-1761) foi o primeiro a usar avaliações probabilísticas através da indução. Isto é, o cálculo da probabilidade de um novo evento com base em estimativas probabilísticas anteriores que podem ter sido derivadas de dados empíricos.

O método bayesiano tem sido utilizado largamente na área da teoria de decisão estatística. Nesse contexto, o teorema de Bayes fornece um mecanismo para combinar uma distribuição de probabilidades constituída anteriormente com novas informações. Essa combinação pode gerar novos estados que podem ser usados depois como um modelo probabilístico que será combinado com mais dados e assim por diante. O objetivo é que as probabilidades anteriores sejam usadas para fazer melhores decisões. Portanto, trata-se de um processo de aprendizado iterativo, e é uma base para se criar programas de computador que aprendam a partir da experiência.

O algoritmo bayesiano para filtragem de spam foi inicialmente descrito por Paul Graham em "*A Plan for Spam*" (2002). Para funcionar, o filtro precisa ser treinado com spam e *e-mails* legítimos para criar uma base de dados. O conteúdo de *e-mails* é confrontado com os dados obtidos no treinamento do filtro para determinar a probabilidade de um *e-mail* ser spam ou não.

O filtro bayesiano é mais eficaz que o filtro por palavras-chave, porque não se baseia num único elemento como numa regra para determinar a probabilidade de um *e-mail* ser ou não um spam.

MAFRA e FRAGA realizaram um estudo comparativo com diferentes implementações do algoritmo bayesiano, que apresentaram uma variação de acerto entre 82,42% e 94,71%. O resultado está na Tabela 5.

Tabela 5 - Comparação de diferentes implementações do algoritmo bayesiano.

Tipos de filtros	**Falso positivo**	**Falso negativo**	**Acertos**
BMF	2	35	663
Bogofilter	1	89	610
Quick Spam	21	102	577
SpamAssassin	2	110	588

Fonte: (MAFRA; FRAGA, 2004)

Paul Graham em "*Better Bayesian Filtering*" (2003) discute a divergência de resultados obtidos com diferentes implementações. Em seu experimento obteve 99,5% de acerto na detecção de spam com apenas 0,03% falsos positivos, mas menciona o resultado de outros pesquisadores com 92% de acerto na detecção de spam com 1,16% falsos positivos.

Segundo o autor, a redução na eficiência do algoritmo pode ter ocorrido, porque:

- A quantidade de *e-mails* submetidos ao teste foi pequena (466 *e-mails* legítimos e 160 spams);
- Informações contidas no cabeçalho foram ignoradas;
- Apenas as raízes das palavras foram armazenadas no banco de dados;
- Foram utilizados todos os itens (*token*) do *e-mail* ao invés de apenas alguns;
- Algoritmos para filtragem de spam devem possuir mecanismo para diminuir a proporção de falsos positivos em detrimento da taxa filtragem de spam.

A empresa GFi, que fornece solução anti-spam chamada "GFi *Mail Essentials*", afirma que o algoritmo bayesiano é capaz de uma taxa de acerto na detecção de spam maior que 98% (GFI, 2006).

5.1.4 Verificação de assinatura

O spam é coletado, através da colaboração de usuários, para a criação de assinaturas que são armazenadas em banco de dados de terceiros. Os clientes podem então consultar as bases de assinaturas que existem no catálogo para identificar o spam.

Para dificultar a criação de assinaturas confiáveis, os *spammers* alteram informações ou incluem seqüências de caracteres sem sentido em cada spam enviado. Por isso as assinaturas podem ser criadas também se baseando em partes do *e-mail* como, por exemplo, URLs (*Uniform Resource Locator*).

5.1.5 Verificação em *whitelist* / *blacklist*

Existem casos em que a classificação dos *e-mails* de uma determinada origem deve ser expressamente liberada ou barrada. Por exemplo, se um usuário deseja

receber propagandas de uma determinada origem, mas que são bloqueadas pelo sistema anti-spam, é preciso cadastrar o endereço de origem na *whitelist*. O inverso também pode ocorrer, caso os *e-mails* de uma origem não estejam sendo bloqueados pelo sistema anti-spam, o seu *e-mail* pode ser cadastrado na *blacklist*.

Cadastrar os *e-mails* de *spammers* numa *blacklist* não é uma boa estratégia, pois o *e-mail* preenchido no campo de remetente pode ser facilmente forjado.

5.1.6 Sistema de *Auto-whitelist*

O sistema de *auto-whitelist* acompanha a pontuação média atribuída aos *e-mails* de um determinado remetente e aproxima a nota do próximo *e-mail* a essa média. Essa técnica permite que mensagens de usuários que às vezes podem ter características de spam passem pelo filtro anti-spam.

O SpamAssassin executa inúmeras técnicas para detectar características de spam nos *e-mails*. Cada característica de spam atribui uma pontuação. Se a soma da pontuação ultrapassar a nota de corte, o *e-mail* é considerado spam.

Digamos que *e-mails* enviados por uma pessoa têm a média da pontuação igual a 0, que para o SpamAssassin é uma pontuação baixa e caracteriza *e-mails* legítimos. O próximo *e-mail* recebe a pontuação 10; que é superior à nota de corte, caracterizando spam. O sistema de *auto-whitelist* atribuirá uma pontuação para aproximar a pontuação final desse *e-mail* para a média. Isso permitirá que o *e-mail* passe pelo filtro.

Como o sistema funciona:

- A pontuação do *e-mail* sem o *auto-whitelist* é calculada;
- A pontuação do passo anterior é subtraída da média. O resultado dessa operação é multiplicado pelo fator de correção, que no SpamAssassin o valor padrão é 0,5;
- A pontuação final do *e-mail* é calculada somando-se a pontuação do primeiro passo com a do segundo.

No exemplo anterior, a pontuação do *e-mail* sem o *auto-whitelist* é igual a 10. A pontuação 10 é subtraída de 0 e depois multiplicada por 0,5, que é igual a -5. A pontuação final do *e-mail* é igual a pontuação sem o *auto-whitelist* somada ao valor do segundo passo, ou seja, é 10+(-5)=5.

O sistema armazena a média dos *e-mails* de um indivíduo, atrelado ao endereço de *e-mail* e o endereço IP. Na verdade, são armazenados apenas os dois primeiros octetos, porque o usuário pode estar utilizando endereços de IP dinâmicos fornecidos pelo provedor. Dessa forma, é improvável que um spam com endereço de remetente forjado se beneficie da pontuação média de um usuário legítimo para passar pelo filtro.

5.2 Dificuldades no bloqueio do spam

Sabendo das campanhas anti-spam, os *spammers* usam os seguintes recursos para dificultar a detecção e o bloqueio do spam:

- Troca freqüente do IP de origem de spam;
- Troca freqüente de campos do cabeçalho como o *e-mail* do remetente e o nome;
- Alterações freqüentes no modo de exibição da propaganda: mudança na grafia das palavras / utilização de imagens / carregamento automático de páginas da *web*;
- Ataques de DoS nos servidores de lista negra e de verificação de assinatura.

Capítulo 6

6 Implantação do sistema anti-spam

O correio eletrônico está cada vez mais presente no dia-a-dia como ferramenta para comunicação corporativa ou pessoal. Os problemas causados pelo spam estão forçando empresas, instituições e provedores a buscarem por soluções para minimizar o impacto.

6.1 Implantação

O sistema anti-spam e antivírus, desse estudo, foi implantado no DT/SIBi/USP (Departamento Técnico do Sistema Integrado de Bibliotecas da Universidade de São Paulo). Nesse local, já havia um servidor de *e-mail*, implantado com *software* proprietário, sem a utilização de qualquer ferramenta para filtragem de *e-mails*. Todos os funcionários do DT/SIBi, incluindo alguns estagiários, possuem conta no servidor de *e-mail*, totalizando 91 contas e 65 listas de distribuição de *e-mail*, que possuem participantes de outras unidades da USP e/ou externos.

Nos últimos anos, houve um aumento significativo na quantidade de solicitações de suporte técnico para solucionar dúvidas relacionadas ao *e-mail*, tanto sobre como agir quando um código malicioso é detectado pelo antivírus instalado em cada estação de trabalho, como também de *e-mails* “estranhos” ou “suspeitos”. Com a necessidade de atualização do hardware e *software* do servidor e com o aumento de *e-mails* indesejados surgiu a oportunidade de realizar esse estudo prático.

Com o servidor antigo em regime de produção, foram feitas a instalação e a configuração dos *softwares* necessários para o funcionamento do novo servidor. Para conhecer e testar o sistema implantado na prática, sem afetar todos os usuários, os *e-mails* de funcionários do setor de informática passaram a ser redirecionados para o

novo servidor. Durante os 4 meses de teste, não ocorreram problemas. Então, todas as contas de *e-mail* foram migradas para o novo servidor.

As novas características e as mudanças feitas no serviço de *e-mail* foram apresentadas aos usuários. Aproveitou-se também para conscientizá-los sobre o spam e para tirar as dúvidas mais freqüentes sobre esse assunto.

6.2 Funcionamento do sistema anti-spam implantado

O sistema anti-spam implantado analisa os *e-mails* através das técnicas de verificação em DNSBL, filtragem por palavras-chave pré-definidas no *software* anti-spam, filtro bayesiano, verificação de assinatura e sistema de *auto-whitelist*.

Cada técnica analisa o *e-mail* recebido e atribui uma pontuação, quando pertinente. A verificação em DNSBL, por exemplo, atribui pontuação apenas quando o endereço IP de origem do *e-mail* está cadastrado no banco de dados. Caso a soma de todas as pontuações seja maior do que um limiar ou "nota de corte", definido na configuração do *software*, o *e-mail* é considerado spam. Por outro lado, se for menor é tratado como *e-mail* legítimo. O limiar padrão do SpamAssassin é cinco.

Casos de falso-positivos podem causar transtornos aos usuários do sistema de correio eletrônico e conseqüentemente aumentar a insegurança quanto a sua confiabilidade. Por exemplo, se um usuário não receber a resposta de um *e-mail* num prazo que ele considera razoável, ele poderá concluir que o sistema anti-spam a bloqueou. Conseqüentemente, o administrador será freqüentemente questionado sobre *e-mails* não recebidos, o que acabará diminuindo a sua produtividade.

Então, para diminuir a probabilidade de ocorrer algum caso de falso positivo no sistema implantado o limiar do SpamAssassin foi aumentado de cinco para sete. Além disso, os *e-mails* com até um ponto a mais que o limiar são marcados e entregues aos usuários para permitir que eles verifiquem se os *e-mails* com a

pontuação próxima ao limiar não se tratam de falsos positivos. Os demais são detidos numa área de quarentena para diminuir o volume de *e-mail* entregue nas caixas postais dos usuários finais.

Assim apenas os *e-mails* com pontuação acima de oito são colocados em quarentena. O resto dos *e-mails* é entregue aos usuários. Mas, os que tiverem pontuação entre 7 e 8 são marcados com "[SPAM]" no início do assunto da mensagem.

Como no período de 10 meses não ocorreram casos confirmados de falsos positivos ocasionados pelo SpamAssassin, a nota de corte foi baixada para 6. Então, os *e-mails* com pontuação acima de sete são colocados em quarentena.

A área de quarentena é importante, porque caso seja necessário é possível recuperar *e-mails* bloqueados indevidamente.

6.3 Hardware e software utilizado no servidor

A configuração de hardware do servidor de *e-mail* é a seguinte:

- Placa mãe Gigabyte 8IR2003
- Processador Pentium 4 - 2.4GHz - cache L2 de 512 KB
- 2 módulos de memória RAM DDR de 256MB - 266MHz
- 2 HDs IDE de 40GB de 7200 RPM
- Unidade de fita DAT Seagate STD2401LW

O sistema operacional Linux, Red Hat 9, foi instalado com uma configuração personalizada para servidor. Foram incluídos os pacotes referentes ao compilador, kernel e outros pré-requisitos do sistema como o PERL (*Practical Extraction and Report Language*) e módulos da CPAN (*Comprehensive Perl Archive Network*). Os *softwares* e o sistema operacional foram atualizados por um serviço oferecido pelo fabricante chamado RHN (*Red Hat Network*).

Os discos rígidos foram configurados, durante a instalação do sistema operacional, para funcionarem como espelho (*software*-RAID 1). Isso facilita o restabelecimento do servidor no caso de falha de um dos discos rígidos, porque a reinstalação do sistema operacional e dos *softwares* necessários, a atualização de tudo e o restauro do backup podem levar alguns dias. No caso do ambiente de instalação, há peças sobressalentes para os outros componentes de hardware do servidor.

O primeiro pacote a ser instalado a partir do código fonte é o Netqmail (CAZABON et al., 2005) que contém o código-fonte do MTA Qmail (BERSTEIN, 2001?) e do Qmail-pop3d. O Qmail é um MTA seguro, confiável e simples, criado para substituir o Sendmail (SILL, 2006). O Qmail-pop3d é um *daemon* POP 3 que acompanha o Qmail e suporta o formato para armazenamento de mensagens "maildir".

Foram instalados também os seguintes pacotes oferecidos pelo criador do Qmail: Ucspi-tcp, Daemontools e Checkpassword. O pacote ucspi-tcp inclui um substituto para o inetd, que gerencia conexões para os serviços oferecidos, e uma ferramenta genérica para rejeitar *e-mails* oriundos de endereços IP cadastrados em listas negras. O pacote daemontools permite controlar o carregamento (*startup*), desligamento (*shutdown*) e recarregamento automático (*automated restarting*) de processos. O checkpassword permite a autenticação dos usuários do POP3 (SAMUEL, 2003).

A instalação e configuração do Netqmail 1.05, Checkpassword 0.90, Ucspi-tcp 0.88 e Daemontools 0.76 foram feitas segundo o guia "*Life with qmail*" de Dave Sill (SILL, 2006). O Qmail foi instalado no diretório /var/qmail conforme o guia e foi configurado para armazenar os *e-mails* no formato Maildir, por ser mais confiável que o Mailbox.

Convenções:

- Todos os pacotes que foram obtidos da Internet foram armazenados no diretório “/usr/local/src” para serem compilados e instalados.
- Nos quadros com instruções de instalação, as linhas que começam com “#!” são usadas para indicar o interpretador de comandos necessário para executar o “script”. As demais linhas que começam com “#” são comentários.
- Os pacotes foram compilados e instalados com o usuário “root”.

Depois de testado o envio e recebimento de *e-mails*, o antivírus ClamAV (KOJM et al., 2005) foi instalado. É um *software* livre distribuído conforme a licença GPL. O propósito principal do *software* é a integração com servidores de *e-mail* (varredura em arquivos anexos) (ClamAV, 200?). O pacote pode ser obtido no *site*: http://www.clamav.net. Foi instalado seguindo os passos abaixo:

```
tar -zxvf clamav-0.70.tar.gz
cd clamav-0.70

groupadd clamav
useradd –g clamav -c "ClamAV" -s /bin/false clamav

./configure --sysconfdir=/etc
make
make check
make install
make clean
```

Para configurar o programa, editar o arquivo de configuração /etc/clamav.conf (Anexo B1):

- Comentar a linha que começa com “Example” (linha 8)
- Descomentar a linha que começa com “LogFile” e alterar o valor para “/var/log/clamd/clamd.log” (linha 14)
- Descomentar a linha que começa com “LogTime” (linha 32)
- Descomentar a linha que começa com “PidFile” (linha 46)
- Descomentar a linha que começa com “MaxThreads” (linha 91)
- Descomentar a linha que começa com “User” e alterar o valor para “qscand” (linha 120)
- Descomentar a linha que começa com “ScanMail” (linha 144)

Criar e configurar as permissões do diretório que armazenará o arquivo que contém o PID (*Process ID*):

```
mkdir /var/run/clamav
chown qscand.qscand /var/run/clamav
```

Para verificar se está funcionando:

```
clamscan -r -l scan.txt .
cat scan.txt
```

Configurar o dono e o grupo dos executáveis:

```
chown qscand.qscand /usr/local/bin/clam*
```

Criar o arquivo de *log* e configurar o dono, o grupo e as permissões:

```
mkdir /var/log/clamd
chown clamav.clamav /var/log/clamd

touch /var/log/clamd/clam-update.log
chmod 644 /var/log/clamd/clam-update.log
chown clamav.clamav /var/log/clamd/clam-update.log
```

Para abrir a interface para configurar o agendamento da atualização automática do antivírus, executar o comando:

```
crontab –e
```

Inserir, no final do arquivo, as linhas abaixo:

Atualização das assinaturas de vírus ClamAV

Todos os dias às 8h e às 20h.

0 8,20 * * * /usr/local/bin/freshclam --quiet -l /var/log/clamd/clam-update.log

Criar o script, para carregamento do *daemon*, /etc/rc.d/init.d/clamd com o seguinte conteúdo:

```
#!/bin/bash
# chkconfig: 345 98 40
# description: Startup and shutdown script for clamd
# processname: clamd
# pidfile: /var/run/clamav/clamd.pid

case "$1" in
start)
  echo -n "Iniciando o Clamav aguarde..."
  /usr/local/sbin/clamd
  echo "Clamav rodando."
  ;;
stop)
  echo -n "Parando o Clamav"
  kill -TERM `cat /var/run/clamav/clamd.pid`
  echo "OK"
  ;;
*)
  echo "Usage $0 {start | stop}"
  exit 1
  ;;
esac
exit 0
```

Configurar a execução do *daemon* na inicialização:

```
chkconfig –add clamd
```

O Qmail-scanner possui várias funcionalidades, dentre elas são destacadas:

- A compatibilidade com praticamente qualquer *software* antivírus para Unix executado por linha de comando;
- A possibilidade de bloquear *e-mails* por tipo de arquivo anexo;
- A integração com o SpamAssassin. (HAAR, 2005).

O SpamAssassin (HUGHES et al., 2005) e os módulos necessários foram obtidos instalados da CPAN. Das técnicas utilizadas no sistema anti-spam, a única que foi instalada separadamente foi a verificação de assinatura, implementada pelo Razor (PRAKASH, 2005).

Antes de instalar o SpamAssassin é preciso definir a variável “LANG”, senão ocorrerá um erro.

```
LANG=C
export LANG
```

Para instalar ou atualizar o SpamAssassin, executar os comandos:

```
perl -MCPAN -e shell
o conf prerequisites_policy ask
install Mail::SpamAssassin
quit
```

O arquivo de configuração do SpamAssassin “local.cf” pode ser encontrado no diretório /etc/mail/spamassassin (Anexo B2). Além de linhas com comentários deve possuir o seguinte conteúdo:

```
required_hits 7 (linha 7)
rewrite_subject 1 (linha 8)
subject_tag [SPAM] (linha 9)
```

A primeira linha define a nota de corte ou limiar da classificação para determinar se o *e-mail* é classificado como legítimo ou spam. A segunda, que o assunto do *e-mail* deve ser alterado. E a terceira, o que deve ser adicionado ao assunto. O SpamAssassin por padrão habilita o uso das técnicas.

Nesse ponto já é possível treinar o filtro bayesiano. Basta obter um arquivo com spams e outro com *e-mails* legítimos conhecidos também como *ham*. O

comando abaixo pressupõe o uso de um arquivo no formato mbox (opção “--mbox”) chamado spam para o treinamento da base de spam (opção “--spam”). Serão exibidos caracteres “.” (opção “--showdots”) para indicar que o processamento do arquivo está sendo executado.

```
sa-learn --showdots --mbox --spam spam
```

Para treinar o filtro com *e-mails* legítimos, o mesmo comando apresentado acima deve ser usado, mas com a opção “--ham” no lugar da opção “--spam” e um arquivo com *e-mails* legítimos chamado, por exemplo, *ham*.

O SpamAssassin por padrão habilita o aprendizado automático. A alimentação da base é feita através de *e-mails* classificados pelo SpamAssassin seguindo determinados critérios. Para um *e-mail* ser alimentado como legítimo é preciso que a pontuação seja abaixo ou igual a 0,1 e para ser alimentado como spam a pontuação deve ser superior ou igual a 12,0, sendo que 3 pontos devem ser obtidos por análise de cabeçalho e 3 de corpo, ou seja, a pontuação mínima para alimentar um *e-mail* como spam é 6. Por padrão, para que o filtro bayesiano seja ativado, é preciso que pelo menos 200 spams e 200 *hams* sejam “aprendidos”. (SPAMASSASSIN, 2004?)

O agente razor foi instalado da seguinte forma:

```
tar -zxvf razor-agents-sdk-2.03.tar.gz
cd razor-agents-sdk-2.03
perl Makefile.PL
make
make test
make install

tar –zxvf razor-agents-2.40.tar.gz
cd razor-agents-2.40
perl Makefile.PL
make
make test
make install
```

O Qmail-scanner possui como pré-requisitos: Qmail 1.03, Perl 5.00_03+ os módulos PERL Time::HiRes, DB_File e Sys::Syslog, o Reformime 1.3.8+ do pacote Maildrop, o TNEF e o *patch* para o Qmailqueue.

O módulo Perl Time::Res (HIETANIEMI et al., 2005) implementa uma interface em Perl para chamadas de sistema usleep, nanosleep, ualarm, gettimeofday

e settimer/getitimer de tempo de alta resolução e temporizadores. O módulo DB_FILE (MARQUESS, 2005) permite que programas em Perl utilizem a biblioteca Berkley DB versão 1, implementada em C, que fornece uma interface para banco de dados dos tipos: hash, btree e recno. O módulo Sys::Syslog (CHRISTIANSEN; WALL, 2002) é uma interface para o programa UNIX syslog.

O Reformime possui a habilidade para desmontar e montar uma mensagem MIME em arquivos separados para cada parte (QMAIL INFO, 2005). O TNEF (SIMPSON, 2005) é um programa para extrair anexos MIME do tipo "application/ms-tnef" de MTAs da Microsoft. O *patch* para o Qmailqueue (GUENTER, 1999) permite a remoção e devolução de *e-mails* da fila do Qmail para passar pelos filtros.

Os pré-requisitos que ainda não estavam presentes no servidor eram: o módulo Sys::Syslog, Reformime, TNEF e o *patch* para o Qmailqueue.

Para obter uma listagem dos módulos Perl instalados no sistema, foram executados os seguintes comandos:

```
cd /usr/lib/perl5
find ./| grep \.pm$
```

Esse comando retornará uma listagem dos arquivos cuja extensão é ".pm". Cada arquivo é um módulo. Por exemplo, para saber se o módulo Time::HiRes está instalado nessa listagem deve aparecer um arquivo de nome "HiRes.pm" dentro de um diretório chamado "Time".

O pacote Sys::Syslog foi instalado da CPAN com os seguintes comandos:

```
perl -MCPAN -e shell
o conf init
cpan> install Sys::Syslog
```

O pacote maildrop versão 1.6.3, obtido em www.courier-mta.org/maildrop/, foi instalado da seguinte forma:

```
cd /usr/local/src
bunzip2 maildrop-1.6.3.tar.bz2
tar xf maildrop-1.6.3.tar
cd maildrop-1.6.3
./configure
make
make install-strip
make install-man
```

O TNEF versão 1.2.3.1, obtido em http://sourceforge.net/projects/tnef/, foi instalado da seguinte forma:

```
cd /usr/local/src
tar xzf tnef-1.2.3.1.tar.gz
cd tnef-1.2.3.1
./configure
make
make install
```

O *patch* para o Qmailqueue, obtido em: http://www.qmail.org/qmailqueue-patch, foi instalado da seguinte forma:

```
cd /usr/local/src
cd netqmail-1.05/qmail-1.03
patch -p1 < ../../qmailqueue-patch
```

Foi criada também a conta e grupo qscand:

```
groupadd qscand
useradd -g qscand -s /bin/false qscand
```

Para o funcionamento do programa, é preciso o pacote perl-suidperl (HAAR, 2004). Na distribuição Red Hat, esse pacote não é instalado por padrão, mas está presente no segundo CD.

```
mnt /dev/cdrom /mnt/cdrom
cd /mnt/cdrom/RedHat/RPMS
rpm –ivh perl-suidperl-5.8.0-88.i386.rpm
```

O pacote qmail-scanner versão 1.21, obtido em qmail-scanner.sourceforge.net/, foi instalado da seguinte forma:

```
cd /usr/local/src/
tar –zxvf qmail-scanner-1.21.tgz
cd qmail-scanner-1.21/
# Configurar no comando abaixo: scanners="fast_spamassassin"
./configure –install
```

O comando abaixo executa o “qmail-scanner-queue.pl” com o UID e GID do usuário “qmaild” para gerar o arquivo de banco de dados para bloqueio de arquivos

por extensão, tamanho e assunto a partir do arquivo /var/spool/qmailscan/quarantine-attachments.txt.

```
setuidgid qmaild /var/qmail/bin/qmail-scanner-queue.pl –g
```

Para configurar os tipos de arquivos que devem ser bloqueados, editar o arquivo /var/spool/qmailscan/quarantine-attachments.txt (Anexo B3) e descomentar as linhas:

- .vbs 0 VBS files not allowed per Company security policy (linha 98)
- .scr 0 SCR files not allowed per Company security policy (linha 100)
- .wsh 0 WSH files not allowed per Company security policy (linha 101)
- .hta 0 HTA files not allowed per Company security policy (linha 102)
- .pif 0 PIF files not allowed per Company security policy (linha 103)
- .com 0 PIF files not allowed per Company security policy (linha 104)

E reconstruir o banco de dados do Qmail-scanner para tornar efetiva a configuração anterior:

```
setuidgid qmaild /var/qmail/bin/qmail-scanner-queue.pl -g
```

Para apagar automaticamente arquivos temporários com mais de 30 horas, abrir a interface para configurar o agendamento de tarefas:

```
crontab –e
```

Inserir, no final do arquivo, as linhas abaixo:

Remoção de arquivos temporários do Qmail-scanner com mais de 30 horas

0 0 * * * /var/qmail/bin/qmail-scanner-queue.pl -z

Aumentar a memória do qmail-smtpd de 2MB para 15MB, afinal o processo estará executando, por exemplo, o PERL e o ClamAV (PEACE, 2004). Abrir o arquivo /var/qmail/supervise/qmail-smtpd/run e na linha abaixo, alterar o *softlimit* de 2000000 para 15000000:

```
exec /usr/local/bin/softlimit -m 2000000 \
```

Redirecionar a fila dos *e-mails* para o qmail-scanner-queue.pl ao adicionar a variável QMAILQUEUE no arquivo do Qmail /etc/tcp.smtp. Cada linha desse arquivo é uma configuração para faixas de IP. Digamos que a faixa de IPs da rede local seja 192.168.1.0/24. Essa faixa de IPs e a de *loopback* têm o encaminhamento (*relay*) de mensagens permitido, então devem ser configuradas da seguinte forma:

127.:allow,RELAYCLIENT=""

192.168.1.:allow,RELAYCLIENT=""

Para que o Qmail-scanner analise as mensagens recebidas de outras redes, esse arquivo deve possuir uma linha que começa com ":allow" e com a variável "QMAILQUEUE" definida. Para habilitar o SpamAssassin, a variável QS_SPAMASSASSIN também deve ser definida. Além disso, a verificação de BMC (*Bad MIME Characters)* foi desabilitada ao definir a variável "BMC_WHITELIST", porque notificações de leitura do provedor "Universo *Online*" (UOL) estavam sendo bloqueadas indevidamente.

:allow,QMAILQUEUE="/var/qmail/bin/qmail-scanner-queue.pl",
QS_SPAMASSASSIN="on", BMC_WHITELIST=""

Depois de alterar o arquivo, é preciso gerar uma versão no formato cdb, que segundo o autor, Bernstein, D., é capaz de fazer buscas rápidas:

```
qmailctl cdb
```

A versão oficial do Qmail-scanner não atende os requisitos necessários para esse estudo, como colocar spam em quarentena e permitir que *e-mails* classificados como spam com pontuação próxima ao limiar sejam marcados e repassados aos usuários. Então, uma versão modificada chamada Qmail-scanner-st (TORIBIO, 2005), que pode ser aplicada através de um *patch* no Qmail-scanner ou obtida como um pacote completo, foi instalada. A versão do Qmail-Scanner-st utilizada no servidor foi a 1.22.st.

```
cd /usr/local/src
tar –zxvf q-s-1.22st-20040502.tgz
cd qmail-scanner-1.22st
./configure -–install
```

Abrir o arquivo de configuração “/var/qmail/bin/qmail-scanner-queue.pl” (Anexo B4) e alterar as variáveis abaixo para os seguintes valores:

- my $descriptive_hdrs=1; (linha 99)
- my $V_FROM='root@alfa.linux.sibi.usp.br'; (linha 108)
- my $QUARANTINE_CC='root@sibi.usp.br'; (linha 113)
- my @local_domains_array=('alfa.sibi.usp.br'); (linha 118)
- my $log_details="syslog"; (linha 188)
- my $sa_quarantine='1'; (linha 212)
- my $sa_alt='1'; (linha 231)
- my $sa_debug='1'; (linha 237)
- my $spamc_subject=" [SPAM] " (linha 292)

A variável “descriptive_hdrs” adicionará informação no *e-mail* em linhas de cabeçalho que começam com “X-Qmail-Scanner”. A variável “V_FROM” contém o endereço do remetente usado quando relatórios são gerados. A variável “QUARANTINE_CC” determina o endereço de quem receberá cópias de relatórios de código malicioso. A variável “@local_domains_array” serve para indicar os domínios considerados locais. A variável “log_details” enviará informações de *log* para o “syslog”. As mensagens que tiverem pontuação superior ao valor de “required_hits”, do SpamAssassin, somadas ao valor de “sa_quarantine” devem ser redirecionadas à quarentena. Se o valor de “sa_alt” for “1”, então o relatório do SpamAssassin será adicionado ao *log*. Se o valor de “sa_debug” for “1” então os testes e pontuações do SpamAssassin serão registrados no arquivo “qmail-queue.log”. A variável “spamc_subject” define o rótulo adicionado no início do assunto das mensagens classificadas como spam e que são redirecionadas aos usuários. A variável “spamc_subject” indica o rótulo que deve ser adicionado às mensagens classificadas como spam que são encaminhadas aos usuários.

Os programas foram configurados para funcionarem de modo integrado. A Figura 8 ilustra o funcionamento do sistema proposto. Quando um *e-mail* é recebido, ele é redirecionado para uma fila controlada pelo Qmail-Scanner-st. Se o *e-mail* tiver um anexo, será analisado pelo ClamAV, senão será apenas submetido aos testes do SpamAssassin.

As setas com linhas pontilhadas indicam os possíveis métodos de acesso às mensagens por clientes de *e-mail*.

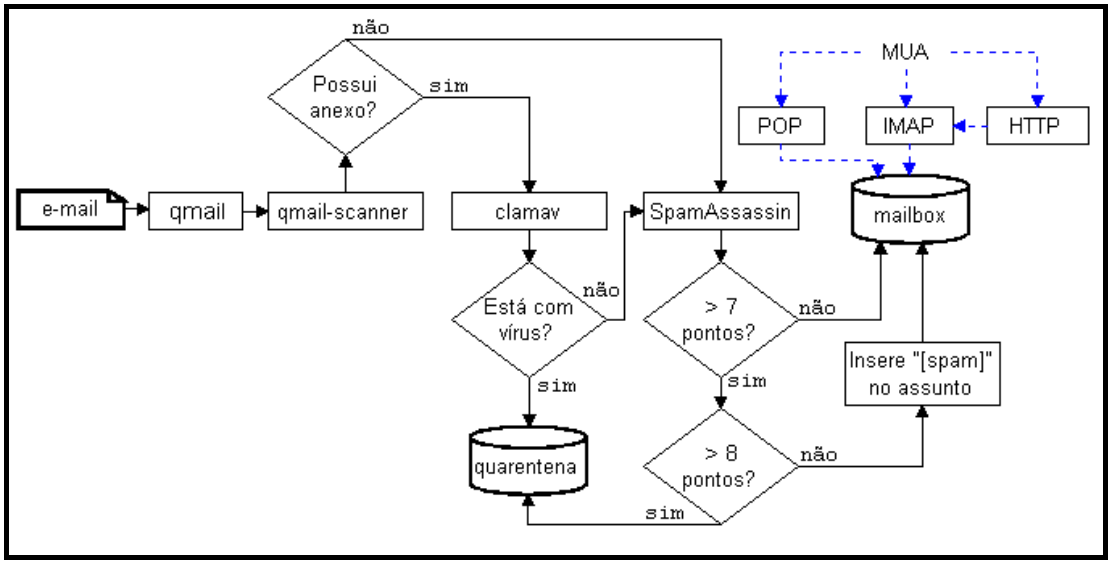

Figura 8 - Funcionamento do sistema proposto.

6.4 Exemplo de análise de um spam

O spam exibido na Figura 9 foi analisado pelo sistema proposto.

Figura 9 - Exemplo de spam.

No arquivo de *log* do Qmail-Scanner é possível verificar os testes aplicados para realizar a classificação dessa mensagem (Figura 10). As linhas foram numeradas para facilitar a identificação. Todas as linhas iniciam com a data e hora, com exceção das linhas com o detalhamento das técnicas e pontuações aplicadas.

A linha 01 é o registro do início da análise do *e-mail*. Na linha 02 há uma indicação para não procurar por *Bad MIME Characters*. A linha 03 informa o tempo decorrido de 0,607021 segundo desde o início da análise. A linha 04 contém as informações de endereço de retorno da mensagem e do destinatário. A linha 05 contém o endereço do remetente e o assunto da mensagem. Na linha 06, há a informação de que o ClamAv levou 0,007487 segundo para verificar se o *e-mail* contém código malicioso. Na linha 07, há a informação de que a pontuação da mensagem do SpamAssassin foi de 14,1 de um limiar de 6.

As pontuações e testes estão entre as linhas 08 e 18. O teste da linha 08 identificou que se trata de uma mensagem de farmácia on-line e atribuiu 3,8 pontos. O teste da linha 09 atribuiu 4,1 pontos por fazer propaganda de Viagra. O teste da linha 10 atribuiu 0,1 ponto por possuir um *link* com o dizer “clique aqui” (*click here*).

O teste da linha 11 atribuiu 0,1 ponto por possuir código em HTML definindo fonte de cor azul. O teste da linha 12 atribuiu 0,1 ponto por possuir HTML na mensagem. O teste da linha 13 atribuiu 0,3 ponto por possuir código em HTML definindo fonte de tamanho grande. O teste da linha 14 atribuiu 0,1 ponto por possuir código em HTML com definição de cor que não é *browser-safe*. O teste da linha 15 do algoritmo bayesiano atribuiu 5,4 pontos por ter a probabilidade de ser spam de 99 a 100%. O teste da linha 16 atribuiu 0,1 ponto por possuir código em HTML definindo fonte de cor vermelha. O teste da linha 17 atribuiu 0,1 ponto por pedir para clicar abaixo (*click bellow*). O teste da linha 18 atribuiu 0,0[1] ponto por possuir de 25 a 50% do texto em caixa alta.

A linha 19 está em branco. A mensagem foi colocada em quarentena, conforme explicado na linha 20. A linha 21 exibe o valor do limiar ("required hits") configurado no SpamAssassin e de "sa_quarantine" e "sa_delete" do Qmail-scanner. Se a pontuação total de uma mensagem for superior a de "required hits" somada a "sa_quarantine", então deve ser colocada em quarentena. Caso contrário, é encaminhada ao destinatário. A variável "sa_delete" está definida como "+0", porque nenhum *e-mail* deve ser apagado. A linha 22 contém o tempo que o SpamAssassin levou para analisar o *e-mail* (10,84369 segundos) e a pontuação (14,1). A linha 23 indica a finalização da análise da mensagem. A linha 24 informa o tempo decorrido de 10,84369 segundos para finalizar a análise do antivírus e do SpamAssassin. A linha 25 indica que o remetente não receberá notificação de que a mensagem foi colocada em quarentena por se tratar de spam. Finalmente, a linha 26 indica que a análise foi terminada e levou no total 10,859795 segundos.

[1] A princípio um teste com pontuação que atribui pontuação "0" não tem serventia na classificação de *e-mails*, mas, se for necessário, pode ter a pontuação alterada.

```
01- Thu, 26 Jan 2006 08:25:41 EDT:3670: +++ starting debugging for process 3670 by
    uid=501
02- Thu, 26 Jan 2006 08:25:41 EDT:3670: WL: The server is in the BMC_WHITELIST, don't
    check BMC
03- Thu, 26 Jan 2006 08:25:42 EDT:3670: w_c: elapsed time from start 0.607021 secs
04- Thu, 26 Jan 2006 08:25:42 EDT:3670: return-path='franky@100000books.com',
    recips='ian@sibi.usp.br'
05- Thu, 26 Jan 2006 08:25:42 EDT:3670: from='"Kayla Abrams"
    <franky@100000books.com>', subj='Visit our new online pharmacy store and save
    upto 85%', via SMTP from 80.55.94.198
06- Thu, 26 Jan 2006 08:25:42 EDT:3670: clamdscan: finished scan of dir
    "/var/spool/qmailscan/tmp/alfa.sibi.usp.br11382711416013670" in 0.007487 secs
07- Thu, 26 Jan 2006 08:25:52 EDT:3670: SA: REPORT hits = 14.1/6.0
08- 3.8 ONLINE_PHARMACY        BODY: Online Pharmacy
09- 4.1 VIAGRA                 BODY: Plugs Viagra
10- 0.1 HTML_LINK_CLICK_HERE   BODY: HTML link text says "click here"
11- 0.1 HTML_FONTCOLOR_BLUE    BODY: HTML font color is blue
12- 0.1 HTML_MESSAGE           BODY: HTML included in message
13- 0.3 HTML_FONT_BIG          BODY: HTML has a big font
14- 0.1 HTML_FONTCOLOR_UNSAFE  BODY: HTML font color not in safe 6x6x6 palette
15- 5.4 BAYES_99               BODY: Bayesian spam probability is 99 to 100%
16- 0.1 HTML_FONTCOLOR_RED     BODY: HTML font color is red
17- 0.1 CLICK_BELOW            Asks you to click below
18- 0.0 UPPERCASE_25_50        message body is 25-50% uppercase
19-
20- Thu, 26 Jan 2006 08:25:52 EDT:3670: SA: yup, this smells like SPAM - quarantine
    message...
21- Thu, 26 Jan 2006 08:25:52 EDT:3670: SA: required_hits 6.0 / sa_quarantine +1 /
    sa_delete +0
22- Thu, 26 Jan 2006 08:25:52 EDT:3670: SA: finished scan in 10.217678 secs -
    hits=14.1
23- Thu, 26 Jan 2006 08:25:52 EDT:3670: ini_sc: finished scan of
    "/var/spool/qmailscan/tmp/alfa.sibi.usp.br11382711416013670"...
24- Thu, 26 Jan 2006 08:25:52 EDT:3670: ini_sc: elapsed time from start 10.84369 secs
25- Thu, 26 Jan 2006 08:25:52 EDT:3670: v_v_t_r: Description contain "spam" - so
    don't notify the sender
26- Thu, 26 Jan 2006 08:25:52 EDT:3670: ------ Process 3670 finished. Total of
    10.859795 secs
```

Figura 10 - Trecho do arquivo de *log* do Qmail-Scanner.

Cada *e-mail* analisado é registrado no arquivo de *log* do Qmail-Scanner. O que varia na classificação de cada *e-mail* são as características de spam detectadas pelas técnicas, que são listadas junto com a pontuação. No exemplo apresentado na Figura 10 corresponde às linhas de 08 a 18. Quanto menos linhas com características de spam forem apresentadas e quanto menor a pontuação atribuída a elas, maior a

probabilidade de o *e-mail* ficar com a pontuação baixa e ser classificado como legítimo.

6.5 Resultados

No dia 10.08.2004 houve a migração do serviço de *e-mail* para o novo servidor com o sistema anti-spam e antivírus.

No período de 01.09.2004 a 31.08.2005 passaram pelo filtro 492.599 *e-mails* que foram tratados pelo sistema da seguinte forma:

52,25% foram encaminhados para os destinatários (257.363)

47,75% foram colocados em quarentena (235.236)

Dos *e-mails* em quarentena:

82,82% foram classificados como spam (194.812)

17,18% continham código malicioso (40.424)

A Figura 11 ilustra um panorama do total de *e-mails* colocados em quarentena diariamente, no período de 01.09.2004 a 31.08.2005, separados em spam e vírus. Os grandes picos em azul escuro representam epidemias dos *worms* Netsky.M (30.09.2004) e Zafi.B (09.11.2004). Os picos em magenta, o aumento significativo de e-mail classificado como spam após a alteração do limiar do SpamAssassin para 6.

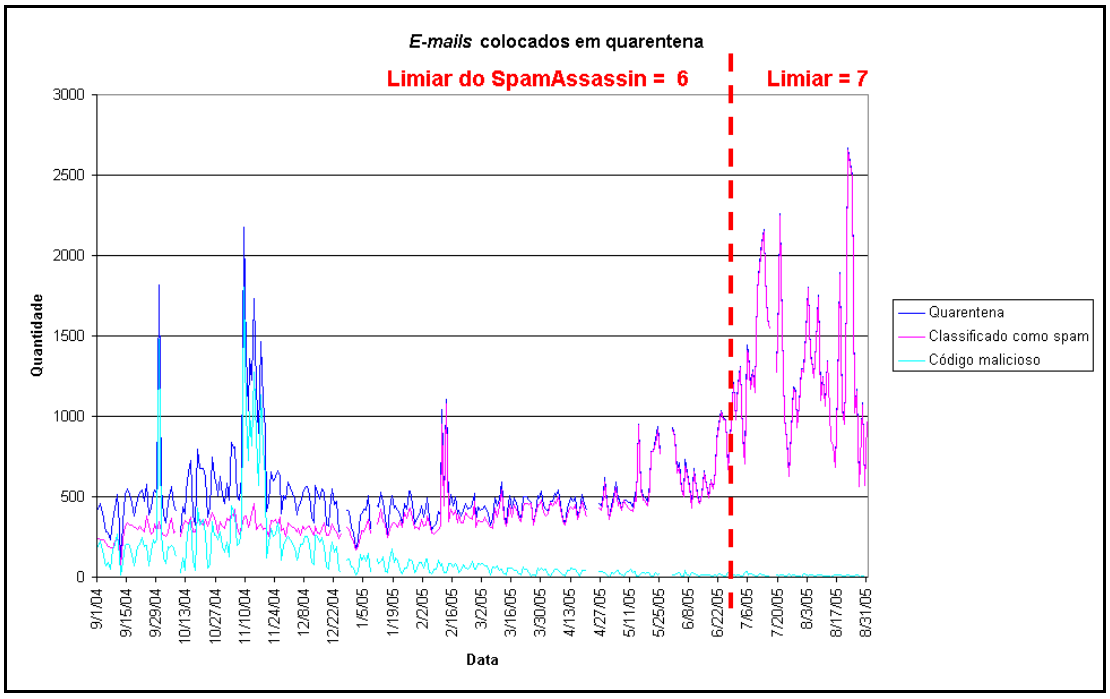

Figura 11 - Quantidade diária de *e-mails* colocados em quarentena.

A Figura 12 apresenta um acompanhamento mensal da porcentagem de mensagens colocadas em quarentena.

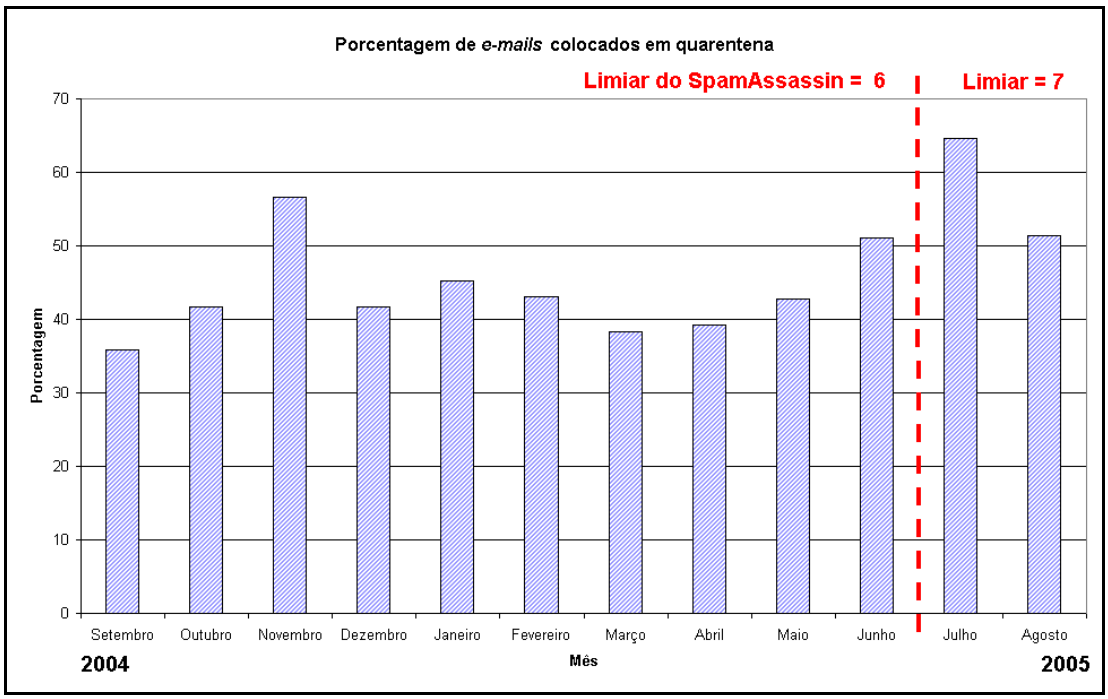

Figura 12 - Porcentagem de mensagens colocadas em quarentena.

A Figura 13 ilustra a média mensal de *e-mail* classificado com spam bloqueado.

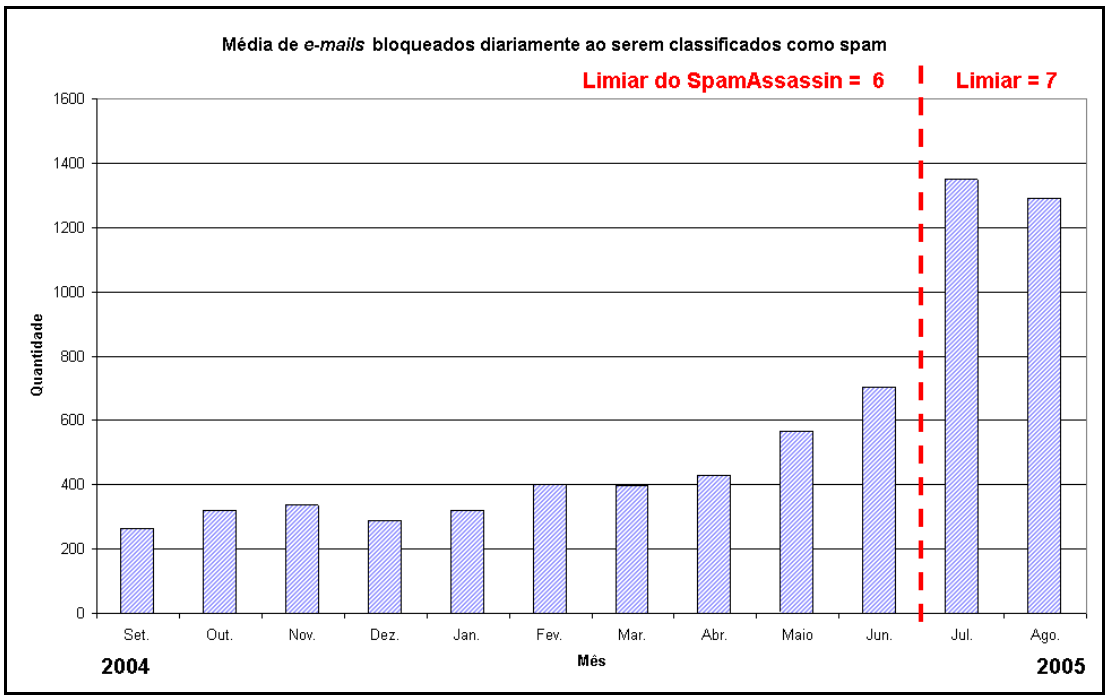

Figura 13 - Média da quantidade diária de spam bloqueado, classificada por mês de ocorrência.

A Figura 14 ilustra a média mensal de código malicioso bloqueado. Apesar da ocorrência de picos em novembro e março, a quantidade diminuiu significativamente a partir de abril.

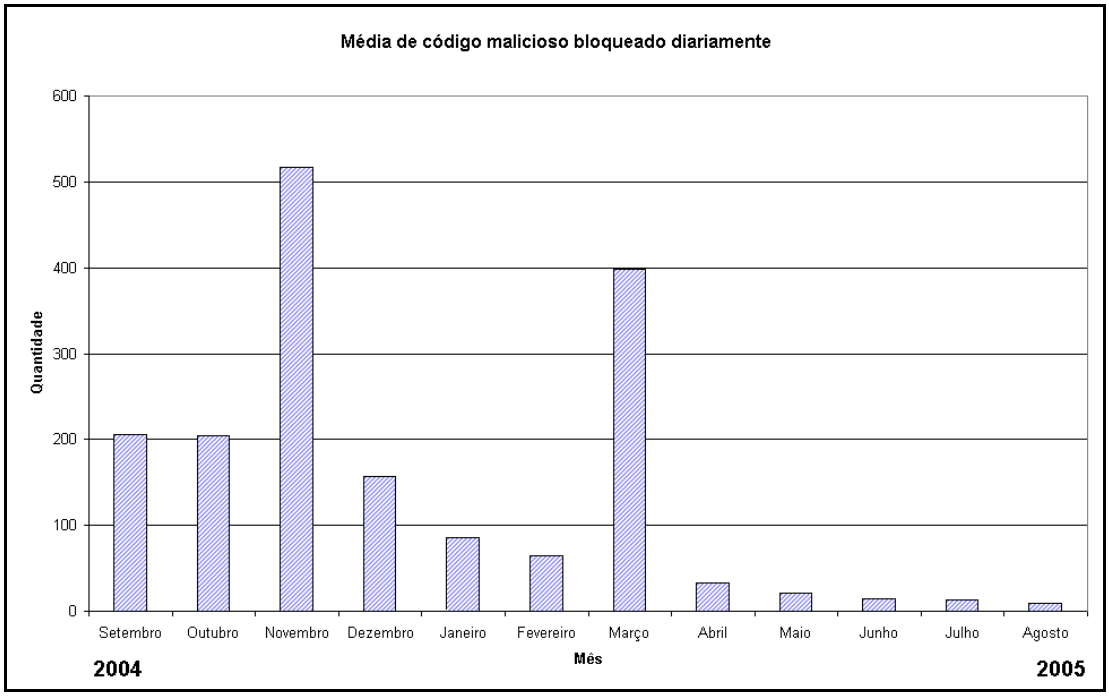

Figura 14 - Média da quantidade diária de código malicioso bloqueado, classificada por mês de ocorrência.

O SpamAssassin, de setembro de 2004 a junho de 2005, foi configurado com o limiar 7. A porcentagem de spam e *e-mail* legítimo está ilustrada na Figura 15.

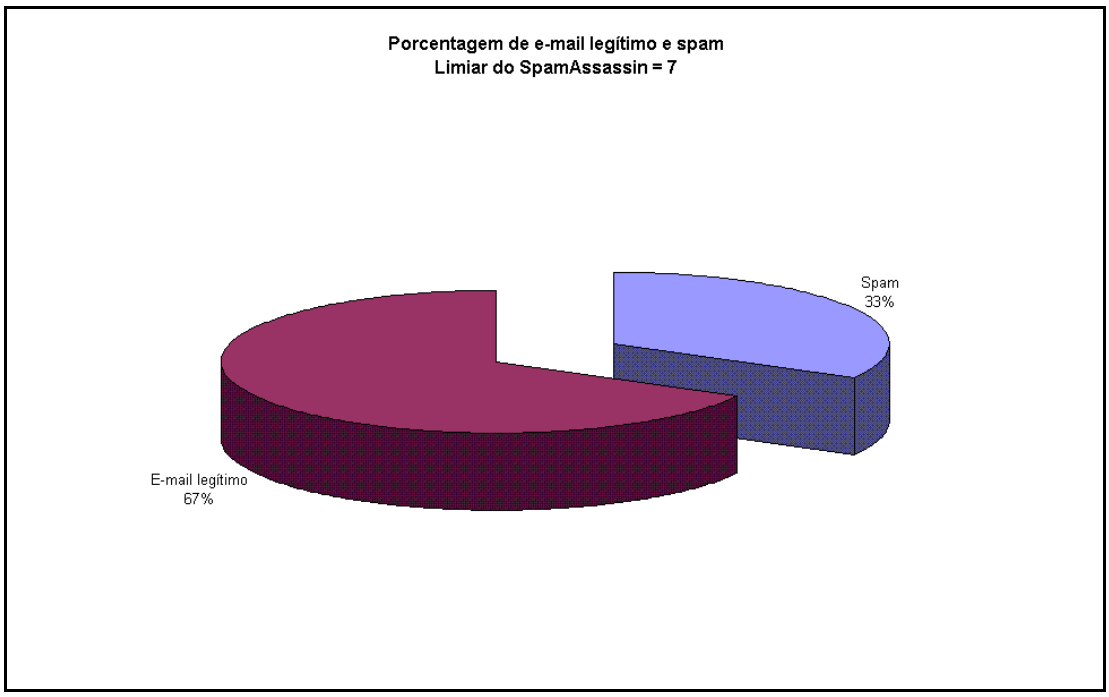

Figura 15 - Porcentagem de *e-mails* classificados como legítimos e spam obtidos com o limiar do SpamAssassin igual a 7.

Por não haver caso registrado de falsos positivos, em julho e agosto de 2005 o SpamAssassin foi configurado com o limiar 6. A porcentagem de spam e *e-mail* legítimo está ilustrada na Figura 16.

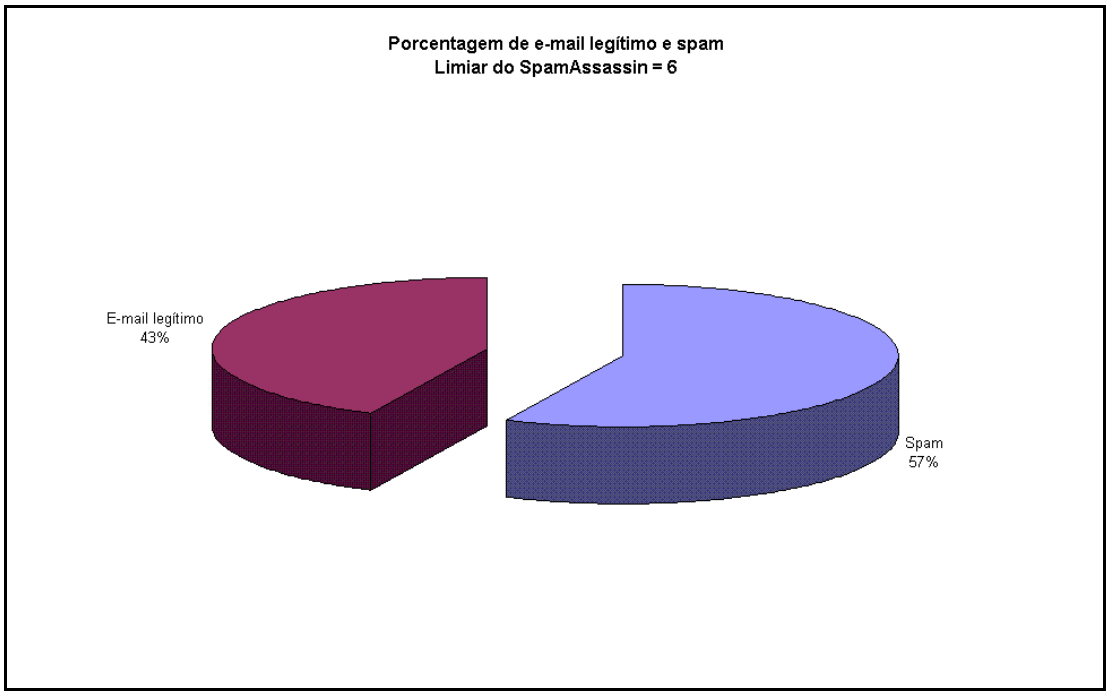

Figura 16 - Porcentagem de *e-mails* classificados como legítimos e spam obtidos com o limiar do SpamAssassin igual a 6.

No período de agosto de 2005 a janeiro de 2006 a carga do sistema foi monitorada para obter uma referência do uso da CPU do servidor cujo *hardware* foi descrito no item 6.3. Para coleta de dados foi utilizado o comando "top", que apresenta o valor da carga do sistema nos últimos 1, 5 e 15 minutos. O valor da carga é a média de processos em estado "pronto" para serem executados. A Figura 17 ilustra a média mensal da carga, desse período, de medidas obtidas a cada 5 minutos. Para complementar, a Figura 18 ilustra a média mensal de tempo ocioso do processador.

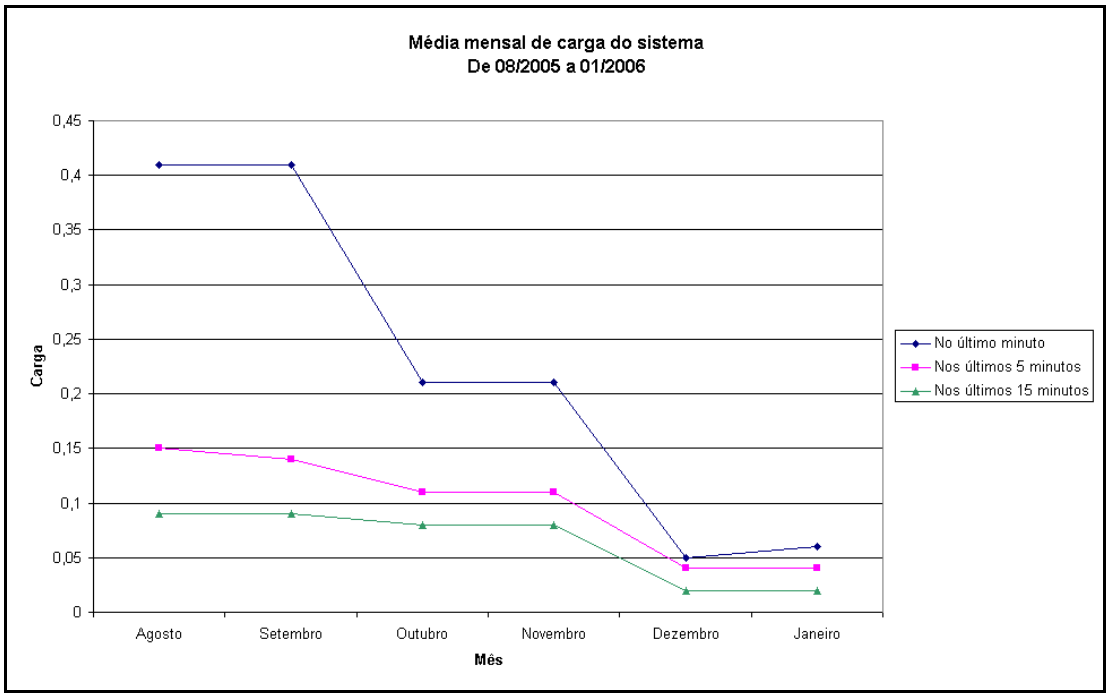

Figura 17 - Média mensal de carga do sistema.

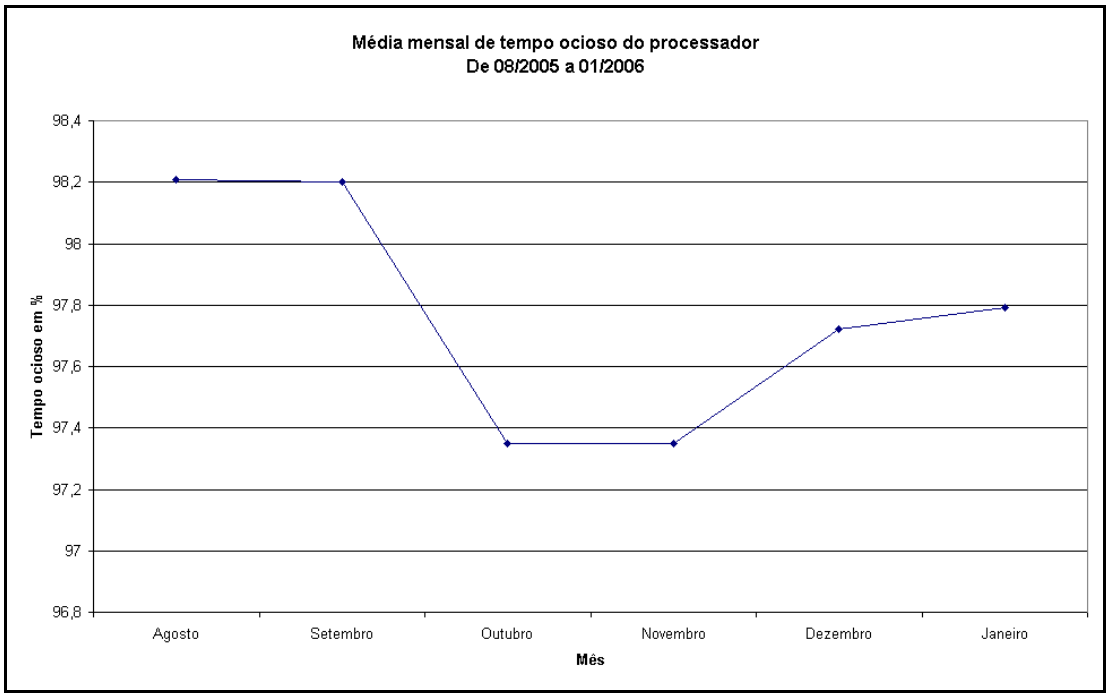

Figura 18 - Média mensal de tempo ocioso do processador.

6.6 Alterações na configuração padrão do sistema anti-spam

O Qmail-scanner bloqueou notificações de leitura de *e-mails* oriundos do provedor UOL. Para solucionar o problema foi preciso desabilitar a verificação de "*Bad MIME Characters*" ao definir a variável "BMC_WHITELIST" conforme explicado no guia de instalação do servidor.

A regra de verificação por palavras-chave do SpamAssassin "subj_illegal_chars" estava sendo acionada em *e-mails* legítimos com caracteres acentuados no campo "assunto". Por padrão, essa regra atribui a pontuação "3.9", que é relativamente alta. Adicionou-se a seguinte linha no arquivo de configuração do SpamAssassin "local.cf" para atribuir a nota "0" quando a regra é acionada:

```
score subj_illegal_chars 0
```

Uma alternativa mais interessante é a alteração da regra para aceitar os caracteres de acentuação, ao invés de alterar a pontuação para 0. Dessa maneira, *e-mails* com caracteres ilegais no assunto ainda seriam identificados.

6.7 Questionário

O suporte técnico do setor de informática está sempre em contato com os usuários e os acompanha de perto no dia-a-dia. O questionário da Figura 19 foi aplicado aos funcionários do suporte técnico de informática para delinear o funcionamento do sistema implantado.

Esse questionário tem como objetivo avaliar o sistema de correio eletrônico com proteção anti-spam e antivírus no período de 01 de setembro de 2004 a 31 de agosto de 2005.

Nome: _____

1) Na sua opinião, como suporte técnico ao usuário, o sistema anti-spam:

a) É satisfatório, pois não bloqueia *e-mails* legítimos.

b) Precisa ser melhorado, pois alguns *e-mails* foram bloqueados indevidamente.

c) Não funciona, pois a quantidade de *e-mails* bloqueados indevidamente foi altíssima.

2) Você atendeu chamados de suporte técnico de reclamação de recebimento de spam em grande quantidade?

a) Nenhum.

b) Poucos.

c) Alguns.

d) Muitos.

3) Você atendeu chamados de suporte técnico de inserção indevida do rótulo "[SPAM]" no campo assunto de *e-mail*:

a) Nenhuma vez.

b) Poucas vezes.

c) Algumas vezes.

d) Muitas vezes.

4) Você atendeu chamados de suporte técnico de reclamação ou dúvida de usuário sobre o não recebimento de *e-mail* que pode ter sido bloqueado pelo sistema?

a) Nenhuma vez.

b) Poucas vezes.

c) Algumas vezes.

d) Muitas vezes.

5) Na sua opinião, o antivírus no servidor de *e-mail* bloqueou:

a) Todos os vírus ou *worms* que se propagam por *e-mail*, porque não foram abertos chamados de suporte técnico para resolver esse tipo de problema.

b) Quase todos os vírus ou *worms* que se propagam por *e-mail*, porque foram abertos poucos chamados de suporte técnico para resolver esse tipo de problema.

c) Uma quantidade razoável de vírus ou *worms* que se propagam por *e-mail*, porque foram abertos alguns chamados de suporte técnico para resolver esse tipo de problema.
d) Poucos vírus, porque foram abertos muitos chamados de suporte técnico para resolver esse tipo de problema.

6) *E-mails* forjados por *spammer*s ou com código malicioso podem deixar usuários confusos. Na sua opinião, a implantação do sistema anti-spam e antivírus no servidor de *e-mail*:
a) Tornou zero ou praticamente zero a quantidade de chamados de suporte técnico para tirar dúvidas relacionadas a *e-mails* suspeitos e/ ou com código malicioso.
b) Reduziu bastante a quantidade de chamados de suporte técnico para tirar dúvidas relacionadas a *e-mails* suspeitos e/ ou com código malicioso.
c) Reduziu um pouco a quantidade de chamados de suporte técnico para tirar dúvidas relacionadas a *e-mails* suspeitos e/ ou com código malicioso.
d) Não mudou em nada a quantidade de chamados de suporte técnico para tirar dúvidas relacionadas a *e-mails* suspeitos e/ ou com código malicioso.

7) De maneira geral, a implantação de um sistema anti-spam e antivírus neste departamento técnico:
a) Trouxe somente benefícios.
b) Trouxe benefícios e novos problemas.
c) Não trouxe benefícios e ainda trouxe novos problemas.

Figura 19 - Questionário aplicado aos funcionários do suporte técnico.

Os dois técnicos de informática, responsáveis pelo atendimento de suporte técnico aos usuários responderam o questionário da seguinte forma:

1) Na sua opinião, como suporte técnico ao usuário, o sistema anti-spam:

Ambos escolheram a alternativa “a”: “É satisfatório, pois não bloqueia *e-mails* legítimos”.

2) Você atendeu chamados de suporte técnico de reclamação de recebimento de spam em grande quantidade?

Um técnico escolheu a alternativa “b”: “Poucos”.

O outro, a alternativa “c”: “Alguns”.

3) Você atendeu chamados de suporte técnico de inserção indevida do rótulo "[SPAM]" no campo assunto de *e-mail*:

Ambos escolheram a alternativa “a”: “Nenhuma vez”.

4) Você atendeu chamados de suporte técnico de reclamação ou dúvida de usuário sobre o não recebimento de *e-mail* que pode ter sido bloqueado pelo sistema?

Ambos escolheram a alternativa “a”: “Nenhuma vez”.

5) Na sua opinião, o antivírus no servidor de *e-mail* bloqueou:

Ambos escolheram a alternativa “b”: “Quase todos os vírus ou *worms* que se propagam por *e-mail*, porque foram abertos poucos chamados de suporte técnico para resolver esse tipo de problema”.

6) *E-mails* forjados por *spammers* ou com código malicioso podem deixar usuários confusos. Na sua opinião, a implantação do sistema anti-spam e antivírus no servidor de *e-mail*:

Ambos escolheram a alternativa “b”: “Reduziu bastante a quantidade de chamados de suporte técnico para tirar dúvidas relacionadas a *e-mails* suspeitos e/ ou com código malicioso”.

7) De maneira geral, a implantação de um sistema anti-spam e antivírus neste departamento técnico:

Ambos escolheram a alternativa “a”: “Trouxe somente benefícios”.

Capítulo 7

7 Conclusão

Durante o período estipulado para o estudo, notou-se um aumento significativo de spam bloqueado pelo sistema. De setembro de 2004 a junho de 2005, a média da quantidade de spam bloqueado diariamente classificada por mês de ocorrência, teve um mínimo de 899 e um máximo de 1509. Em julho e agosto de 2005, o valor subiu para 2112 e 2534 respectivamente, atingindo um acréscimo de cerca de 60% no volume de spam sobre o valor da média mais alta dos meses anteriores.

O aumento da porcentagem de spam bloqueado em julho e agosto de 2005 coincidem com a diminuição do limiar do SpamAssassin de 7 para 6. Nos meses em que o sistema operou com o limiar 7, 33% do total de *e-mails* foram classificados como sendo spam. Ao alterar o limiar para 6, 57% foram classificados como sendo spam.

A proporção de spam bloqueado em julho e agosto de 2005 foi de 64% e 51% respectivamente e aproximaram com o valor de 61% apresentado pela VIII edição do "*Internet Security Threat Report* – Relatório Regional América Latina" da Symantec (2005a), referente ao 1º semestre de 2005. A estatística da MessageLabs publicada no documento "*MessageLabs Intelligence Report: Spam Intercepts Timeline*" (MESSAGELABS, 2005) informa que nesses dois meses a proporção foi de 65,16% e 65,10%, respectivamente.

No total, o sistema foi capaz de bloquear 47,75% de *e-mails* indesejados, sem casos de falsos positivos, que seriam recebidos por usuários que:

- Perderiam tempo para separar os *e-mails* legítimos de spam;
- Poderiam se distrair com propagandas;
- Poderiam abrir chamados de suporte técnico para tirar dúvidas de *e-mails* forjados ou com código malicioso.

7.1 Análise do resultado do questionário

Segundo os funcionários responsáveis pelo suporte técnico não ocorreram casos de falsos positivos e nem reclamações de *e-mails* que poderiam ter sido bloqueados indevidamente pelo sistema. Algumas vezes, usuários solicitaram para verificar se houve bloqueio indevido de *e-mail*, mas em nenhum caso foi constatado falso positivo.

E-mails de notificação de leitura enviados pelo provedor UOL foram bloqueados indevidamente pelo Qmail-scanner, mas a verificação de "BMC" foi desabilitada conforme explicado na instalação do sistema, resolvendo o problema definitivamente.

Conforme a equipe de suporte técnico, usuários não se queixaram de *e-mails* legítimos rotulados com "[SPAM]" no campo "assunto". Logo após a migração de todas as contas de *e-mail*, os usuários receberam treinamento sobre o funcionamento do servidor de *e-mail* com sistema anti-spam e antivírus. Os usuários foram orientados a acionar a equipe de informática em caso de rotulagem incorreta.

Em relação ao antivírus, constatou-se que o servidor de *e-mail* conseguiu bloquear quase todos os *e-mails* com código malicioso. O antivírus é incapaz de detectar novas ameaças enquanto não estiver com o banco de dados de assinaturas atualizados. Esse processo depende da agilidade da empresa desenvolvedora do *software* em obter uma amostra do código malicioso e gerar uma vacina. Depende também do antivírus receber essa atualização. A verificação da disponibilidade de novas vacinas foi agendada para ser feita duas vezes ao dia: às 8h e às 20h. Além disso, *e-mails* com arquivos anexos com as extensões .vbs, .scr, .wsh, .hta, .pif e .com são bloqueados por estarem comumente relacionados com código malicioso.

No ambiente em que o servidor foi implantado, os computadores também estavam equipados com software antivírus proprietário, fator que aumentou a segurança dos usuários em relação a códigos maliciosos.

Para aumentar a eficácia no bloqueio de código malicioso, a freqüência da verificação da disponibilidade de novas assinaturas pode ser aumentada e mais tipos de arquivos relacionados com código malicioso podem ser bloqueados como .exe e .zip.

A implantação de servidor de *e-mail* com anti-spam e antivírus conseguiu reduzir bastante a quantidade de chamados de suporte técnico para tirar dúvidas relacionadas a *e-mails* suspeitos e/ ou com código malicioso. Esse resultado já era esperado com o sistema de quarentena. Quanto menos *e-mails* indesejados os usuários recebem, menos problemas são causados por eles.

Para finalizar, eles afirmaram que o sistema trouxe somente benefícios, mas ainda há usuários que se queixam de recebimento de spam em grande quantidade.

Como sugestão de melhora do sistema, os falsos negativos recebidos pelos usuários devem ser usados para realimentar o banco de dados do algoritmo bayesiano para aumentar as chances de bloquear os *e-mails* que conseguem passar pelo filtro.

7.2 Avaliação do servidor

Itens a favor:

- Baixo custo;
- Estabilidade (não ocorreram travamentos dos programas utilizados);
- Bom desempenho (no ambiente em estudo);
- Fácil administração (não foi necessário fazer a manutenção de regras de filtragem);

Itens contra:

- Processo de instalação e configuração do sistema relativamente complexo;
- Na fase de teste, a monitoração de *e-mails* em quarentena consumiu bastante tempo;
- SpamAssassin faz uso intensivo de CPU e memória RAM. (NARCISO, 2005) (MAFRA; FRAGA, 2004);
- Dependendo do *software*, não há documentação em português.

7.3 Proposta para prevenção e combate ao spam

Buscar proteção de spam e código malicioso:

- Com a implantação de software anti-spam no servidor de *e-mail*;
- Com a instalação de software antivírus nos computadores de usuários e no servidor de *e-mail* para se proteger de código malicioso;
- Com o uso de *firewall* pessoal nos computadores de usuários;
- Com política para manter *softwares* atualizados.

Conscientizar os usuários:

- Treinar e orientar os usuários conforme as recomendações para o uso do *e-mail* do capítulo 4 e também com o auxílio da "Cartilha de segurança para a Internet" do CERT.br.

Impedir que computadores da instituição sejam envolvidos com o envio de spam:

- Com o servidor de *e-mail* configurado para fazer o *relay* apenas de mensagens oriundas da faixa de IP da instituição ao qual pertence;
- Com o acesso aos *e-mails* de fora da instituição via *webmail* com conexão segura ou via SMTP-AUTH, ambos com política de senhas bem definida para dificultar ataques de dicionário;
- Com o uso de servidores *proxy*, quando necessário, configurados adequadamente para não serem explorados por *spammers*;
- Com regra de *firewall* da instituição para impedir que computadores de usuários se conectem a endereços externos à instituição na porta 25. Isso obriga o uso do servidor SMTP da instituição para o envio de *e-mails* e dificulta o envio de spam por computadores infectados por código malicioso de *hackers* e *spammers*, que geralmente operam dessa maneira.

7.4 Avaliação geral do sistema implantado

O sistema implantado conseguiu reduzir spam e código malicioso de modo a beneficiar todos os envolvidos: usuários, suporte técnico e administrador do servidor.

A redução de *e-mails* indesejados sem casos confirmados de falsos positivos foi importante para manter a credibilidade do sistema de correio eletrônico, que se popularizou pela rapidez, praticidade e confiabilidade na entrega de *e-mails*.

Altos índices de falsos positivos nas empresas e instituições poderiam acarretar numa situação insólita: enviar *e-mail* e depois confirmar com o destinatário por telefone se a mensagem foi recebida.

O suporte técnico se beneficiou, além da redução de spam, da diminuição de chamados de suporte técnico para atender casos de *e-mails* forjados por “spammers” ou com código malicioso.

Do ponto de vista da administração do servidor, a compreensão do funcionamento do sistema e a implantação foram etapas que consumiram bastante tempo. Porém após a implantação, praticamente não houve intervenções por parte do administrador para manter o sistema em funcionamento.

Apesar de alguns usuários ainda reclamarem do recebimento de grandes quantidades de spam, o sistema é válido já que trouxe apenas benefícios, sem onerar o administrador em atividades que envolveriam a análise de spam para fazer a manutenção de técnicas anti-spam.

7.5 Sugestões de melhorias no sistema

No sistema proposto, não há um mecanismo que facilita os usuários a realimentarem o filtro bayesiano para melhorar a classificação dos *e-mails*.

A freqüência de regras por palavras-chave acionadas em *e-mails* legítimos e spam pode ser objeto de estudo. Regras freqüentemente acionadas em spam e *e-mails* legítimos podem ser desabilitadas para diminuir o uso do processador. Regras que aparecem com freqüência em spam podem ter a pontuação reajustada para valor mais alto. Regras que aparecem com freqüência em *e-mails* legítimos podem ser desabilitadas também ou ter a pontuação reajustada para valor mais baixo. Isso poderia resultar num sistema mais eficaz e eficiente.

Uma área de quarentena privada permitiria que cada usuário fizesse verificações sempre que necessário para saber se ocorreu algum caso de falso positivo. Se usuários suspeitarem do resultado da classificação dos *e-mails* feita pelo sistema anti-spam, eles não dependeriam do administrador. No entanto, essa solução poderia resultar no aumento de chamados técnicos para elucidar dúvidas sobre *e-mails* forjados por *spammers* ou *hackers*, porque os usuários teriam contato com todos os *e-mails* indesejados.

O uso do sistema de *auto-whitelist* também precisa ser analisado. Apesar de trazer benefícios, pode ser explorado pelos *spammers*. *E-mails* enviados por *spammers* forjados para se parecerem com mensagens legítimas, sem características de spam, podem obter pontuações baixas e passar pelo filtro. Quando o spam é enviado realmente, o sistema de *auto-whitelist* pode facilitar a passagem desses *e-mails* pelo filtro devido à média dos *e-mails* enviados anteriormente.

7.6 Propostas para novos trabalhos

Dimensionamento de servidores de *e-mail* com sistema anti-spam.

Estudo de novas técnicas anti-spam.

Estudo para definir as pontuações de regras por palavras-chave dependendo da freqüência que aparecem em spam e *e-mails* legítimos.

Referências

APWG. **Phishing Attack Trends Report**. 2004. Disponível em: <http://www.antiphishing.org/APWG_Phishing_Attack_Report-Jul2004.pdf>. Acesso em: 21 jan. 2006.

BERSTEIN, D. **Qmail: the internet's MTA of choice**. 2001? Disponível em: <http://cr.yp.to/qmail.html>. Acesso em: 25 jan. 2006.

CAZABON, C.; SILL, D.; BRAUER, H.; SAMUEL, P.; NELSON, R. **Netqmail**. 2005. Disponível em: <http://www.qmail.org/netqmail/>. Acesso em: 25 jan. 2006.

Center for Democracy & Technology. **Why am I getting all this spam? Unsolicited commercial e-mail research six month report**. 2003. Disponível em: <http://www.cdt.org/speech/spam/030319spamreport.shtml>. Acesso em: 20 fev. 2006.

CERT. **Advisory CA-2004-04 Love Letter Worm**. 2000. Disponível em: <http://www.cert.org/advisories/CA-2000-04.html>. Acesso em: 19 jan. 2006.

CERT.BR. **Cartilha de segurança para a Internet**. 2005. Disponível em: <http://cartilha.cert.br/download/>. Acesso em: 23 fev. 2006.

CERT.BR. **Estatísticas do CERT.br -- Spam**. 2006. Disponível em: <http://www.cert.br/stats/spam/>. Acesso em: 07 fev. 2006.

CHIEN, E.; EWELL, B. **Security response – VBS.LoveLetter and variants**. 2001?. Disponível em: <http://www.symantec.com/avcenter/venc/data/vbs.loveletter.a.html>. Acesso em: 29 jan. 2006.

CHRISTIANSEN T.; WALL, L. **Sys::Syslog**. 2002? Disponível em: <http://search.cpan.org/dist/Sys-Syslog/Syslog.pm>. Acesso em: 27 jan. 2006.

CLAMAV. **ClamAV: Abstract**. 200? Disponível em: <http://www.clamav.net/abstract.html#pagestart>. Acesso em: 30 jan. 2006.

DSBL. **DSBL: Frequently Asked Questions**. 200? Disponível em: <http://dsbl.org/faq>. Acesso em: 15 fev. 2006.

FTC. **False claims in spam – A report by the FTC's division of marketing practices**. 2003. Disponível em: <http://www.ftc.gov/reports/spam/030429spamreport.pdf>. Acesso em: 21 jan. 2006.

GFI. **Why Bayesian filtering is the most effective anti-spam technology**. 2006. Disponível em: <http://www.gfi.com/whitepapers/why-bayesian-filtering.pdf>. Acesso em: 19 jan. 2006.

GNU Project. **O que é software livre?** 2005. Disponível em: <http://www.gnu.org/philosophy/free-sw.pt.html>. Acesso em: 21 jan. 2006.

GRAHAM, Paul. **A Plan For Spam**. 2002. Disponível em: <http://www.paulgraham.com/spam.html>. Acesso em: 01 fev. 2006.

GRAHAM, Paul. **Better Bayesian Filtering**. 2003. Disponível em: <http://www.paulgraham.com/better.html>. Acesso em: 01 fev. 2006.

GRUPO BRASIL ANTI-SPAM. **Cartilha AntiSPAM**. 200? Disponível em: <http://brasilantispam.locaweb.com.br/main/cartilha.htm>. Acesso em: 06 fev. 2006.

GUENTER, B. **QMAILQUEUE patch for qmail-1.03**. 1999. Disponível em: <http://www.qmail.org/qmailqueue-patch>. Acesso em: 25 jan. 2006.

HAAR, J. **Qmail-Scanner: Content Scanner for Qmail**. 2005. Disponível em: <http://qmail-scanner.sourceforge.net/>. Acesso em: 25 jan. 2006.

HAAR, J. **Qmail-Scanner: Frequently Asked Questions**. 2004. Disponível em: <http://qmail-scanner.sourceforge.net/FAQ.php>. Acesso em: 30 jan. 2006.

HIETANIEMI, Jarkko; WEGSCHEID, D.; SCHERTLER, R.; AAS, G. **TIME::HiRes**. 2005. Disponível em: <http://search.cpan.org/dist/Time-HiRes/HiRes.pm>. Acesso em: 27 jan. 2006.

HORMEL FOODS CORPORATION. **SPAM and the Internet**. 200? Disponível em: <http://www.spam.com/ci/ci_in.htm>. Acesso em: 21 jan. 2006.

HUGHES, Craig et al. **The Apache SpamAssassin Project**. 2005. Disponível em: <http://spamassassin.apache.org/>. Acesso em: 25 jan. 2006.

KOJM, T. et al. **Clam Antivirus**. 2005. Disponível em: <http://www.clamav.net/>. Acesso em: 25 jan. 2006.

LOWE, R. **History Of Spam**. 200?a. Disponível em: <http://www.mailmsg.com/SPAM_history.htm>. Acesso em: 21 jan. 2006.

LOWE, R. **Monty Python Spam Song**. 200?b. Disponível em: <http://www.mailmsg.com/SPAM_python.htm>. Acesso em: 21 jan. 2006.

MAFRA, Paulo Manoel; FRAGA, Joni da Silva. **Configuração de filtros para controle de spam e vírus**. 2004. Disponível em: <http://www.mafra.eti.br:8080/artigos/mafra_ssi04.pdf>. Acesso em: 05 fev. 2006.

MARQUESS, P. **DB_FILE**. 2005. Disponível em: <http://www.cpan.org/modules/by-module/DB_File/DB_File-1.814.readme>. Acesso em: 27 jan. 2006.

MCAFEE. **Press Release – McAfee.com commemorates one-year anniversary of the infamous "ILOVEYOU" virus outbreak with free Virus Scan and protection**. 2001. Disponível em: <http://uk.mcafee.com/root/genericURL_genericLeftNav.asp?genericURL=/common/pressIncludes/Apr302001_1.asp&genericLeftNav=/common/en-gb/html_files/aboutUs_nav.asp>. Acesso em: 21 jan. 2006.

MCWILLIAMS, B. **Swollen orders show spam's allure**. Wired News, agosto, 2003. Disponível em: <http://www.wired.com/news/business/1,59907-0.html>. Acesso em: 07 fev. 2006.

MESSAGELABS. **MessageLabs Intelligence Report: Spam Intercepts Timeline**. 2005. Disponível em: <http://www.messagelabs.com/portal/server.pt/gateway/PTARGS_0_0_456_454_-454_43/http%3B/0120-0176-CTC1%3B8080/publishedcontent/publish/_dotcom_libraries_en/files/monthly_reports/messagelabs_intelligence_report__spam_intercepts_timeline_november_2005_uk_5.pdf>. Acesso em: 20 fev. 2006.

MICROSOFT. **Microsoft reforça combate ao spam**. 2003. Disponível em: <http://www.microsoft.com/brasil/pr/2003/spam.asp>. Acesso em: 21 jan. 2006.

MÓDULO SECURITY. **Spam: quanto se perde e como evitar? – Parte 2**. 2003. Disponível em: <http://www.modulo.com.br/comum/docs_iii_pv.jsp?catid=7&objid=1911&idiom=0&pagenumber=0>. Acesso em: 22 fev. 2006.

MOVIMENTO ANTI-SPAM BRASILEIRO. **O que é SPAM?** 200? Disponível em: <http://www.spambr.org/oquee.html>. Acesso em: 06 fev. 2006.

NARCISO, Marcelo Gonçalves. **Mecanismos contra spams para servidor de correio eletrônico**. 2005. Disponível em: <http://www.cnptia.embrapa.br/modules/tinycontent3/content/2004/doc50.pdf>. Acesso em: 02 fev. 2006.

PAIVA, M. A. L. **O imprestável chamado spam**. 2002. Disponível em: <http://www.legiscenter.com.br/materias/materias.cfm?ident_materias=134>. Acesso em: 21 jan. 2006.

PEACE, S. **Qmail-Scanner and ClamAV HowTo**. 2004. Disponível em: <http://www.ibiblio.org/pub/Linux/docs/HOWTO/other-formats/pdf/Qmail-ClamAV-HOWTO.pdf>. Acesso em: 30 jan. 2006.

POSTINI. **Postini Resource Center - Message Threats**. 2006. Disponível em: <http://www.postini.com/stats/>. Acesso em: 08 fev. 2006.

PRAKASH, V. **Vipul's Razor – spam should not be propagated beyond necessity**. 2005. Disponível em: <http://razor.sourceforge.net/>. Acesso em: 25 jan. 2006.

QMAIL INFO. **Reformime**. 2005. Disponível em: <http://www.qmailinfo.org/index.php/Reformime>. Acesso em: 25 jan. 2006.

SAMUEL, P. **Qmail an internet mail transport agent or qmail – fast, secure, reliable. Pick any three!** 2003. Disponível em: <http://www.gormand.com.au/peters/tutorials/qmail-oclug-2003.pdf>. Acesso em: 25 jan. 2006.

SILL, D. **Life with qmail**. 2006. Disponível em: <http://www.lifewithqmail.org/lwq.pdf>. Acesso em: 25 jan. 2006.

SIMPSON, Mark. **freshmeat.net: project Details for tnef**. 2005. Disponível em: <http://freshmeat.net/projects/tnef/>. Acesso em: 15 fev. 2006.

SIMPSON, P. **Spoofed identities: virus, spam or scam?** Computer Fraud & Security, Volume 2003, Issue 10, outubro de 2003, Pages 6-8. Disponível em: <http://www.sciencedirect.com/science/journal/13613723>. Acesso em: 25 jan. 2006.

SPAMASSASSIN. **Mail::SpamAssassin::Conf**. 2004? Disponível em: <http://spamassassin.apache.org/full/3.0.x/dist/doc/Mail_SpamAssassin_Conf.html>. Acesso em: 03 fev. 2006.

SYMANTEC. **Internet Security Threat Report – Relatório Regional América Latina**. 2005a. Disponível em: <http://www.symantec.com/region/br/enterprisesecurity/threat_report/Resumo_ISTR8.pdf>. Acesso em: 22 fev. 2006.

SYMANTEC. **Security response – W32.Bobax.D**. 2005b. Disponível em: <http://securityresponse.symantec.com/avcenter/venc/data/w32.bobax.d.html>. Acesso em: 06 fev. 2006.

SYMANTEC. **Security response – W32.Klez.H@mm**. 2004b. Disponível em: <http://securityresponse.symantec.com/avcenter/venc/data/pf/w32.klez.h@mm.html>. Acesso em: 13 fev. 2006.

SYMANTEC. **Security response – W32.Mydoom.BI@mm**. 2005c. Disponível em: <http://securityresponse.symantec.com/avcenter/venc/data/w32.mydoom.bi@mm.html>. Acesso em: 06 fev. 2006.

SYMANTEC. **Security response – W32.Sobig.F@mm**. 2004a. Disponível em: <http://securityresponse1.symantec.com/sarc/sarc-intl.nsf/html/br-w32.sobig.f@mm.html>. Acesso em: 06 fev. 2006.

TANENBAUM, A. **Redes de computadores**. Tradução [3. ed. original] Insight Serviços de Informática. Rio de Janeiro: Campus, 1997.

TECHTARGET. **Inode**. 2003. Disponível em: <http://searchopensource.techtarget.com/sDefinition/0,290660,sid39_gci213729,00.html>. Acesso em: 06 mar. 2006.

TECHTARGET. **Phishing**. 2005a. Disponível em: <http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci916037,00.html>. Acesso em: 22 jan. 2006.

TECHTARGET. **Spam - a Whatis.com definition**. 2005b. Disponível em: <http://searchmobilecomputing.techtarget.com/sDefinition/0,290660,sid40_gci213031,00.html>. Acesso em: 07 fev. 2006.

TEIXEIRA, Renata Cicilini. **Combatendo o spam – Aprenda como evitar e bloquear e-mails não solicitados**. São Paulo: Novatec, 2004.

TELLES, Rodrigo Pereira. **Qmail – Guia de consulta**. 2005. Disponível em: <http://qmail.telles.org/qmail-guiadeconsulta/index.html>. Acesso em: 06 mar. 2006.

TORIBIO, S. **Qmail-scanner-1.25st (st patch)**. 2005. Disponível em: <http://toribio.apollinare.org/qmail-scanner/>. Acesso em: 25 jan. 2006.

VIDAL, A. G. R.; FAVA, V. L. **Perfil da Indústria digital**. 2002. Disponível em: <http://www.idigital.fea.usp.br/idigital/repositorio/0/documentos/idigital2002.pdf>. Acesso em: 22 jan. 2006.

WEBOPEDIA. **The difference between a vírus, worm and trojan horse?** 2004. Disponível em: <http://www.webopedia.com/DidYouKnow/Internet/2004/virus.asp>. Acesso em: 05 abr. 2006.

WOOD, P. **The convergence of viruses and spam**. MessageLabs. 2004? MessageLabs. Disponível em: <http://www.messagelabs.com/intelligence/whitepapers/pdf/SobigWhitePaper.pdf>. Acesso em: 24 ago. 2004.

WOOD, P. **Save yourself from eternal spamnation.** MessageLabs, abril, 2004. Disponível em: <http://www.messagelabs.com/intelligence/whitepapers/pdf/SpamWhitePaper-US.pdf>. Acesso em: 24 ago. 2004.

YIN, Robert K. **Estudo de caso: planejamento e métodos**. Tradução [3. ed.] Daniel Grassi. Porto Alegre: Bookman, 2005.

Lista de RFCs

RFC 1939 - Post Office Protocol - Version 3

RFC 2045 - Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies

RFC 2046 - Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types

RFC 2047 - MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text

RFC 2048 - Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures

RFC 2049 - Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples

RFC 2821 - Simple Mail Transfer Protocol

RFC 2822 - Internet Message Format

RFC 3501 - INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1

Anexo A – Exemplos de e-mails indesejados

ANEXO A1 – *Phishing scam* enviado em nome do Itaú

O *e-mail* abaixo não foi enviado pelo Itaú, pois se trata de um *phishing scam*, ou seja, um golpe. A pessoa que recebe o *e-mail* e que é correntista do Itaú é convencida a entrar no *site* do banco e a fazer um cadastro, colocando o número da conta e a senha. Na Figura 20, apesar do rótulo do *link* ser http://www.itau.com.br, ao posicionar o ponteiro do *mouse* sobre o *link* é possível constatar que se trata do endereço IP 130.234.190.56 porta 8090 conforme indicado na barra de status, da Figura 21. Há casos em que a informação da barra de status também é camuflada por código *JavaScript*.

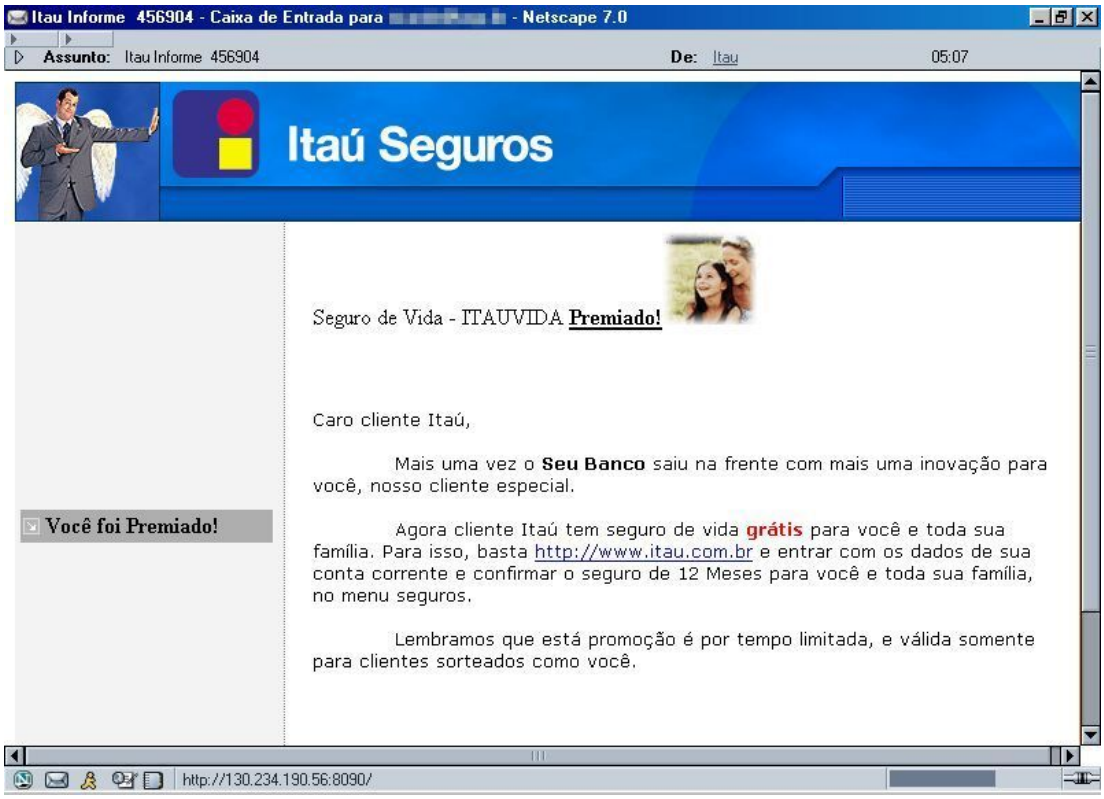

Figura 20 - *Phishing scam* enviado em nome do Itaú.

A Figura 21 apresenta a página inicial de uma cópia do *site* do Itaú forjada pelo *hacker*. Note que na barra de endereço ao invés de aparecer o endereço da instituição está um endereço IP que não está registrado como sendo do Itaú.

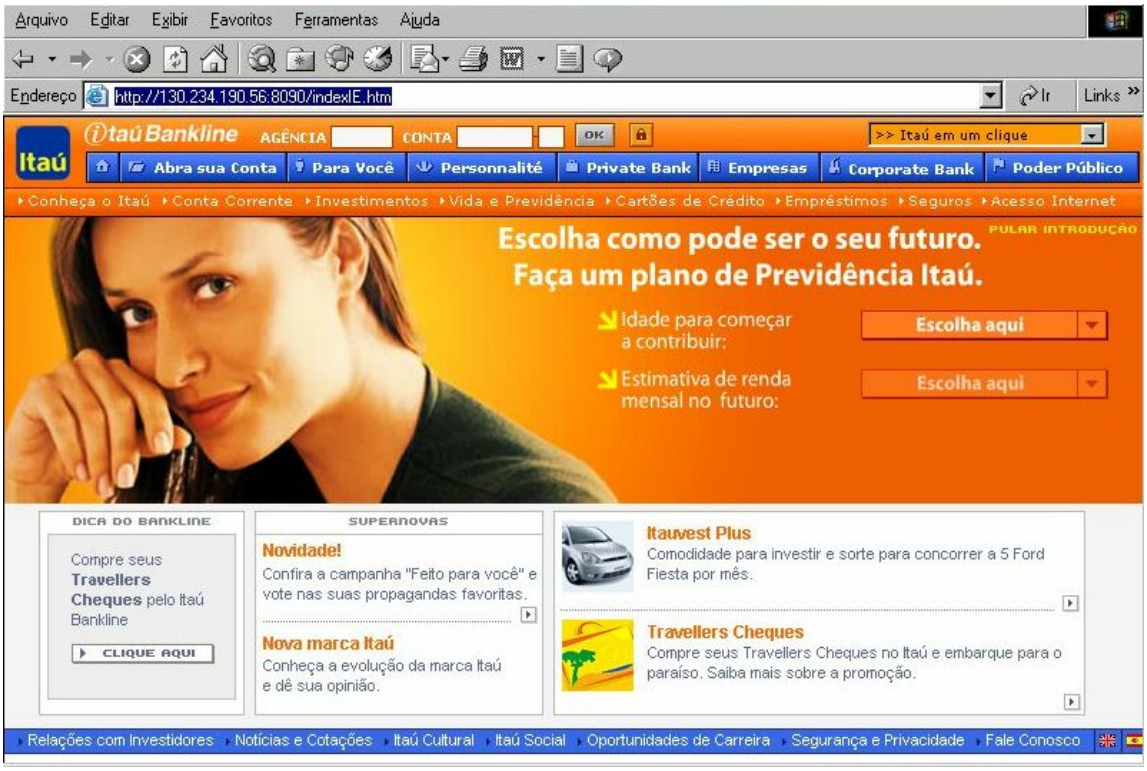

Figura 21 - *Site* do *hacker* utilizado no *phishing scam*.

A página inicial do *site* oficial do Itaú está sendo exibida na Figura 22. É importante observar que pelo visual do *site* não é possível determinar qual é o oficial e qual é a cópia.

Figura 22 - *Site* oficial do Itaú.

ANEXO A2 – "*Nigerian 419 scam*"

FROM THE DESK OF: DR. HABIB ALI
MANAGER CREDIT AND FOREIGN BILLS.
UNION BANK PLC.
Private Email(habibali@123.com)

Attn: Sir

My name is Habib Ali the manager, credit and foreign bills of Union Bank Nigeria Plc. I am writing in respect of a foreign customer of my bank with account number 14-255-2004/utb/t who perished in a plane crash Korean Air Flight 801] with the whole passengers aboard on August 6,1997.

Since the demise of this client customer, I personally have watched with keen interest to see the next of kin but all has proved abortive as no one has come to claim his funds of USD$.30 m, [Thirty million United States dollars] which has been with my branch for a very long time. On this note, I decided to seek for a foreigner who I can present as the next of kin hence no one has come up to be the next of kin. And the banking ethics here does not allow such money to stay more Than seven banking years, after which the money will be called into the bank treasury as unclaimed funds after this period. In view of this I got your contact from a business contact enquiry.

Note that you do not need to have similar names with the late beneficiary. I will give you 25% of the total. Upon the receipt of your response, I will commence on manipulating your information into the file as the next of kin and I will send you information on how we intend to actualize our objective in a legal manner.

I will not fail to bring to your notice that this business is hitch free and that you should not entertain any fear as all modalities for fund transfer can be finalized within five banking days, after your claim is established as the certified next of kin to the deceased.

When you receive this letter. Kindly, send me your private Tel/Fax Numbers through my private email habibali@123.com above for quick communication
Awaiting your favorable response.

Yours truly,

Dr Habib Ali
Private Email(habibali@123.com)

ANEXO A3 – Corrente

O exemplo de corrente abaixo está sendo apresentado, conforme recebido, sem alterações. Note o sinal “>” no início de cada linha, indicando que a corrente já foi encaminhada outras vezes.

Assunto:FW: Tem Que Ter Fe...
Wed, 01 Sep 2004 15:26:14 +0000

>Date: Wed, 1 Sep 2004 17:05:34 +0200
> > > Vamos acreditar!!!
> > > SANTO ANJO DO SENHOR
> > > So funciona no dia.
> > > Funciona no dia que voce receber a mensagem.
>Sei que você pode pensar que TALVEZ SEJA uma bobagem, mas fiquei impressionado c/ a precisao do horario.
> Vamos conferir se acontecera alguma coisa. Os
> > anjos existem. Mas algumas vezes nao possuem asas e passamos a
>chama-los de amigos como voce...
> > Passe, se seu coracao pedir, para todos os seus amigos de verdade.
> >Algo bom vai lhe acontecer hoje... as 11:11 h. da
> > noite. Isto nao e uma brincadeira, alguem vai telefonar ou falar com voce, aquilo
> > que tem esperado ouvir.
> > Procure nao quebrar esta corrente. Mande esta
> > mensagem para sete pessoas, no minimo...
> > Santo Anjo do Senhor, meu zeloso guardador, se a ti me confiou a piedade
> > Divina, sempre me rege, me guarda, me governa, me ilumina. Amem!
> > Faca o seu pedido agora

ANEXO A4 – Boato

O exemplo de boato abaixo está sendo apresentado, conforme recebido, sem alterações. Note o sinal “>” no início de cada linha, indicando que esse boato já foi encaminhado outras vezes.

Subject: Enc: novo vírus
Date: Mon, 30 Aug 2004 17:34:12 –0300

> > Pessoal,
> > Não sei se essa mensagem é verdadeira, em todo caso, em se tratando de
> > vírus, é melhor não vacilar!!!
> >
> >
> >
> > AVISO IMPORTANTE!!!
> >
> > ATENÇÃO!!!
> >
> >
> > Durante as próximas semanas fiquem atentos e não abram nenhum e-mail com o
> > arquivo anexo "Últimas de Atenas", independentemente do quem lhe enviou o
> > e-mail.
> >
> >
> >
> > POR FAVOR FAÇA CIRCULAR ISTO ENTRE TODA SUA FAMILIA E AMIGOS.
> >
> >
> >
> > Não ABRIR o anexo "Últimas de Atenas".
> > É um vírus que **abre uma tocha** no seu micro e queima todo o seu disco
> rígido
> > C:
> >
> > Este vírus vem de uma pessoa conhecida da sua lista de endereços.
> >

> >
> >
> > Por favor, envie este e-mail a todos seus contatos.
> > É preferível receber 25 vezes esta mensagem que perder tudo.
> >
> >
> >
> > Se você receber um correio chamado "Últimas de Atenas.com" você nao
> > abra!!!!
> >
> > Apague imediatamente! Este vírus suprime os arquivos inteiros de seu
> > computador.
> >
> >
> >
> >
> > Esta mensagem e uma correspondencia reservada. Se voce a recebeu por
> > engano,
> > por favor desconsidere-a.
> > O sistema de mensagens da Internet nao e considerado seguro ou livre de
> > erros. Esta instituicao
> > nao se responsabiliza por opinioes ou declaracoes veiculadas atraves de
> > e-mails.

ANEXO A5 – Notificação de bloqueio de *e-mail* com código malicioso

A notificação de bloqueio de *e-mail* com código malicioso apresentada abaixo foi enviada para o endereço de *e-mail* de remetente obtido pelo vírus no catálogo de endereço do computador infectado.

Essa situação gera dúvidas e o suporte técnico é acionado para examinar se o computador da pessoa que recebeu a notificação realmente está comprometido. Depois de se gastar o tempo do técnico e do responsável pelo computador, constata-se que não havia problema algum.

Assunto: Alerta de Vírus
De: não_responda@siteplanet.com.br
Data:18/06/04 18:36

A L E R T A D E V Í R U S

Esta é uma mensagem automática, não é preciso respondê-la!

O servidor de e-mails da SitePlanet encontrou vírus no e-mail
que você enviou para "clinivet".
Por medida de segurança, o e-mail não foi entregue!
Remova o vírus e envie o e-mail novamente.

ATENÇÃO: Caso você não tenha enviado tal e-mail, por favor,
desconsidere este alerta.

Anexo B – Arquivos de configuração

ANEXO B1 – Arquivo de configuração do ClamAV /etc/clamav.conf

```
##
## Example config file for the Clam AV daemon
## Please read the clamav.conf(5) manual before editing this file.
##


# Comment or remove the line below.
#Example

# Uncomment this option to enable logging.
# LogFile must be writable for the user running the daemon.
# Full path is required.
###LogFile /tmp/clamd.log
LogFile /var/log/clamd/clamd.log

# By default the log file is locked for writing - the lock protects against
# running clamd multiple times (if want to run another clamd, please
# copy the configuration file, change the LogFile variable, and run
# the daemon with --config-file option). That's why you shouldn't uncomment
# this option.
#LogFileUnlock

# Maximal size of the log file. Default is 1 Mb.
# Value of 0 disables the limit.
# You may use 'M' or 'm' for megabytes (1M = 1m = 1048576 bytes)
# and 'K' or 'k' for kilobytes (1K = 1k = 1024 bytes). To specify the size
# in bytes just don't use modifiers.
###LogFileMaxSize 2M
LogFileMaxSize 0

# Log time with an each message.
LogTime

# Log also clean files. May be useful in debugging but will drastically
# increase the log size.
#LogClean

# Use system logger (can work together with LogFile).
#LogSyslog

# Enable verbose logging.
#LogVerbose

# This option allows you to save the process identifier of the listening
# daemon (main thread).
PidFile /var/run/clamav/clamd.pid

# Optional path to the global temporary directory.
# Default is system specific - usually /var/tmp or /tmp.
#TemporaryDirectory /var/tmp

# Path to the database directory.
# Default is the hardcoded directory (mostly /usr/local/share/clamav,
# but it depends on installation options).
#DatabaseDirectory /var/lib/clamav

# The daemon works in local or network mode. Currently the local mode is
# recommended for security reasons.

# Path to the local socket. The daemon doesn't change the mode of the
# created file (portability reasons). You may want to create it in a directory
# which is only accessible for a user running daemon.
```

```
LocalSocket /tmp/clamd

# Remove stale socket after unclean shutdown.
#FixStaleSocket

# TCP port address.
#TCPSocket 3310

# TCP address.
# By default we bind to INADDR_ANY, probably not wise.
# Enable the following to provide some degree of protection
# from the outside world.
#TCPAddr 127.0.0.1

# Maximum length the queue of pending connections may grow to.
# Default is 15.
#MaxConnectionQueueLength 30

# When activated, input stream (see STREAM command) will be saved to disk before
# scanning - this allows scanning within archives.
#StreamSaveToDisk

# Close the connection if this limit is exceeded.
#StreamMaxLength 10M

# Maximal number of a threads running at the same time.
# Default is 5, and it should be sufficient for a typical workstation.
# You may need to increase threads number for a server machine.
MaxThreads 10

# Waiting for data from a client socket will timeout after this time (seconds).
# Default is 120. Value of 0 disables the timeout.
#ReadTimeout 300

# Maximal depth the directories are scanned at.
MaxDirectoryRecursion 15

# Follow a directory symlinks.
# SECURITY HINT: You should have enabled directory recursion limit to
# avoid potential problems.
#FollowDirectorySymlinks

# Follow regular file symlinks.
#FollowFileSymlinks

# Do internal checks (eg. check the integrity of the database structures)
# By default clamd checks itself every 3600 seconds (1 hour).
#SelfCheck 600

# Execute a command when a virus is found. In the command string %v will
# be replaced by the virus name.
#
#VirusEvent /usr/local/bin/send_sms 123456789 "VIRUS ALERT: %v"

# Run as selected user (clamd must be started by root).
# By default it doesn't drop privileges.
###User clamav
User qscand

# Initialize the supplementary group access (for all groups in /etc/group
# user is added in. clamd must be started by root).
#AllowSupplementaryGroups

# Don't fork into background. Useful in debugging.
#Foreground

# Enable debug messages in libclamav.
#Debug

##
## Document scanning
##
```

```
# This option enables scanning of Microsoft Office document macros.
ScanOLE2

##
## Mail support
##

# Uncomment this option if you are planning to scan mail files.
ScanMail

##
## Archive support
##


# Comment this line to disable scanning of the archives.
ScanArchive


# By default the built-in RAR unpacker is disabled by default because the code
# terribly leaks, however it's probably a good idea to enable it.
#ScanRAR


# Options below protect your system against Denial of Service attacks
# with archive bombs.

# Files in archives larger than this limit won't be scanned.
# Value of 0 disables the limit.
# WARNING: Due to the unrarlib implementation, whole files (one by one) in RAR
#               archives are decompressed to the memory. That's why never disable
#               this limit (but you may increase it of course!)
ArchiveMaxFileSize 10M

# Archives are scanned recursively - e.g. if Zip archive contains RAR file,
# the RAR file will be decompressed, too (but only if recursion limit is set
# at least to 1). With this option you may set the recursion level.
# Value of 0 disables the limit.
ArchiveMaxRecursion 5

# Number of files to be scanned within archive.
# Value of 0 disables the limit.
ArchiveMaxFiles 1000

# Mark potential archive bombs as viruses (0 disables the limit)
ArchiveMaxCompressionRatio 200

# Use slower decompression algorithm which uses less memory. This option
# affects bzip2 decompressor only.
#ArchiveLimitMemoryUsage

# Mark encrypted archives as viruses (Encrypted.Zip, Encrypted.RAR).
#ArchiveBlockEncrypted


##
## Clamuko settings
## WARNING: This is experimental software. It is very likely it will hang
##               up your system !!!
##

# Enable Clamuko. Dazuko (/dev/dazuko) must be configured and running.
#ClamukoScanOnAccess

# Set access mask for Clamuko.
ClamukoScanOnOpen
ClamukoScanOnClose
ClamukoScanOnExec

# Set the include paths (all files in them will be scanned). You can have
# multiple ClamukoIncludePath options, but each directory must be added
# in a seperate option. All subdirectories are scanned, too.
```

```
ClamukoIncludePath /home
#ClamukoIncludePath /students

# Set the exclude paths. All subdirectories are also excluded.
#ClamukoExcludePath /home/guru

# Limit the file size to be scanned (probably you don't want to scan your movie
# files ;))
# Value of 0 disables the limit. 1 Mb should be fine.
ClamukoMaxFileSize 1M

# Enable archive support. It uses the limits from clamd section.
# (This option doesn't depend on ScanArchive, you can have archive support
# in clamd disabled).
ClamukoScanArchive
```

ANEXO B2 – Arquivo de configuração do SpamAssassin /etc/mail/spamassassin/ local.cf

```
# These values can be overridden by editing ~/.spamassassin/user_prefs.cf
# (see spamassassin(1) for details)

# These should be safe assumptions and allow for simple visual sifting
# without risking lost emails.

required_hits 7
rewrite_subject 1
subject_tag [SPAM]

score SUBJ_ILLEGAL_CHARS 0
```

ANEXO B3 – Arquivo de configuração do Qmail-scanner /var/spool/qmailscan/ quarantine-attachments.txt

```
# Sample of well-known viruses that perlscan_scanner can use
#
# This is case-insensitive, and TAB-delimited.
#
# ******
# REMEMBER: run /var/qmail/bin/qmail-scanner-queue.pl -g after
# this file is modified
# ******
#
# Format: three columns
#
# filename<TAB>size (in bytes)<TAB>Description of virus/whatever
#
# OR:
#
# string<TAB>Header<TAB>Description of virus/whatever
#
# [this one allows you to match on (e.g.) Subject line.
#
# NOTE 1: This is the crudest "virus scanning" you can do - we are
# arbitrarily deciding that particular filenames of certain sizes contain
# viruses - when they may not. However this can be useful for the times
# when a new virus is discovered and your scanner cannot detect it (yet).
#
# NOTE 2: This is only good for picking up stand-alone viruses like the
# following. Macro viruses are impossible to detect with this method as
# they infect users docs.
#
# NOTE 3: Wildcards are supported. This system can also be used to deny
# Email containing "bad" extensions (e.g. .exe, .mp3, etc). No other
# wildcard type is supported. Be very careful with this feature. With
# wildcards, the size field is ignored (i.e. any size matches).
#
# .exe       0              Executable attachment too large
#
# That would ban .EXE files from your site (but would
# still allow .zip files...
#
# .mp3      0              MP3 attachments disallowed
#
# ...would stop any Email containing MP3 attachments passing.
#
# NOTE 4: No you can't use  this to ban any file (i.e. *.*) that's over
# a certain size  - you should
# "echo 10000000 > /var/qmail/control/databytes"
# to set the maximum SMTP message size to 10Mb.
#
# NOTE 5: The second option allows you to match on header. This would allow
# you to block Email viruses when you don't know anything else other than
# there's a wierd Subject line (or From line, or X-Spanska: header, ...).
# Note that it's a case-sensitive, REGEX string, and the system will
# automatically surround it with ^ and $ before matching. i.e. if you
# want wildcards, explicitly put them in...
#
# The string _must_be_ "Virus-" followed by the header you wish to match
# on - followed by a colon (:).
#
# e.g.
#
# Pickles.*Breakfast  Virus-Subject:          Fake Example Pickles virus
#
# will match "Subject: Pickles for Breakfast" - and
# not "Subject: Pickles - where did you go?"
```

```
#
#
# NOTE 6: Similar to the headers option, you can match on the mail ENVELOPE
# headers - i.e. "MAIL FROM:" and "RCPT TO:". These are identical to
# Virus-<header>, except that the header names are MAILFROM and RCPTTO only.
#
# e.g.
#
# bogus@address.here              Virus-MAILFROM:   Bad mail envelope not allowed here!
#
# NOTE 7: Another "faked" header - "Virus-TCPREMOTEIP" can be used to match
# actions against the IP address of the SMTP client.
#

EICAR.COM                       69          EICAR Test Virus
Happy99.exe                     10000       Happy99 Trojan
zipped_files.exe        120495  W32/ExploreZip.worm.pak virus
ILOVEYOU                        Virus-Subject:          Love Letter Virus/Trojan
message/partial.*               Virus-Content-Type:  Message/partial MIME attachments blocked by policy
#The following matches Date: headers that are over 100 chars in length
#these are impossible in the wild
.{100,}                         Virus-Date:                     MIME Header Buffer Overflow
.{100,}                         Virus-Mime-Version: MIME Header Buffer Overflow
.{100,}                         Virus-Resent-Date:    MIME Header Buffer Overflow
#
#Let's stop that nasty BadTrans virus from uploading your keystrokes...
ZVDOHYIK@yahoo.com|udtzqccc@yahoo.com|DTCELACB@yahoo.com|I1MCH2TH@yahoo.com|WPADJQ12@yahoo.co
m|smr@eurosport.com|bgnd2@canada.com|muwripa@fairesuivre.com|eccles@ballsy.net|S_Mentis@mail-x-
change.com|YJPFJTGZ@excite.com|JGQZCD@excite.com|XHZJ3@excite.com|OZUNYLRL@excite.com|tsnlqd@excite.com
|cxkawog@krovatka.net|ssdn@myrealbox.com          Virus-To:  BadTrans Trojan exploit!

#
# These are examples of prudent defaults to set for most sites.
# Commented out by default
.vbs    0       VBS files not allowed per Company security policy
#.lnk   0       LNK files not allowed per Company security policy
.scr    0       SCR files not allowed per Company security policy
.wsh    0       WSH files not allowed per Company security policy
.hta    0       HTA files not allowed per Company security policy
.pif    0       PIF files not allowed per Company security policy
.com    0       COM files not allowed per Company security policy


# ******
# REMEMBER: run /var/qmail/bin/qmail-scanner-queue.pl -g after
# this file is modified
# ******
#
# EOF
```

ANEXO B4 – Arquivo de configuração do Qmail-scanner /var/qmail/bin/qmail-scanner-queue.pl

```
#!/usr/bin/perl -T
#
# File: qmail-scanner-queue.pl
# Version: 1.22 - patched - st - 20040502
#
# Author: Jason L. Haar <jhaar@users.sourceforge.net>
#
# Patched by: Salvatore Toribio <toribio@pusc.it>
#
# See the file READMEpatched for information about the patch
# Includes the patch from Chris Hine to discard (st: or delete/reject) spam
#
# This file was auto-generated by:
#
# ./configure --qs-user qscand --qs-group qscand --spooldir /var/spool/qmailscan --
qmaildir /var/qmail --bindir /var/qmail/bin --qmail-queue-binary
/var/qmail/bin/qmail-queue  --admin root --domain alfa.linux.sibi.usp.br --admin-
fromname "" --notify psender,nmlvadm --local-domains alfa.linux.sibi.usp.br --silent-
viruses auto --block-password-protected 0 --lang en_GB --debug 0 --minidebug 1 --
unzip 0 --add-dscr-hdrs 0 --dscr-hdrs-text "X-Qmail-Scanner" --archive 0 --run-first-
p-s 0 --redundant no --log-details 0 --log-crypto 0 --fix-mime 2  --ignore-eol-check
0 --virus-to-delete 0 --sa-delta 0 --sa-subject "" --sa-quarantine 0 --sa-delete 0 --
sa-reject 0 --sa-alt 0 --sa-debug 0 --scanners "auto" --install 1
#
# Description: This is a replacement/add-on for Qmail 1.0.3's qmail-queue.
# It can call several blocking programs - such as virus scanners - on every
# SMTP-received Email message, checking for viruses and blocked filenames,
# only allowing the message to continue if it passes the tests.
#
#   Copyright (C) 1999,2000,2001 the people mentioned above
#
#   This program is free software; you can redistribute it and/or modify
#   it under the terms of the GNU General Public License as published by
#   the Free Software Foundation; either version 1, or (at your option)
#   any later version. See <URL:http://www.gnu.org/copyleft/gpl.html>
#   for a copy.
#
#   This program is distributed in the hope that it will be useful,
#   but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU General Public License for more details.
#
#   The software is provided as is. Please bear in mind that we have
#   done this in my spare time. While it is as accurate as we could
#   make it there is a reasonable chance that there are mistakes
#   somewhere in here. If you email me and tell me about them, I will
#   be happy to fix them but I can't take responsibility for your system.
#   Basically use this at your own risk.
#
#########################################################################

#########################################################################
##
## Required Packages
##
## Qmail-1.03
## Perl 5.005_03+
## Maildrop-0.73
## Bruce Guenter's QMAILQUEUE patch <URL:http://www.qmail.org/qmailqueue-patch>
## Perl module Time::HiRes and DB_File
##
##
## So-far tested Virus scanners:
##     Trend's Virus scanner for Linux
```

```
##     MacAfee's (NAI's) virus scanner for Linux
##     Sophos's virus scanner for Linux
##     H+BEDV's antivir scanner for Linux
##     F-Secure's fsav scanner for Linux
##     P-Prot for Linux
##     Sophie (daemonized Sophos scanner)
##     Trophie (daemonized Trend scanner)
##
#########################################################################

#########################################################################
##
## Site-specific config
##
#########################################################################


delete @ENV{qw(IFS CDPATH ENV BASH_ENV QMAILMFTFILE QMAILINJECT)};

use strict 'vars', 'subs';

#Set locale to "C" (English). That way any string checks on forked apps
#will tend to be in English - simplifying/standardizing regex matches
my $orig_locale=$ENV{'LC_ALL'};
$ENV{'LC_ALL'} = 'C';

use Sys::Syslog qw(:DEFAULT setlogsock);
setlogsock('unix');

my $VERSION="1.22st";

# Mail header to add to each scanned message to report stuff in...
# Default is to not generate them ($descriptive_hdrs = 0) - as  that
# info is also in the Received: headers...
###my $descriptive_hdrs=0;
my $descriptive_hdrs=1;
my $V_HEADER="X-Qmail-Scanner";
my($qsmsgid);
$qsmsgid=tolower("$V_HEADER-message-id");


# From: line  information used when making reports
###my $V_FROM='root@alfa.linux.sibi.usp.br';
my $V_FROM='root@sibi.usp.br';
my $V_FROMNAME="";

# Address carbon-copied on any virus reports
###my $QUARANTINE_CC='root@alfa.linux.sibi.usp.br';
my $QUARANTINE_CC='root@sibi.usp.br';

#Array of local domains that are checked against for
#deciding whether or not to send recipient alerts to
###my @local_domains_array=('alfa.linux.sibi.usp.br');
my @local_domains_array=('alfa.sibi.usp.br');

# Array of virus that we don't want to inform the sender of.
my
@silent_viruses_array=('klez','bugbear','hybris','yaha','braid','nimda','tanatos','so
big','winevar','palyh','fizzer','gibe','cailont','lovelorn','swen','dumaru','sober','
hawawi','hawaii','holar-
i','mimail','poffer','bagle','worm.galil','mydoom','worm.sco','tanx','novarg','@mm','
cissy','cissi','qizy','bugler','dloade','netsky','spam');

# st: Virus that will be deleted without notifying anyone,
# you can add other viruses in the form "virus1|virus2|virus3".
# Most of the viruses in the 'silent_viruses_array' could be
# added to this list safely.
# i.e. "mydoom|worm.sco|novarg|tanx|bagle|netsky|roca"
my $virus_to_delete="";

#Array of virus scanners used must point to subroutines
my @scanner_array=("clamdscan_scanner","spamassassin");
```

```
#Addresses that should be alerted of any quarantined Email
my $NOTIFY_ADDRS='psender,nmlvadm';

#Try to fix bad MIME messages before passing to MIME unpacker
my $BAD_MIME_CHECKS='2';

#Disable just the EOL char check instead of all of BAD_MIME_CHECKS
my $IGNORE_EOL_CHECK='0';

#Block password protected zip files
my $BLOCK_PASSWORD_PROTECTED_ARCHIVES='0';

# The full path to qmail programs we'll need.
my $qmailinject = '/var/qmail/bin/qmail-inject';
my $qmailqueue  = '/var/qmail/bin/qmail-queue';

# What directory to use for storing temporary files.
my $scandir = '/var/spool/qmailscan';

#What maildir folder to store working files in
my $wmaildir='working';

#What maildir folder to store failed messages in (for cronjob scan)
my $fmaildir='failed';

#What maildir folder to store quarantine in
my $vmaildir='quarantine';

#What maildir folder to archive received Email in instead of deleting
my $archiveit='0';
my $archivedir='archives';

#Name of file in $scandir where debugging output goes
my $debuglog="qmail-queue.log";

#Name of file where quarantine reports go (for long-term storage)
my $quarantinelog="quarantine.log";

#Generate nice random filename
my ($sysname, $hostname, $release, $version, $machine) = uname();
#my $hostname='alfa.linux.sibi.usp.br'; #could get via call I suppose...

#If you trust the virus scanners handling of mbox format itself
#you may want to let it have a go at the "raw" message, and original
#zip files if present
my $redundant_scanning='0';

#If you want to log via file/syslog information of all Email
# that passes through your system (from/to/subj/size/attachments)
###my $log_details="0";
my $log_details="syslog";

#If you'd like Q-S to report which messages are PGP or S/MIME,
#turn this on
my $log_crypto="0";

# st: Do you want to run first perl_scanner? (1/0)
my $run_first_p_s='0';

# st: If $spamc_subject is defined and fast_spamassassin mode is selected,
# a tag will be added to the subject indicating how the message is to
# be considered as spam, in this way:
# LOW: required_hits < score < required_hits + sa_delta
# MEDIUM: required_hits + sa_delta < score < required_hits + 2 * sa_delta
# HIGH: required_hits + 2 * sa_delta < score
# Be aware, 2*sa_delta must be lower than sa_quarantine.
# 'required_hits' is the value set in the SpamAssassin configuration file.
my $sa_delta='0';

# st: Spam messages with a score higher than
# (required_hits + sa_quarantine) should be quarantined.
# Only relevant if SpamAssassin is used.
# Score of 0 means deliver all messages. Defaults to 0.
###my $sa_quarantine='0';
```

```
my $sa_quarantine='1';

# st: Spam messages with a score higher than
# (required_hits + sa_delete) should be deleted (or rejected).
# Only relevant if SpamAssassin is used. Score of 0
# means deliver all messages. Defaults to 0.
# If sa-quarantine is set, sa-delete must be greater.
my $sa_delete='0';

# st: If you enable sa-reject and sa-delete is properly set,
# messages with a score higher than (required_hits + sa_delete)
# will be rejected before the smtp session is closed.
# Otherwise they are just dropped silently. (1/0)
my $sa_reject='0';

# st: Use the alternative subroutine for spamassassin, it runs in
# *fast_spamassassin* mode and doesn't pass the '-u' option
# to spamc, allows to log the spamassassin report. (1/0)
###my $sa_alt='0';
my $sa_alt='1';

# st: If sa_alt is enabled an you enable this option, you will
# have a beautiful log with the tests and the scores of
# spamassassin in the file qmail-queue.log (1/0)
###my $sa_debug='0';
my $sa_debug='1';

# st: Enable this option to do not pass to spamassassin messages
# from MAILER-DAEMON, see READMEpatched for details. (1/0)
my $SA_SKIP_MD='0';

#Full path to file in which virus-scanner versioning info is kept
my $versionfile="$scandir/qmail-scanner-queue-version.txt";

#DB file (without extension) where bad filenames are kept.
# You edit $db_filename.txt, and "qmail-scanner-queue.pl -g" generates
$db_filename.db
my $db_filename="$scandir/quarantine-attachments";

#What locale is used on this system
#$sys_locale="LOCALE";

#Full paths to binaries used within this script follow - small performance
#improvement :-)


my $mimeunpacker_binary='/usr/local/bin/reformime ';
my $unzip_binary='/usr/bin/unzip';
my $unzip_options='-Pxx1278424852xx';
my $tnef_binary='/usr/local/bin/tnef';
my $rm_binary='/bin/rm';
my $grep_binary='/bin/grep';
my $find_binary='/usr/bin/find';
my $uudecode_binary='';
my $uudecode_pipe='';


my $uvscan_binary='';
my $csav_binary='';
my $sweep_binary='';
my $sophie_binary='';
my $trophie_binary='';
my $iscan_binary='';
my $hbedv_binary='';
my $hbedv_options='';
my $avp_binary='';
my $avpdaemon_binary='';
my $fprot_binary='';
my $fsecure_binary='';
my $inocucmd_binary='';
my $ravlin_binary='';
my $vexira_binary='';
my $clamscan_binary='/usr/local/bin/clamscan';
```

```
my $clamscan_options="-r -m --unzip --unrar --unzoo --lha --disable-summary --max-recursion=10 --max-space=100000";
my $clamdscan_binary='/usr/local/bin/clamdscan';
my $clamdscan_options="-r --disable-summary --max-recursion=10 --max-space=100000";
my $spamc_binary='/usr/bin/spamc';
# st: fast_spamassassin options=" -c -f " / verbose_spamassassion options=" -c "
my $spamc_options=' -c -f';
###my $spamc_subject="";  # st: if fast_spamassassin mode is selected
my $spamc_subject="[SPAM] ";  # st: if fast_spamassassin mode is selected
my $spamassassin_binary='/usr/bin/spamassassin';

# st: If somebody is using spamassassin with unix socket...
my $spamc_socket='';
$spamc_binary.=" -U $spamc_socket" if ($spamc_socket ne "");

my ($sa_comment,$sa_level);
my $sa_symbol='+';
my ($tag_score)="";
my $SNEAKY_WINDOWS_EXTENSIONS="exe|com|pps|w[pm][szd]|vcf|nws|cmd|bat|pif|sc[rt]|dll|ocx|do[ct]|xl[swt]|p[po]t|rtf|vb[se]?|hta|p[lm]|sh[bs]|hlp|chm|eml|ws[cfh]|ad[ep]|jse?|md[abew]|ms[ip]|reg|as[dfx]|cil";
my $VALID_WINDOWS_EXTENSIONS="sav|htm|html|pst|ost|txt|gif|jpeg|mpeg|jpg|png|mny|wav|tif|$SNEAKY_WINDOWS_EXTENSIONS";

$ENV{'PATH'}='/bin:/usr/bin';

my $SCANINFO='';

my $MAX_FILE_LENGTH=100;
my $MAX_NUM_HDRS=140;
my $QE_LEN=20;

#Maximum amount of time we allow Q-S to run before returning
# a temp failure. This is so remote SMTP servers don't get confused
# over whether or not they have delivered to a SMTP server
# that's refused to say "OK" for over an hour...
# We'll default to 20 minutes. If the scanner loop takes more than 20
# minutes to scan the message, then something *must* be wrong with the
# scanner.
my $MAXTIME=20*60;

#Finally, are you sure your virus scanners can unpack zip files?
#McAfee's doesn't!
my $force_unzip=0;

#Descriptive string to use in generated Email
my $destring="virus";

########################################################################
##
## End of site-specific settings
##
########################################################################



#Want debugging? Enable this and read $scandir/qmail-queue.log
my $DEBUG='0';

# st: Minimal debug only works if $DEBUG=0
my $MINIDEBUG='1';

my @uufile_list = ();
my @attachment_list = ();
my @zipfile_list = ();

#Want microsec times for debugging
use Time::HiRes qw( usleep ualarm gettimeofday tv_interval );
use POSIX;

use vars qw/ $opt_v $opt_h $opt_g $opt_r $opt_z/;
```

```
use Getopt::Std;
#my ($opt_v,$opt_h,$opt_g,$opt_r,$opt_z);

getopts('vhgrz');

my ($start_time,$last_time);
$start_time = $last_time = [gettimeofday];

(my $prog=$0) =~ s/^.*\///g;

if ( $opt_h ) {
  print "

$prog

    -h - This help
    -v - show details about this install.
         Please include in any bug reports.
    -z - gather virus scanner/DAT versions
         and cleanup old temp files
    -g - generate perlscanner database
    -r - read from perlscanner database\n";
  exit;
}


if ( $opt_g || $opt_r) {
  &generate_quarantine_db;
    exit 0;
}
if ( $opt_v ) {
  &show_version;
  exit 0;
}
chdir($scandir);
umask(0077);

if (! -d "$scandir/tmp") {
  mkdir("$scandir/tmp",0700) || &error_condition("cannot create $scandir/tmp - $!");
}

my ($quarantine_event,$quarantine_event_tmp)=0;

my $file_id = &uniq_id();

#For security reasons, tighten the follow vars...
$ENV{'SHELL'} = '/bin/sh' if exists $ENV{SHELL};
$ENV{'TMP'} = $ENV{'TMPDIR'} = "$scandir/tmp/$file_id";
#$ENV{'QMAILSUSER'} = $ENV{'QMAILSHOST'} = '';



if ($mimeunpacker_binary =~ /reformime/) {
  $mimeunpacker_binary .= " -x$ENV{'TMPDIR'}/";
} elsif ($mimeunpacker_binary =~ /ripmime/) {
   $mimeunpacker_binary .= " --unique_names -i - -d $ENV{'TMPDIR'}/";
}


#Get current timestamp for logs
my ($sec,$min,$hour,$mday,$mon,$year,$nowtime);
($sec,$min,$hour,$mday,$mon,$year) = localtime(time);
# st: if you want to change the date format, uncoment/coment the apropiate
# next lines and also in the subroutine 'close_log' and the end of the file
# before the scanners routines. BE CAREFUL...
# Format: dd/mm/yyyy hh:mm:ss....
#my $nowtime = sprintf "%02d/%02d/%02d %02d:%02d:%02d", $mday, $mon+1, $year+1900,
$hour, $min, $sec;
# Format: day, dd month yyyy hh:mm:ss +TZ...
#my $nowtime = strftime("%a, %d %b %Y %H:%M:%S %Z", localtime(time));
my ($smtp_sender,$remote_smtp_ip,$uid);
```

```
$uid=$<;

# st: I need the process number, and other variables
my $nprocess=$$;
my $sa_report='';
my ($sa_score,$required_hits)=('0','0');
# st: Flag to delete message
my $del_message='0';

if ($DEBUG || $MINIDEBUG ) {
  open(LOG,">>$scandir/$debuglog");
  select(LOG);$|=1;
  &debug("+++ starting debugging for process $$ by uid=$uid");
  &minidebug("+++ starting debugging for process $$ by uid=$uid");
}

# st: if the variable SA_ONLYDELETE_HOST is set in the tcpserver
# don't reject messages coming from those IPs, just delete them
# You should set this variable for your secondary mail server.
if (defined($ENV{'SA_ONLYDELETE_HOST'}) || defined($ENV{'SA_WHITELIST'})) {
  $sa_reject="0";
  &debug("WL: The server is a SA_ONLYDELETE_HOST, don't reject");
  &minidebug("WL: The server is a SA_ONLYDELETE_HOST, don't reject");
}

# st: if the variable BMC_WHITELIST is set in the tcpserver
# don't search for 'bad mime characters' in the headers of messages
# coming from those IPs.
# It would be hard to mantain this whitelist...
if (defined($ENV{'BMC_WHITELIST'})) {
  $BAD_MIME_CHECKS='0';
  &debug("WL: The server is in the BMC_WHITELIST, don't check BMC");
  &minidebug("WL: The server is in the BMC_WHITELIST, don't check BMC");
}

&debug("setting UID to EUID so subprocesses can access files generated by this
script");
$< = $>;            # set real to effective uid
#$( = $);            # set real to effective gid

&debug("program name is $prog, version $VERSION");
if ($opt_z) {
  &scan_queue;
  exit 0;
}


&scanner_info;

my ($smtp_sender,$remote_smtp_ip);


if ($ENV{'TCPREMOTEIP'}) {
  $smtp_sender="via SMTP from $ENV{'TCPREMOTEIP'}";
  $remote_smtp_ip=$ENV{'TCPREMOTEIP'};
  # st: do not reject mails from localhost useful for fetchmail
  $sa_reject="0" if ($remote_smtp_ip eq "127.0.0.1");
  $tag_score.="RC:1($remote_smtp_ip):" if (defined($ENV{'RELAYCLIENT'}));
  &debug("incoming SMTP connection from $smtp_sender");
} else {
  $smtp_sender="via local process $$";
  $remote_smtp_ip='127.0.0.1';
  $tag_score.="RC:1($remote_smtp_ip):"; #Always would be relayed
  # st: do not reject mails from localhost useful for fetchmail
  $sa_reject="0";
  &debug("incoming pipe connection from $smtp_sender");
}
$tag_score.="RC:0($remote_smtp_ip):" if ($tag_score !~ /RC:1/);

my (%headers );
my ($CRYPTO_TYPE,$altered_subject, $HEADERS, $env_returnpath, $returnpath);
```

```
my ($ATTACHMENT,
%BOUNDARY,$BOUNDARY_REGEX,$attachment_header,$attachment_value,%attach_hdrs,%content_
type);
my ($ct_attachment_filename,$cd_attachment_filename);
my ($env_recips, $recips, $trecips, $recip, $one_recip);
my ($alarm_status,$elapsed_time,$msg_size,$file_desc);
my ($description,$quarantine_description,$illegal_mime);
my $skip_text_msgs=1;
my $plain_text_msg=0;
my $contains_rfc822=0;
my $xstatus=0;
my $attachment_counter=0;

&working_copy;

  # st: working_copy is quite heavy, let see the elapsed time from start
  my $elapsed_1=tv_interval ($start_time, [gettimeofday]);
  &minidebug("w_c: elapsed time from start $elapsed_1 secs");

#Now alarm this area so that hung networks/virus scanners don't cause
#double-delivery...

eval {
  $SIG{ALRM} = sub { die "Maximum time exceeded. Something cannot handle this
message." };
  alarm $MAXTIME;

  &deconstruct_msg; #JLH if (!$quarantine_event);


  #Now unset env var QMAILQUEUE so any further Email's sent don't
  #go through the Qmail-Scanner again
  &debug("unsetting QMAILQUEUE env var");
  delete $ENV{'QMAILQUEUE'};

  #This SMTP session is incomplete until we see dem envelope headers!
  &grab_envelope_hdrs;
  &debug("from=$headers{'from'},subj=$headers{'subject'}, $qsmsgid=$headers{$qsmsgid}
$smtp_sender");
  &minidebug("from='$headers{'from'}', subj='$headers{'subject'}', $smtp_sender");

  #Add envelope details to headers array so that they can be matched within
  #perlscanner.
  #Note how they're uppercase cf the message headers which are all forced
  #lowercased. This is to ensure no-one can override them...

  $headers{'MAILFROM'}=$returnpath;
  $headers{'RCPTTO'}=$recips;
  $headers{'TCPREMOTEIP'}=$remote_smtp_ip;

  if ( ($BAD_MIME_CHECKS > 1 && $headers{'mime-version'} eq "") || ($headers{'mime-
version'} ne "" && $headers{'content-type'} =~ /^text\/plain/i)) {
    #Hmm, doesn't look nice, but it feels better to make this a separate check for
some reason
    if ($skip_text_msgs && ($contains_rfc822 < 2) && !@uufile_list &&
!@attachment_list) {
      &debug("This is a PLAIN text message (because it's either not mime, or is
text/plain), skip virus scanners - but not SA");
      &minidebug("This is a PLAIN text message, skip virus scanners - but not SA");
      $plain_text_msg=1;
    }
  }

  #Now, start the scanners!
  #if (!$quarantine_event) {
    &init_scanners;
  #}

  # st: if the message is marked to delete skip the mailing routines
  if (!$del_message) {
    if ($quarantine_event) {
      &debug("unsetting TCPREMOTEIP env var");
      delete $ENV{'TCPREMOTEIP'};
      #Reset locale back to original
```

```
      $ENV{'LC_ALL'}=$orig_locale;
      &email_quarantine_report;
    } else {
      &qmail_parent_check;
      &qmail_requeue($env_returnpath,$env_recips,"$scandir/$wmaildir/new/$file_id");
    }
  }
  alarm 0;
};

$alarm_status=$@;
if ($alarm_status and $alarm_status ne "" ) {
  if ($alarm_status eq "Maximum time exceeded. Something cannot handle this
message.") {
    &error_condition("ALARM: taking longer than $MAXTIME secs. Requeuing...");
  } else {
    &error_condition("Requeuing: $alarm_status");
  }
}


#Msg has been delivered now, so don't want hangs in this part
#to affect delivery

if ($log_details) {
  $tag_score .= "$CRYPTO_TYPE:" if ($log_crypto && $CRYPTO_TYPE ne "");
  $tag_score=":$tag_score" if ($tag_score ne "");
  if ($trecips =~ /\0T/) {
    for $recip (split(/\0T/,$trecips)) {
      &log_msg("qmail-scanner",($quarantine_event ne "0" ?
"$quarantine_event$tag_score" :
"Clear$tag_score"),$elapsed_time,$msg_size,$returnpath,$recip,$headers{'subject'},$he
aders{$qsmsgid},$file_desc) if ($recip ne "");
    }
  } else {
    #Only one recip
    &log_msg("qmail-scanner",($quarantine_event ne "0" ?
"$quarantine_event$tag_score" :
"Clear$tag_score"),$elapsed_time,$msg_size,$returnpath,$recips,$headers{'subject'},$h
eaders{$qsmsgid},$file_desc);
  }
}
&cleanup;

# st: write to the log the end of the process
&close_log;
exit 0;

##############################################################################
# Error handling
##############################################################################

#Generate uniq identifiers
sub uniq_id {
  return "$hostname" . time .  __LINE__ . $$;
}


# Fail with the given message and a temporary failure code.
sub error_condition {
  my ($string,$errcode)=@_;
  $errcode=111 if (!$errcode);
  eval {
    syslog('mail|err',"$V_HEADER-$VERSION:[$file_id] $string");
  };
  if ($@) {
    setlogsock('inet');
    syslog('mail|err',"$V_HEADER-$VERSION:[$file_id] $string");
  }
  if ($log_details ne "syslog") {
    warn "$V_HEADER-$VERSION:[$file_id] $string\n";
  }
  #$nowtime = sprintf "%02d/%02d/%02d %02d:%02d:%02d", $mday, $mon+1, $year+1900,
$hour, $min, $sec;
```

```
  &debug("error_condition: $V_HEADER-$VERSION: $string");
  &minidebug("error_condition: $V_HEADER-$VERSION: $string");
  &cleanup;
  &close_log;
  exit $errcode;
}

sub debug {
  my $dnowtime = strftime("%a, %d %b %Y %H:%M:%S %Z", localtime(time));
  print LOG "$dnowtime:$nprocess: ",@_,"\n" if ($DEBUG);
}

sub working_copy {
  my
($hdr,$last_hdr,$value,$num_of_headers,$last_header,$last_value,$attachment_filename)
;
  select(STDIN); $|=1;

  &debug("w_c: mkdir $ENV{'TMPDIR'}");
  mkdir("$ENV{'TMPDIR'}",0700)||&error_condition("$ENV{'TMPDIR'} exists - try again
later...");
  chdir("$ENV{'TMPDIR'}")||&error_condition("cannot chdir to $ENV{'TMPDIR'}/");
  if (-f "$scandir/$wmaildir/tmp/$file_id" || -f "$scandir/$wmaildir/new/$file_id") {
    &error_condition("$file_id exists, try again later");
  }
  &debug("w_c: start dumping incoming msg into $scandir/$wmaildir/tmp/$file_id
[",&deltatime,"]");
  open(TMPFILE,">$scandir/$wmaildir/tmp/$file_id")||&error_condition("cannot write to
$scandir/$wmaildir/tmp/$file_id - $!");

  my $still_headers=1;
  my $begin_content='';
  my $still_attachment='';
  while (<STDIN>) {
    if ($still_headers) {
      $HEADERS .= $_;
      #Catch any naughty illegal header chars here
      if ($BAD_MIME_CHECKS && !$IGNORE_EOL_CHECK && /\r|\0/) {
       $illegal_mime=1;
       &debug("w_c: found CRL/NULL in header - invalid if this is a MIME message");
       &minidebug("w_c: found CRL/NULL in header - invalid if this is a MIME
message");
      }
      #Put headers into array
      if (/^\s+(.*)$/ && $last_hdr) {
       #Hmmm, a continuation...
       $headers{$last_hdr} .= $1 if (!$illegal_mime);
      } elsif (/^([^\s]+)/) {
       #This means it's not a continuation header
       if (!$quarantine_event && $BAD_MIME_CHECKS && ($headers{'mime-version'} ne "")
&& !/^([^\s]+):(.*)$/) {
         #Wow - a header (not header+value) that goes onto another line - not likely!
         $illegal_mime=1;
         $destring='problem';
         $quarantine_description="Disallowed breakage found in header name -
potential virus";
         $quarantine_event="Policy:Bad_MIME_Break";
         $description .= "\n---perlscanner results ---\n$destring
'$quarantine_description' found in message";
         &debug("w_c: disallowed breakage found in header name ($_) - potential
virus");
         &minidebug("w_c: disallowed breakage found in header name ($_) - potential
virus");
         #next;
       } else {
         /^([^\s]+):(.*)$/;
         $hdr=$1;
         $last_hdr=tolower($hdr);
         $value=$2;
         $value =~ s/^\s//;
         if (!$quarantine_event && $BAD_MIME_CHECKS && $hdr =~ /^[^X].*\(/i) {
           #Wow - a comment *inside* a standard header name. Only viruses are known
to do that
           #Should we test for [^0-9a-z\_\-\=\+] instead?
```

```
            $illegal_mime=1;
            $destring='problem';
            $quarantine_description='Disallowed MIME comment found in header name -
730 potential virus';
            $quarantine_event="Policy:Bad_MIME_Comment";
            $description .= "\n---perlscanner results ---\n$destring
'$quarantine_description' found in message";
            &debug("w_c: $quarantine_description");
735         &minidebug("w_c: $quarantine_description");
          }
          $num_of_headers++;
        }
        #Don't let this array grow without bounds...
740     if ($num_of_headers < $MAX_NUM_HDRS) {
          if ($hdr =~ /^to|cc/i && $headers{tolower($hdr)}) {
            #Special-case the To: and Cc: headers.
            #Broken mailers generate messages with multiple
            #instances of these, so merge them into one...
745         $headers{tolower($hdr)} .= ",$value";
          } elsif ($hdr =~ /^(from|x-mail|User-Agent|Organi|Received|Message-
ID|Subject)/i && $headers{tolower($hdr)}) {
            #Make sure any multiples of these headers are remembered, so that
            #perlscanner checks can see all instances - just wrap em up
750         #into one long line
            $headers{tolower($hdr)} .= " $value";
          } elsif (!$quarantine_event && $BAD_MIME_CHECKS > 1 && (($headers{'mime-
version'} ne "" && tolower($hdr) eq "mime-version") || ($headers{'content-type'} ne
"" && tolower($hdr) eq "content-type") || ($headers{'content-transfer-encoding'} ne
755 "" && tolower($hdr) eq "content-transfer-encoding") || ($headers{'content-
disposition'} ne "" && tolower($hdr) eq "content-disposition"))) {
            #Why would a legit message have important MIME headers defined >1 time? It
could imply someone is trying to sneak
            #something past SMTP scanners...
760         #To much parsing needs to be done to do this correctly - stuff 'em - break
the sucker ;-/
            &debug("Duplicate MIME headers found [$hdr] - renaming");
            print TMPFILE "$V_HEADER-$VERSION: renamed duplicate MIME headers\n";
            $_="$V_HEADER-Renamed-$_";
765       } else {
            #All other headers: the last occurance wins!
            $headers{tolower($hdr)}=$value;
          }
        }
770   }
      if (/^(\r|\r\n|\n)$/) {
       #headers have finished
       $still_headers=0;
       #Try to workaround those nasty broken viruses that produce Content-Type
775 without MIME-Version
       #to get around virus scanners
       if ($headers{'mime-version'} eq "") {
         #Make sure it's a MIME-style Content-type, Sun used to use Content-type for
other purposes...
780      if ($BAD_MIME_CHECKS && $headers{'content-type'} =~ /\//) {
           print TMPFILE "$V_HEADER-$VERSION: added fake MIME-Version header\nMIME-
Version: 1.0\n";
           $headers{'mime-version'}="1.0";
           &debug("w_c: added fake MIME-Version header");
785      }
       } elsif ($BAD_MIME_CHECKS > 1 && $headers{'content-type'} eq "") {
         #OK, now do the same for Content-Type. RFCs state "if no Content-Type
present, then it's text/plain"
         #However, Outlook chooses to read the entire message and "figures out" it's
790 mixed/multipart, etc.
         #This'll break that - as it should.
         #I wonder if I shouldn't just block these instead, the only ones I've seen
are either viruses or spam...
         print TMPFILE "$V_HEADER-$VERSION: added fake Content-Type header\nContent-
795 Type: text/plain\n";
         $headers{'content-type'}="text/plain";
         &debug("w_c: added fake Content-Type header");
       }
       if ( $headers{'content-type'} =~ /\// ) {
```

```
800          if ( $headers{'content-type'} =~
/^(\s+|)([^\/\s\(]+)(\s+|)\/(\s+|)([^\/\s\(\;]+)/ ) {
               $content_type{$attachment_counter}="$2/$5";
               &debug("w_c: primary Content-Type of $content_type{$attachment_counter}
found");
805            if ($log_crypto) {
                 if ($content_type{$attachment_counter} =~ /multipart\/signed/i) {
                  $CRYPTO_TYPE="CR:SMIME(signed)" if ($headers{'content-type'} =~
/protocol=\"application\/(x\-|)pkcs/i);
                  $CRYPTO_TYPE="CR:PGP(signed)" if ($headers{'content-type'} =~
810 /protocol=\"application\/(x\-|)pgp/i);
                  &debug("found MIME-based crypto ($CRYPTO_TYPE)");
                  &minidebug("found MIME-based crypto ($CRYPTO_TYPE)");
                 } elsif ($content_type{$attachment_counter} =~ /multipart\/encrypted/i)
{
815                $CRYPTO_TYPE="CR:PGP(encrypted)" if ($headers{'content-type'} =~
/protocol=\"application\/(x\-|)pgp/i);
                  &debug("found MIME-based crypto ($CRYPTO_TYPE)");
                  &minidebug("found MIME-based crypto ($CRYPTO_TYPE)");
                 }elsif ($content_type{$attachment_counter} =~ /application\/(x\-
820 |)pkcs7/i) {
                  $CRYPTO_TYPE="CR:SMIME(encrypted)" if ($headers{'content-type'} =~
/application\/(x\-|)pkcs7/i);
                  &debug("found MIME-based crypto ($CRYPTO_TYPE)");
                  &minidebug("found MIME-based crypto ($CRYPTO_TYPE)");
825              }
               }
             } else {
               $destring="problem";
               $illegal_mime=1;
830            $quarantine_description="Disallowed MIME Content-Type found - potential
virus";
               $quarantine_event="Policy:Bad_MIME_Type";
               $description .= "\n---perlscanner results ---\n$destring
'$quarantine_description' found in message";
835          }
           }
           #if ( $headers{'content-type'} =~
/boundary(\s*)=(|\s+|\s*\")([^\"\;]+)($|\;|\")/i) {
           if ( $headers{'content-type'} =~
840 /boundary(\s*)=(|\s+|\s*\")([^\s\"\;]+)($|\;|\")/i) {
             $BOUNDARY{$attachment_counter}=$3;
             if (!$quarantine_event && $BAD_MIME_CHECKS > 1 &&
($BOUNDARY{$attachment_counter} =~ /\"|\;/ || $BOUNDARY{$attachment_counter} eq ""))
{
845            &debug("w_c: RFC2046 says boundaries ($BOUNDARY{$attachment_counter})
can't contain such chars [see bcharsnospace]");
               #$destring="problem";
               #$illegal_mime=1;
               #$quarantine_description="Disallowed MIME boundary found - potential
850 virus";
               #$quarantine_event="Policy:Bad_MIME_Boundary";
               #$description .= "\n---perlscanner results ---\n$destring
'$quarantine_description' found in message";
             }
855          #Strip off stuff after semicolon, and escape any odd chars
             $BOUNDARY{$attachment_counter} =~ s/(\"|\;).*$//g;
             #$BOUNDARY{$attachment_counter} =~ s/([^a-z0-9=\_])/\\\1/gi;
             $BOUNDARY{$attachment_counter} =~ s/(\W)/\\$1/g;
             if (!$quarantine_event && $BAD_MIME_CHECKS > 1 &&
860 length($BOUNDARY{$attachment_counter}) > 250) {
               #RFC2046 says boundarys are 0-70 chars
               $destring="problem";
               $illegal_mime=1;
               $quarantine_description="Disallowed MIME boundary length found
865 (".length($BOUNDARY{$attachment_counter}).") - potential virus";
               $quarantine_event="Policy:Bad_MIME_Length";
               $description .= "\n---perlscanner results ---\n$destring
'$quarantine_description' found in message";
             }
870          $BOUNDARY_REGEX=$BOUNDARY{$attachment_counter};
             &debug("w_c: found a top-level boundary definition of
$BOUNDARY{$attachment_counter}");
           }
```

```
        if ( $headers{'content-type'} =~ /name(|\s+)=(|\s+|\s*\")([^\s\"].*)/i) {
          $ATTACHMENT=$3;
          $attachment_counter++;
          #Strip off stuff after semicolon
          $ATTACHMENT =~ s/(\"|\;).*$//g;
          &debug("w_c: found a top-level file attachment definition of $ATTACHMENT");
          push(@attachment_list, $ATTACHMENT);
        }
        if ($headers{'message-id'} eq "" && !$headers{$qsmsgid}) {
          $headers{$qsmsgid}="<".time . __LINE__ . $$ . "\@$hostname>";
          print TMPFILE "$V_HEADER-Message-ID: $headers{$qsmsgid}\n";
        } else {
          if (!$headers{$qsmsgid}) {
            $headers{$qsmsgid}=$headers{'message-id'};
          }
        }
      }
    }
    if (/^(\r|\r\n|\n)$/) {
      #&debug("w_c: attachment num=$attachment_counter");
      #&debug("w_c: last attachment header: $attachment_header:$attachment_value");
      $attach_hdrs{tolower($attachment_header)}=$attachment_value;
      if ($still_attachment ne "") {
       $still_attachment='';
       $begin_content=$attach_hdrs{'content-transfer-encoding'};
      } else {
       $begin_content='';
      }
      $attachment_header=$attachment_value='';
      #Let's see what the last MIME attachment contained
      if ($cd_attachment_filename ne "" && $ct_attachment_filename ne "" &&
$ct_attachment_filename ne $cd_attachment_filename) {
       if (!$quarantine_event && $BAD_MIME_CHECKS > 1) {
         &debug("w_c: Disallowed MIME filename manipulation - potential virus");
         &minidebug("w_c: Disallowed MIME filename manipulation - potential virus");
         $illegal_mime=1;
         $destring="problem";
         $quarantine_description='Disallowed MIME filename manipulation - potential
virus';
         $quarantine_event="Policy:Bad_MIME_Manipulation";
         $description .= "\n---perlscanner results ---\n$destring
'$quarantine_description' found in message attachment: \"$ct_attachment_filename\" !=
\"$cd_attachment_filename\"";
       }
      }
      #$ct_attachment_filename=$cd_attachment_filename='';
      if ($attach_hdrs{'content-type'} =~ /name(|\s+)=(|\s+|\s*\")([^\s\"].*)/i &&
$ATTACHMENT eq "") {
       $ATTACHMENT=$3;
       #Strip off stuff after semicolon
       $ATTACHMENT =~ s/(\"|\;).*$//g;
       $ATTACHMENT=tolower($ATTACHMENT);
       if (!grep(/^\Q$ATTACHMENT\E$/,@attachment_list)) {
         &debug("found C-T attachment filename $ATTACHMENT");
         push(@attachment_list, $ATTACHMENT);
       }
       $ct_attachment_filename=$ATTACHMENT;
       $ATTACHMENT='';
       #&debug("w_c: found a Content-Type attachment filename of
\"$ct_attachment_filename\"");
      }
      if ($attach_hdrs{'content-disposition'} =~
/name(|\s+)=(|\s+|\s*\")([^\s\"].*)/i && $ATTACHMENT eq "") {
       $ATTACHMENT=$3;
       #Strip off stuff after semicolon
       $ATTACHMENT =~ s/(\"|\;).*$//g;
       $ATTACHMENT=tolower($ATTACHMENT);
       if (!grep(/^\Q$ATTACHMENT\E$/,@attachment_list)) {
         push(@attachment_list, $ATTACHMENT);
         &debug("found C-D attachment filename $ATTACHMENT");
       }
       $cd_attachment_filename=$ATTACHMENT;
       $ATTACHMENT='';
```

```
       #&debug("w_c: found a Content-Disposition attachment filename of
\"$cd_attachment_filename\"");
      }
      if ($attach_hdrs{'content-type'} =~ /boundary(|\s+)=(|\s+|\s*\")([^\s\"].*)/i)
{
       $BOUNDARY{$attachment_counter}=$3;
       #Strip off delimiters around boundary
       $BOUNDARY{$attachment_counter} =~ s/(\"|\;).*$//g;
       $BOUNDARY{$attachment_counter} =~ s/(\W)/\\$1/g;
       if (!$quarantine_event && $BAD_MIME_CHECKS > 1 &&
length($BOUNDARY{$attachment_counter}) > 250) {
         #RFC2046 says boundarys are 0-70 chars
         $destring="problem";
         $illegal_mime=1;
         $quarantine_description="Disallowed MIME boundary length found
(".length($BOUNDARY{$attachment_counter}).") - potential virus";
         $quarantine_event="Policy:Bad_MIME_Boundary";
         $description .= "\n---perlscanner results ---\n$destring
'$quarantine_description' found in message";
       }
       if ( !$quarantine_event && $BAD_MIME_CHECKS > 1 &&
$BOUNDARY{$attachment_counter} =~ /^($BOUNDARY_REGEX)$/i) {
         &debug("w_c: hmm, a new boundary defintion that has already being set.
Sounds like a trojan");
         &minidebug("w_c: hmm, a new boundary defintion that has already being set.
Sounds like a trojan");
          &debug("w_c: broken attachment MIME details - block it!");
          &minidebug("w_c: broken attachment MIME details - block it!");
          $illegal_mime=1;
         $destring="problem";
          $quarantine_description='Disallowed MIME boundary found in attachment -
potential virus';
         $quarantine_event="Policy:Bad_MIME_Boundary";
         $description .= "\n---perlscanner results ---\n$destring
'$quarantine_description' found in message";
        }
       if ($BOUNDARY_REGEX ne "") {
         $BOUNDARY_REGEX.="|".$BOUNDARY{$attachment_counter};
       } else {
         $BOUNDARY_REGEX=$BOUNDARY{$attachment_counter};
       }
       #&debug("w_c: BOUNDARY_REGEX=$BOUNDARY_REGEX");
      }
      if ($attach_hdrs{'content-type'} =~ /\//) {
       $attachment_filename='';
       $attachment_filename=$cd_attachment_filename ne "" ? $cd_attachment_filename :
$ct_attachment_filename;
       #&debug("w_c: just parsed attachment $attach_hdrs{'content-type'}:
filename=$attachment_filename");
       if ( $attach_hdrs{'content-type'} =~
/^(\s+|)([^\/\s\(]+)(\s+|)\/(\s+|)([^\/\s\(\;]+)/ ) {
         $content_type{$attachment_counter}="$2/$5";
         &debug("w_c: attachment  $attachment_counter: Content-Type of
$content_type{$attachment_counter} found");
         if ($attachment_filename =~ /\.(scr|pif|vbs|exe)$/i &&
$content_type{$attachment_counter} !~ /^(message|text|application)/i) {
           $quarantine_description="Disallowed file ($attachment_filename) assosiated
with unrelated MIME type ($content_type{$attachment_counter}) - potential virus";
           &debug("w_c: $quarantine_description");
           &minidebug("w_c: $quarantine_description");
           $illegal_mime=1;
           $destring='problem';
           $quarantine_event="Policy:Forged_Attachment";
           $description .= "\n---perlscanner results ---\n$destring
'$quarantine_description' found in attachment $attachment_filename";
         }
       }
       $attach_hdrs{'content-type'}='';
       $ct_attachment_filename=$cd_attachment_filename='';
      }
    } else {
      #&debug("line=$_");
    }
```

```
    if ($still_attachment ne "") {
      #&debug("w_c: check those attachment headers ($_)");
      if (/^([^\s]+):(|\s+)(.*)$/) {
       $last_header=$attachment_header;
       $last_value=$attachment_value;
       $attachment_header=$1;
       $attachment_value=$3;
       $attachment_value =~ s/^\s+//;
       if ($last_header) {
         #&debug("w_c: $last_header:$last_value");
         $attach_hdrs{tolower($last_header)}=$last_value;
       }
       #&debug("w_c: beginning of $attachment_header, value=$attachment_value");
      } elsif (/^\s(.+)/) {
       #&debug("w_c: line :$_: reached");
       $attachment_value.=$1;
      } elsif (/^(\r|\r\n|\n|\s+)$/) {
       #Yeah - I should block spaces, but too many valid lists send out such junk...
       $still_attachment='';
      } else {
       #This will catch headers that are *correctly* broken over two lines.
       #No known mailer does that, but virus writers do, so we block it.
       #Note that a lot of mailing-lists (and AV systems...) shove their trailers
       #on the bottom of messages irrespective of whether they are MIME or not - so
       #we must allow such "hacks" to slip through
       if (!$quarantine_event && $BAD_MIME_CHECKS > 1 && ($BOUNDARY_REGEX ne "" &&
$still_attachment !~ /^\-\-($BOUNDARY_REGEX)\-\-$/) ) {
         &debug("w_c: broken attachment MIME details
(still_attachment=$still_attachment, but BOUNDARY_REGEX=\"$BOUNDARY_REGEX\")- block
it!");
         &minidebug("w_c: broken attachment MIME details
(still_attachment=$still_attachment, but BOUNDARY_REGEX=\"$BOUNDARY_REGEX\")- block
it!");
         $illegal_mime=1;
         $destring="problem";
         $quarantine_description='Disallowed content found in MIME attachment -
potential virus';
         $quarantine_event="Policy:Bad_MIME_Header";
         $description .= "\n---perlscanner results ---\n$destring
'$quarantine_description' found in message";
       }
      }
    }
    if ($begin_content =~ /base64/i && !/^\s/) {
      #&debug("w_c: begin=\"$begin_content\",line=$_");
      $begin_content='';
      #Only looking for base64 encoded as both QP and binary appear to arrive
corrupted under Outlook
      if ($_ =~ /^TV(qq|qQ|r1|o8|ou)/) {
       &debug("w_c: base64 looks like a Windows executable,
filename=$attachment_filename,type=$content_type{$attachment_counter}");
       if (!$quarantine_event && $BAD_MIME_CHECKS > 1 &&
$content_type{$attachment_counter} !~ /^application/i) {
         #As far as I'm aware, a Windows/DOS executable should always be of type
"application/<something>"
         $illegal_mime=1;
         $destring="problem";
         $quarantine_description="Disallowed executable attachment associated with
\"$content_type{$attachment_counter}\" MIME type - potential virus";
         $quarantine_event="Policy:Forged_Attachment";
         $description .= "\n---perlscanner results ---\n$destring
'$quarantine_description' found in attachment \"$attachment_filename\"";
         &debug("w_c: $quarantine_description");
         &minidebug("w_c: $quarantine_description");
       }
      }
      if ($_ =~ /^(UEsDB[AB]|UEswMFBL)/) {
       &debug("w_c: base64 looks like a zip file,
filename=$attachment_filename,type=$content_type{$attachment_counter}");
       if (!$quarantine_event && $BAD_MIME_CHECKS > 2 && $attachment_filename !~
/\.zip$/i) {
         #This is a zip file, and yet the filename doesn't end in .zip - should
quarantine it!
         $illegal_mime=1;
```

```
1095          $destring="problem";
              $quarantine_description="Disallowed zip attachment when not assosiated with
a .zip filename - potential virus";
              $quarantine_event="Policy:Forged_Attachment";
              $description .= "\n---perlscanner results ---\n$destring
1100 '$quarantine_description' found in attachment \"$attachment_filename\"";
              &debug("w_c: $quarantine_description");
              &minidebug("w_c: $quarantine_description");
            }
          }
1105      }
          if ($BOUNDARY_REGEX ne "" && /^\-\-($BOUNDARY_REGEX)/) {
            $still_attachment=$_;
            chomp($still_attachment);
            if (/^\-\-(.*)\-\-.$/) {
1110         &debug("w_c: found end of attachment boundary, BOUNDARY_REGEX was
\"$BOUNDARY_REGEX\"...");
             my ($delete_bb)=$1;
             $delete_bb =~ s/(\W)/\\$1/g;
             $BOUNDARY_REGEX =~ s/\Q$delete_bb\E//;
1115         $BOUNDARY_REGEX =~ s/\|\|//;
             $BOUNDARY_REGEX =~ s/(^\||\|$)//;
             &debug("w_c: now that \"$delete_bb\" has been removed, it's
\"$BOUNDARY_REGEX\"...");
            }
1120        $attachment_counter++;
            #&debug("w_c: found :$BOUNDARY_REGEX: - must be attachment section
$attachment_counter");
          }
          if ($log_crypto && $CRYPTO_TYPE eq "") {
1125        $CRYPTO_TYPE="CR:PGP(old-signed)" if (/^\-\-\-\-\-BEGIN PGP SIGNATURE\-\-\-\-\-
/);
            $CRYPTO_TYPE="CR:PGP(old-encrypted)" if (/^\-\-\-\-\-BEGIN PGP MESSAGE\-\-\-\-
\-/);
            &debug("found old PGP crypto ($CRYPTO_TYPE)") if ($CRYPTO_TYPE ne "");
1130        &minidebug("found old PGP crypto ($CRYPTO_TYPE)") if ($CRYPTO_TYPE ne "");
          }
          &check_and_grab_uuencoding;
          print TMPFILE ;
        }
1135    close(TMPFILE)||&error_condition("cannot close $scandir/$wmaildir/tmp/$file_id -
$!");

        #Set the tag_score after the file as been read
        #$tag_score .= "$CRYPTO_TYPE:" if ($log_crypto && $CRYPTO_TYPE ne "");
1140    $HEADERS =~ s/\r|\0//g;

        &debug("w_c: rename new msg from $scandir/$wmaildir/tmp/$file_id to
$scandir/$wmaildir/new/$file_id [",&deltatime,"]");
        #Not atomic but who cares about the overhead - this is the only app using this
1145 area...

link("$scandir/$wmaildir/tmp/$file_id","$scandir/$wmaildir/new/$file_id")||&error_con
dition("cannot link $scandir/$wmaildir/tmp/$file_id into
$scandir/$wmaildir/new/$file_id - $!");
1150    unlink("$scandir/$wmaildir/tmp/$file_id")||&error_condition("cannot delete
$scandir/$wmaildir/tmp/$file_id - $!");
        my ($dev,$ino,$mode,$nlink,$uid,$gid,$rdev,$atime,$mtime,$ctime,$blksize,$blocks);

($dev,$ino,$mode,$nlink,$uid,$gid,$rdev,$msg_size,$atime,$mtime,$ctime,$blksize,$bloc
1155 ks) = stat("$scandir/$wmaildir/new/$file_id");
        if (!$headers{'date'}) {
          my (@day, @mon);

$day[0]='Sun';$day[1]='Mon';$day[2]='Tue';$day[3]='Wed';$day[4]='Thu';$day[5]='Fri';$
1160 day[6]='Sat';

$mon[0]='Jan';$mon[1]='Feb';$mon[2]='Mar';$mon[3]='Apr';$mon[4]='May';$mon[5]='Jun';$
mon[6]='Jul';$mon[7]='Aug';$mon[8]='Sep';$mon[9]='Oct';$mon[10]='Nov';$mon[11]='Dec';
          my
1165 ($tm_sec,$tm_min,$tm_hour,$tm_mday,$tm_mon,$tm_year,$tm_wday,$tm_yday,$tm_isdst);

($tm_sec,$tm_min,$tm_hour,$tm_mday,$tm_mon,$tm_year,$tm_wday,$tm_yday,$tm_isdst)=loca
ltime;
```

```
    $tm_year += 1900;
    $headers{'date'}=$day[$tm_wday].", $tm_mday ".$mon[$tm_mon]." $tm_year
$tm_hour:$tm_min:$tm_sec";
  }
}

sub grab_envelope_hdrs {
  select(STDOUT); $|=1;

  open(SOUT,"<&1")||&error_condition("cannot dup fd 0 - $!");
  while (<SOUT>) {
    ($env_returnpath,$env_recips) = split(/\0/,$_,2);
    if ( ($returnpath=$env_returnpath) =~ s/^F(.*)$// ) {
      $returnpath=$1;
      ($recips=$env_recips) =~ s/^T//;
      $recips =~ /^(.*)\0+$/;
      $recips=$1;
      $recips =~ s/\0+$//g;
      #Keep a note of the NULL-separated addresses
      $trecips=$recips;
      $one_recip=$trecips if ($trecips !~ /\0T/);
      $recips =~ s/\0T/\,/g;
    }
    #only meant to be one line!
    last;
  }
  close(SOUT)||&error_condition("cannot close fd 1 - $!");
  if ( ($env_returnpath eq "" && $env_recips eq "") || ($returnpath eq "" && $recips
eq "") ) {
    #At the very least this is supposed to be $env_returnpath='F' - so
    #qmail-smtpd must be officially dropping the incoming message for
    #some (valid) reason (including the other end dropping the connection).
    &debug("g_e_h: no sender and no recips.");
    &minidebug("g_e_h: no sender and no recips, from $smtp_sender. Dropping.");
    &cleanup;
    &close_log;
    exit;
  }
  &debug("g_e_h: return-path is \"$returnpath\", recips is \"$recips\"");
  &minidebug("return-path='$returnpath', recips='$recips'");
}


sub deconstruct_msg {
  my ($start_decon_time) = [gettimeofday];
  my $save_filename ='';
  my ($new_filename,$MAYBETNEF,$tnef_status);

  &debug("d_m: starting $mimeunpacker_binary <$scandir/$wmaildir/new/$file_id
[",&deltatime,"]");
  open(MIME,"$mimeunpacker_binary  <$scandir/$wmaildir/new/$file_id
2>&1|")||&error_condition("cannot call $mimeunpacker_binary - $!");
  while (<MIME>) {
    next if (/exists/);
    &error_condition("d_m: output spotted from $mimeunpacker_binary ($_) - that
shouldn't happen!");
  }
  close(MIME)||&error_condition("cannot close $mimeunpacker_binary - $!");
  my $unpacker='';

  opendir(DIR,"$ENV{'TMPDIR'}/")||&error_condition("cannot open dir $ENV{'TMPDIR'}/ -
$!");
  my @allfiles = grep(!/^\.+$/, readdir(DIR));
  closedir(DIR);
  &debug("d_m: finished $mimeunpacker_binary [",&deltatime,"]");
  #If you have the tnef app, you'll be able to scan broken M$ attachments

  if ( $tnef_binary ) {
    &debug("d_m: Checking all attachments to see if they're MS-TNEF");
    foreach $save_filename (@allfiles) {
      #Clean up $save_filename so as to keep taint happy
      $save_filename =~ /^(.*)$/; $save_filename=$1;
      ($new_filename=$save_filename) =~ s/([^a-z0-9\.\-\_\+\=\~]+)//gi;
      if ($save_filename ne $new_filename) {
```

```
        $new_filename =~ /(\.[^\.]+)$/;
        $new_filename=&uniq_id."$new_filename";
        rename($save_filename,$new_filename);
        &debug("d_m: ren $save_filename to $new_filename");
        $save_filename=$new_filename;
       }
       #Who cares if it is or isn't tnef, just scan it!
       if ($tnef_binary) {
        $MAYBETNEF=`$tnef_binary --number-backups -d $ENV{'TMPDIR'}/ -f
$ENV{'TMPDIR'}/$save_filename 2>&1`;
        $tnef_status=$?;
        &debug("d_m: is $ENV{'TMPDIR'}/$save_filename is a TNEF file?: $tnef_status
[",&deltatime,"]");
       }
     }
   }

   #If you're happy with your scanners zip file handling, you can
   #skip this whole section. McAfee's doesn't support zip for starters!

   if ($force_unzip || $BLOCK_PASSWORD_PROTECTED_ARCHIVES || $log_crypto) {
     &debug("d_m: Check for zip files...");
     #Re-initialize directory listing
     opendir(DIR,"$ENV{'TMPDIR'}/")||&error_condition("cannot open dir $ENV{'TMPDIR'}/
- $!");
     @allfiles = grep(!/^\.+$/, readdir(DIR));
     closedir(DIR);
     foreach $save_filename (@allfiles) {
       #Clean up $save_filename so as to keep taint happy
       $save_filename =~ /^(.*)$/; $save_filename=$1;
       ($new_filename=$save_filename) =~ s/([^a-z0-9\.\-\_\+\=\~]+)//gi;
       if ($save_filename ne $new_filename) {
        $new_filename =~ /(\.[^\.]+)$/;
        $new_filename=&uniq_id."$new_filename";
        rename($save_filename,$new_filename);
        &debug("d_m: ren $save_filename to $new_filename");
        $save_filename=$new_filename;
       }
       if ( $save_filename =~ /\.(zip|exe)$/i) {
        &unzip_file($save_filename);
       }
     }
   }
   if (!$redundant_scanning) {
     if (-f "$ENV{'TMPDIR'}/$save_filename") {
       system $rm_binary,"-f","$ENV{'TMPDIR'}/$save_filename";
     }
   }
   my($decon_time)=tv_interval ($start_decon_time, [gettimeofday]);
   &debug("d_m: unpacking message took $decon_time seconds");
}

sub init_scanners {
   my($start_init_scanners_time)=[gettimeofday];
   &debug("ini_sc: start scanning");
   chdir("$ENV{'TMPDIR'}/");

   #Delete original zip'ped attachment as there's no point
   #in the other scanners double-scanning it - unless $redundant_scanning
   #is set....
   if ($redundant_scanning) {
     link("$scandir/$wmaildir/new/$file_id","$ENV{'TMPDIR'}/orig-$file_id");
   }
   &debug("ini_sc: recursively scan the directory $ENV{'TMPDIR'}/");

   #Run AV scanners - even if the message is already going to be quarantined
   #due to some Policy: this way you get the definitive answer as to what is
   #a virus...

# st: Run first p_s? ####### I don't think that this is actually useful but...

   if ($run_first_p_s) {
     &perlscan_scanner if (!$quarantine_event);
     #
```

```
    &scanloop if (!$quarantine_event);
  } else {
    &scanloop; #JLH if (!$quarantine_event);

    #Only run perlscanner if no reason to quarantine found so far
    &perlscan_scanner if (!$quarantine_event);
  }

############################

  chdir("$scandir");

  # st: mark the viruses we don't want to quarantine, but delete them
  if (($virus_to_delete ne "") && ($quarantine_description=~/($virus_to_delete)/i)) {
    $del_message='1';
    &debug("v_t_d: Virus ($quarantine_description), dropping");
    &minidebug("v_t_d: Virus ($quarantine_description), dropping");
  }

  my($decon_time)=tv_interval ($start_init_scanners_time, [gettimeofday]);
  &debug("ini_sc: scanning message took $decon_time seconds");
  &minidebug("ini_sc: finished scan of \"$ENV{'TMPDIR'}\"...");

  # st: let see the elapsed time from start
  $elapsed_1=tv_interval ($start_time, [gettimeofday]);
  &minidebug("ini_sc: elapsed time from start $elapsed_1 secs");
}


sub perlscan_scanner {
  #This is most efficient if called from within deconstruct_msg

  my($start_perlscan_time)=[gettimeofday];
  my (%array,$var,$lfile,$filename,$section,$apptype,$save_filename);
  my ($type,$desc,$file,@allfiles);
  my ($ps_skipfile,$extension);
  my
($dev,$ino,$mode,$nlink,$uid,$gid,$rdev,$size,$atime,$mtime,$ctime,$blksize,$blocks,$
fsize);
  my ($attachment_list,$perlscan_time);
  &debug("p_s: starting scan of directory \"$ENV{'TMPDIR'}\"...");

  use DB_File;


  tie (%array, 'DB_File', "$db_filename.db", O_RDONLY, 0600) ||
&error_condition("cannot open  $db_filename.db - $!");

  if (!$quarantine_event && $illegal_mime && $headers{'mime-version'} &&
$BAD_MIME_CHECKS) {
    $destring="problem";
    $quarantine_description="Disallowed  characters found in MIME headers" if
(!$quarantine_description);
    $quarantine_event="Policy:Bad_MIME";
    $description .= "\n---perlscanner results ---\n$destring
'$quarantine_description'\n found in message";
  }
  #check out headers against DB...

  foreach $var (sort keys(%array)) {
    ($type,$desc)=split(/\t/,$array{$var},2);
    &debug("p_s:  '$var' = '$type' = '$desc'");
    if ($type !~ /^[0-9]+$/) {
      &debug("p_s:  type is a header!");
      #Strip off numeric uid...
      $var =~ s/^[0-9]+://;
      $type =~ s/^Virus-//g;
      &debug("p_s:  checking for objects containing $type: $var");
      if ($headers{$type} =~ /^$var$/) {
       $quarantine_description="$desc";
       ($quarantine_event=$quarantine_description) =~ s/\s/_/g;
       $quarantine_event="Perlscan:".substr($quarantine_event,0,$QE_LEN);
       $quarantine_event=~s/_$//g;
```

```
1390        $description .= "\n---perlscanner results ---\n$destring
'$quarantine_description' found in file $ENV{'TMPDIR'}/$file";
            &debug("p_s: something to block! ($quarantine_description)");
            &minidebug("p_s: something to block! ($quarantine_description)");
           }
1395      } else {
            &debug("p_s: type is a size!");
          }
        }

1400    opendir(DIR,"$ENV{'TMPDIR'}/")||&error_condition("cannot open dir $ENV{'TMPDIR'}/ -
$!");
        @allfiles = grep(!/^\.+$/, readdir(DIR));
        closedir(DIR);

1405    #append any ORIGINAL uuencoded filenames to this directory array
        #so that perlscanner can match on uuencoded filenames
        foreach $file (@allfiles,@uufile_list,@zipfile_list,@attachment_list) {
          #skip files that reformime/ripmime generates.
          #This will potentially allow baddies to smuggle files through
1410      #by using filenames like this... Nothing can be done about that:-(
          #Reformime generates filenames of the form:
          # 967067231.24320-X.host.name (where X is a number)
          #Ripmime generates filenames of the form:
          # textfileX (where X is a number)
1415      if ($file =~ /^[0-9]+\.[0-9]+\-[0-9]+\.$hostname|^$file_id|^textfile[0-9]+/) {
            &debug("p_s: skipping auto-generated file $file");
            $ps_skipfile=1;
          } else {
            &debug("p_s: checking $file against perlscanner database...");
1420        $ps_skipfile=0;
          }

          if (!$quarantine_event && length($file) > 256 && $BAD_MIME_CHECKS > 1 ) {
            &debug("w_c: majorly long attachment filename found - block it");
1425        &minidebug("w_c: majorly long attachment filename found - block it");
            $illegal_mime=1;
            $destring='problem';
            $quarantine_description="Disallowed attachment file length found
(".length($file).") - potential virus";
1430        $quarantine_event="Policy:Attach_Length";
            $description .= "\n---perlscanner results ---\n$destring
'$quarantine_description' found in file $ENV{'TMPDIR'}/$file";
            $file_desc .= "$file:$msg_size\t" if ($file_desc !~ /\Q$file\E:$size\t/);
            return;
1435      }

          #Do the patently obvious filename security checks here
          if ( $BAD_MIME_CHECKS > 1) {
            #Not as thorough as I'd like - but I got too many false positives doing it more
1440    generically... :-(
            #The VALID_WINDOWS_EXTENSIONS is based on double-barrel virii caught in a years
worth of Qmail-Scanner
            #logs (gotta love those logs!). Notice that I expressly allow "file.exe.exe"
through - as the double-extension
1445        #doesn't hide anything [just implies a user made a mistake]
            if (!$quarantine_event && ($file =~
/(^.*)\.($VALID_WINDOWS_EXTENSIONS)\.($SNEAKY_WINDOWS_EXTENSIONS)$/i) && $file !~
/(\.[a-z0-9]{3})\1$/) {
             $quarantine_description="Disallowed double-barrelled attachment filename
1450    ($file) - potential virus";
             &debug("w_c: $quarantine_description");
             &minidebug("w_c: $quarantine_description");
             $illegal_mime=1;
             $destring='problem';
1455         $quarantine_event="Policy:Win_Ext";
             $description .= "\n---perlscanner results ---\n$destring
'$quarantine_description' found in file $ENV{'TMPDIR'}/$file";
             $file_desc .= "$file:$msg_size\t" if ($file_desc !~ /\Q$file\E:$size\t/);
             return;
1460        }
            if (!$quarantine_event && $file =~ /\{[0-9a-f]{8}\-[0-9a-f]{4}\-[0-9a-f]{4}\-
[0-9a-f]{4}\-[0-9a-f]{12}\}$/i) {
```

```
        $quarantine_description="Disallowed CLSID file extensions ($file) - potential
virus";
1465    &debug("w_c: $quarantine_description");
        &minidebug("w_c: $quarantine_description");
        $illegal_mime=1;
        $destring='problem';
        $quarantine_event="Policy:Win_CLSID";
1470    $description .= "\n---perlscanner results ---\n$destring
'$quarantine_description' found in file $ENV{'TMPDIR'}/$file";
        $file_desc .= "$file:$msg_size\t" if ($file_desc !~ /\Q$file\E:$size\t/);
        return;
       }
1475   }
    if ($file =~ /(^.*)(\.[^\.]+)\.?$/) {
      $extension=tolower($2);
    } else {
      $extension="";
1480   }
    $lfile = tolower($file);
    &debug("p_s: file $file is lowercased to $lfile and has extension $extension") if
(!$ps_skipfile);
    #Stat'ing attachment names from @attachment_list will fail on filenames that
1485 reformime rewrites
    #that's OK, as they'll still be picked up via their new filename

($dev,$ino,$mode,$nlink,$uid,$gid,$rdev,$size,$atime,$mtime,$ctime,$blksize,$blocks)
= stat("$file");
1490    if ($ino && $file_desc !~ /\Q$file\E:$size\t/) {
      #Sanity check so that the virtual attachments don't get double-counted
      $file_desc .= "$file:$size\t";
    }
    &debug("p_s: compare $lfile (size $size) against perlscanner database") if
1495 (!$ps_skipfile);
    if ( ($array{$lfile} || $array{$extension}) && !$ps_skipfile ) {
      if ($array{$lfile}) {
       ($fsize,$quarantine_description) = split(/\t/,$array{$lfile},2);
      } else {
1500   $destring="Disallowed attachment type";
       ($fsize,$quarantine_description) = split(/\t/,$array{$extension},2);
      }
      $attachment_list.="$file:$size,";
      if (!$quarantine_event && $size eq $fsize || $fsize =~ /^(\-|\*|any|0)$/i ) {
1505   &debug("p_s: Quarantine $file! ($quarantine_description)");
       &minidebug("p_s: Quarantine $file! ($quarantine_description)");
       ($quarantine_event=$quarantine_description) =~ s/\s/_/g;
       $quarantine_event="Perlscan:".substr($quarantine_event,0,$QE_LEN);
       $quarantine_event=~s/_$//g;
1510   $description .= "\n---perlscanner results ---\n$destring
'$quarantine_description' found in file $ENV{'TMPDIR'}/$file";
       $section=$apptype=$save_filename=$filename="";
#       return;
      }
1515   }
  }
  # st: cosmetic, if the messages is spam don't call it a virus.
  if ($quarantine_description =~ /spam/i) {
     $destring="Problem";
1520 }
  untie %array;
  chdir("$scandir/");
  my($stop_perlscan_time)=[gettimeofday];
  $perlscan_time = tv_interval ($start_perlscan_time, $stop_perlscan_time);
1525 &debug("p_s:  finished scan of dir \"$ENV{'TMPDIR'}\" in $perlscan_time secs");
  &minidebug("p_s: finished scan in $perlscan_time secs");
}


1530 sub scanloop {
  &debug("scanloop: starting scan of directory \"$ENV{'TMPDIR'}\"...");

  my ($scanner);

1535 #Remember any policy blocks that have already occurred, but reset
  #$quarantine_event so that if a virus is found, that "wins"
```

```
  $quarantine_event_tmp=$quarantine_event;
  $quarantine_event='0';

  foreach $scanner (@scanner_array) {
    #Any scanner errors caused by broken zip files/etc will be ignored
    # - not sure how that should be handled...
    &debug("scanloop: scanner=$scanner,plain_text_msg=$plain_text_msg");

    # st: call spamassassin_alt if sa_alt is enabled
    $scanner = "spamassassin_alt" if ( $scanner =~ /spam/i && $sa_alt eq "1" );

    #Just run virus scanners over mail that isn't plain text
    if ($plain_text_msg) {
      #If it's plain text - just run anti-spam checks
      &{$scanner} if ($scanner =~ /spam/i);
    }else {
      &{$scanner};
    }
    if ($quarantine_event) {
      #If one scanner finds a virus - why run the rest over it?
      last;
    }
  }
  if (!$quarantine_event) {
    $quarantine_event=$quarantine_event_tmp;
  } else {
    #Make sure this is set correctly
    $destring="virus" if ($quarantine_event !~ /spam/i);
  }
  &debug("scanloop: finished scan of \"$ENV{'TMPDIR'}\"...");
}

sub qmail_requeue {
  my($sender,$env_recips,$msg)=@_;
  my ($temp,$findate);

  &debug("q_r: fork off child into $qmailqueue...");

  #($recips=$env_recips) =~ s/^T//;
  #$recips =~ s/\0T/\,/g;
  #$recips =~ /^(.*)\0+$/;
  #$recips = $1;
  #$recips =~ s/\0+$//g;

  # Create a pipe through which to send the envelope addresses.
  pipe (EOUT, EIN) or &error_condition("Unable to create a pipe. - $!");
  select(EOUT);$|=1;
  select(EIN);$|=1;
  # Fork qmail-queue.  The qmail-queue child will then open fd 0 as
  # $message and fd 1 as the reading end of the envelope pipe and exec
  # qmail-queue.  The parent will read in the addresses and pass them
  # through the pipe and then check the exit status.

  $elapsed_time = tv_interval ($start_time, [gettimeofday]);
  local $SIG{PIPE} = 'IGNORE';
  my $pid = fork;

  if (not defined $pid) {
    &error_condition ("Unable to fork. (#4.3.0) - $!");
  } elsif ($pid == 0) {
    # In child.  Mutilate our file handles.
    close EIN;

    open(STDIN,"<$msg")|| &error_condition ("Unable to reopen fd 0. (#4.3.0) - $!");

    open (STDOUT, "<&EOUT") ||  &error_condition ("Unable to reopen fd 1. (#4.3.0) -
$!");
    select(STDIN);$|=1;
    &debug("q_r: xstatus=$xstatus");
    open (QMQ, "|$qmailqueue")|| &error_condition ("Unable to open pipe to
$qmailqueue [$xstatus] (#4.3.0) - $!");
    ($sec,$min,$hour,$mday,$mon,$year) = gmtime(time);
    $elapsed_time = tv_interval ($start_time, [gettimeofday]);
    $findate = POSIX::strftime( "%d %b ",$sec,$min,$hour,$mday,$mon,$year);
```

```
    $findate .= sprintf "%02d %02d:%02d:%02d -0000", $year+1900, $hour, $min, $sec;
    print QMQ "Received: from $returnpath by $hostname by uid $uid with qmail-
scanner-$VERSION \n";
    print QMQ " ($SCANINFO Clear:$tag_score. \n";
    print QMQ " Processed in $elapsed_time secs); $findate\n";
    print QMQ "X-Spam-Status: $sa_comment\n" if ($sa_comment ne "");
    print QMQ "X-Spam-Level: $sa_level\n" if ($sa_comment ne "" && $sa_level ne "");
    if ( $descriptive_hdrs ) {
      print QMQ "${V_HEADER}-Mail-From: $returnpath via $hostname\n";
      print QMQ "${V_HEADER}-Rcpt-To: $recips\n" if ($descriptive_hdrs eq "2");
      print QMQ "$V_HEADER: $VERSION (Clear:$tag_score. Processed in $elapsed_time
secs Process $nprocess)\n";
    }
    my $still_headers=1;
    my $seen_env=0;
    while (<STDIN>) {
      if ($still_headers) {
       #if (!$seen_env && /^X\-Envelope\-From:/) {
         #$seen_env=1;
         #just skip the next line (X-Envelope-To:)
         #<STDIN>;
         #next;
       #}
       #remove any X-Spam-Status/Level IFF we've set a SA value ourselves
       if (($sa_comment ne "" && /^X-Spam-Status:/i) || ($sa_level ne "" && /^X-Spam-
Level:/i) ) {
         #Hmm, better get rid of any other continuation headers to this!
         while (<STDIN>) {
           $still_headers=0 if (/^(\r|\r\n|\n)$/);
           if ($still_headers && /^\s/i) {
             next;
           } else {
             $still_headers=1;
             last;
           }
         }
       }
       if ($sa_comment =~ /^yes/i && $spamc_subject ne "" && !/^Subject:
\Q$spamc_subject\E/i && /^(Subject):(\s?)([^\n]+)\n/i ) {
         $altered_subject="$1: $spamc_subject $3";
         if ($altered_subject !~ /^: \Q$spamc_subject\E/) {
           &debug("altering subject line to $altered_subject");
           print QMQ "$altered_subject\n";
           next;
         }
       }
       $still_headers=0 if (/^(\r|\r\n|\n)$/);
        #Insert Subject: line if e-mail dosn't contain one but must be tagged
        print QMQ "Subject: $spamc_subject\n" if ((!$still_headers) && ($sa_comment
=~ /^yes/i) && (!$altered_subject) && $spamc_subject ne "" );

      }
      print QMQ;
    }
    close(QMQ); #||&error_condition("Unable to close pipe to $qmailqueue (#4.3.0) -
$!");
    $xstatus = ( $? >> 8 );
    if ( $xstatus > 10 && $xstatus < 41 ) {
      &error_condition("mail server permanently rejected message. (#5.3.0) -
$!",$xstatus);
    } elsif ($xstatus > 0) {
      &error_condition("Unable to open pipe to $qmailqueue [$xstatus] (#4.3.0) -
$!",$xstatus);
    }
    #This child is finished - exit
    exit;
  } else {
    # In parent.
    close EOUT;

    # Feed the envelope addresses to qmail-queue.
    print EIN "$sender\0$env_recips";
    close EIN  || &error_condition ("Write error to envelope pipe. (#4.3.0) - $!");
}
```

```
  # We should now have queued the message.  Let's find out the exit status
  # of qmail-queue.
  waitpid ($pid, 0);
  $xstatus =($? >> 8);
  if ( $xstatus > 10 && $xstatus < 41 ) {
    &error_condition("mail server permanently rejected message. (#5.3.0) -
$!",$xstatus);
  } elsif ($xstatus > 0) {
    &error_condition("Unable to close pipe to $qmailqueue [$xstatus] (#4.3.0) -
$!",$xstatus);
  }
}


sub valid_virus_to_report {
  my ($virus_type)=@_;
  my ($virus)='';
  # This subroutine is used to determine if the virus found during the scan
  # is reportable. i.e. do we want to send a message to this user or not as is
  # the case with the KLEZ virus.
  #&debug("v_v_t_r: called with $virus_type");
  foreach $virus (@silent_viruses_array) {
    #&debug("v_v_t_r: does $virus_type contain $virus?");
    if ($virus_type =~ /$virus/i) {
      &debug("v_v_t_r: $virus_type contain $virus - so don't notify the sender");
      &minidebug("v_v_t_r: Description contain \"$virus\" - so don't notify the
sender");
      return 0;
    }
  }
  return 1;
}

sub automated_msg {
  if ($headers{'x-loop'} || $headers{'x-listname'} || $headers{'x-listmember'} ||
$headers{'mailing-list'} || $headers{'x-mailing-list'} || $headers{'precedence'} =~
/^(bulk|list|junk)$/i || $returnpath =~
/^$|^\#\@\[\]$|anonymous|nobody|daemon|request|bounce|mailer|postm|owner|lists|words|
majordom|experts|\-(return|error)/i) {
    return 1;
  } else {
    return 0;
  }
}

sub bounce_msg {
  if ($returnpath =~ /^$|^\#\@\[\]$|(daemon|bounce|mailer|postm)/i) {
    return 1;
  } else {
    return 0;
  }
}

sub is_unreplyable_email {
  my ($addr_type)=@_;
  my ($dom,$is_local)='';
  #This subroutine is used to see if the sender of this message
  #was a mailing-list/postmaster/etc, or the recipient is a local user.
  #If it is we don't want to send a reply.
  #&debug("i_u_e: called with $addr_type");

  if ($addr_type eq "recips") {
    foreach $dom (@local_domains_array) {
      #&debug("i_u_e: does $recips contain $dom?");
      if ($recips =~ /$dom$/i) {
       #&debug("i_u_e: yes it does!");
       $is_local++;
      }
    }
  } else {
    $is_local="99";
    if (&automated_msg ) {
      #&debug("i_u_e: $addr_type is a mailing-list");
```

```
      return 1;
    }
  }
  #
  #Only reply if it is a local address
  if (!$is_local) {
    #&debug("i_u_e: is_local=$is_local");
    return 1;
  } else {
    #&debug("i_u_e: is_local=$is_local");
    return 0;
  }
}

sub email_quarantine_report {
  my($start_email_time)=[gettimeofday];
  &debug("e_v_r: quarantine  msg to $scandir/$vmaildir/new/$file_id");

link("$scandir/$wmaildir/new/$file_id","$scandir/$vmaildir/new/$file_id")||&error_con
dition("cannot link $scandir/$wmaildir/new/$file_id into $scandir/$vmaildir/new/ -
$!");
  open(QTINE,">>$scandir/$vmaildir/new/$file_id");
  print QTINE "\n*** Qmail-Scanner Quarantine Envelope Details Begin ***\n";
  print QTINE "${V_HEADER}-Mail-From: \"$returnpath\" via $hostname\n";
  print QTINE "${V_HEADER}-Rcpt-To: \"$recips\"\n";
  print QTINE "$V_HEADER: $VERSION ($SCANINFO $destring Found. Processed in
",tv_interval($start_time,[gettimeofday])," secs) process $nprocess \n";
  print QTINE "Quarantine-Description: $quarantine_description\n";
  if (($quarantine_description =~ /spam/i) && $sa_report) {
    print QTINE "SA_REPORT hits = $sa_score/$required_hits\n$sa_report\n";
  }
  print QTINE "*** Qmail-Scanner Envelope Details End ***\n";
  close QTINE;
  &email_sender("sender") if (&valid_virus_to_report($quarantine_description));
  &email_sender("admin");
  if ($trecips =~ /\0T/) {
    for $recip (split(/\0T/,$trecips)) {
      &email_recips($recip);
    }
  } else {
    &email_recips($recips);
  }
  &write_quarantine_report;
  $elapsed_time = tv_interval ($start_time, [gettimeofday]);
  &debug("e_v_r: email_quarantine_report took ".tv_interval ($start_email_time,
[gettimeofday])." seconds to execute");
}

sub cleanup {
  closelog();
  chdir("$scandir/");
  if ($archiveit !~ /^(1|yes)$/i) {
    #This will only archive mail where the sender or recipient matches the regex that
is $archiveit
    if ($headers{'MAILFROM'} !~ /$archiveit/i && $headers{'RCPTTO'} !~ /$archiveit/i)
{
      $archiveit=0;
    }
  }
  if (!$archiveit) {
    &debug("cleanup: $rm_binary -rf $ENV{'TMPDIR'}/ $scandir/$wmaildir/new/$file_id")
;
  } else {
    # check if $archivedir exists
    if (! -d "$scandir/$archivedir") {
      mkdir("$scandir/$archivedir",0700) || &error_condition("cannot create
$scandir/$archivedir - $!");
      mkdir("$scandir/$archivedir/new",0700) || &error_condition("cannot create
$scandir/$archivedir/new - $!");
      mkdir("$scandir/$archivedir/cur",0700) || &error_condition("cannot create
$scandir/$archivedir/cur - $!");
      mkdir("$scandir/$archivedir/tmp",0700) || &error_condition("cannot create
$scandir/$archivedir/tmp - $!");
    }
```

```
    if ( -f "$scandir/$wmaildir/new/$file_id" ) {
      &debug("cleanup: archiving into $scandir/$archivedir/new/");
      &minidebug("cleanup: archiving into $scandir/$archivedir/new/");
      rename("$scandir/$wmaildir/new/$file_id","$scandir/$archivedir/new/$file_id");
      #This will do for now. Not pretty - but very cheap!
      #We need to append this information, otherwise how do you know who this message
      #was from or to?
      #
      open(ARCHIVE,">>$scandir/$archivedir/new/$file_id");
      print ARCHIVE "\n*** Qmail-Scanner Envelope Details Begin ***\n";
      print ARCHIVE "${V_HEADER}-Mail-From: \"$returnpath\" via $hostname\n";
      print ARCHIVE "${V_HEADER}-Rcpt-To: \"$recips\"\n";
      print ARCHIVE "$V_HEADER: $VERSION ($SCANINFO  Clear:$tag_score. Processed in
",tv_interval($start_time,[gettimeofday])," secs)\n";
      print ARCHIVE "*** Qmail-Scanner Envelope Details End ***\n";
      close ARCHIVE;
    }
  }
  system("$rm_binary -rf $ENV{'TMPDIR'}/ $scandir/$wmaildir/new/$file_id") if ($DEBUG
< 100 && $file_id ne "");

}


sub scan_queue {
  my ($scanner,$SCANINFO,$files,$sweep_eng,$sweep_product,$sophie_eng,$dir);
  my $start_scan_time =time;
  my ($inocucmd_eng,$inocucmd_product,$spamassassin_eng);

  chdir($scandir);
  &debug("s_q: re-create the quarantine version file");
  &minidebug("s_q: re-create the quarantine version file");
  foreach $scanner (@scanner_array) {
    $scanner =~ s/_scanner//;
    &debug("s_q: detecting version of $scanner");
    if ($scanner eq "uvscan") {
      open(UV,"$uvscan_binary --version|")||die "failed to call $uvscan_binary --
version - $!";
      while (<UV>) {
       chomp;
       if (/^Scan engine (v[0-9\.]+) /) {
         $SCANINFO .="uvscan: $1/";
       } elsif (/^Virus data file (v[0-9\.]+) /) {
         $SCANINFO .= "$1. ";
       }
      }
      close(UV);
    } elsif ($scanner eq "csav") {
      open(CS,"$csav_binary -virno|")||die "failed to call $csav_binary -virno - $!";
      while (<CS>) {
       chomp;
       if (/Command Software AntiVirus for Linux (version [0-9\.]+)/) {
         $SCANINFO .="csav: $1/";
       } elsif (/^CSA[V]: (.*)/) {
         $SCANINFO .= "$1/";
       }
      }
      close(CS);
    } elsif ($scanner eq "trophie" ) {
      open(IS,"$trophie_binary -v 2>&1|")||die "failed to call $trophie_binary -v -
$!";
      while (<IS>) {
       chomp;
       if (/VSAPI version (.*)/) {
         $SCANINFO .= "trophie: $1/";
       } elsif (/Pattern version ([0-9]+) \(pattern number ([0-9]+)\)/) {
         $SCANINFO .= "$1/$2. ";
       }
      }
      close(IS);
    } elsif ($scanner eq "iscan") {
      open(IS,"$iscan_binary -v|")||die "failed to call $iscan_binary -v - $!";
      while (<IS>) {
       chomp;
```

```
       if (/Virus Scanner (v[0-9\.]+), VSAPI (v[0-9\.\-]+)/) {
         $SCANINFO .="iscan: $1/$2/";
       } elsif (/Pattern version ([0-9\.]+)/) {
         $SCANINFO .= "$1/";
       } elsif (/Pattern number ([0-9\.]+)/) {
         $SCANINFO .= "$1. ";
       }
      }
      close(IS);
    } elsif ($scanner eq "fsecure") {
      open(FS,"$fsecure_binary --version|")||die "failed to call $fsecure_binary --
version - $!";
      while (<FS>) {
        chomp;
        if (/^F-Secure.*(Release|version) ([0-9\.]+) build ([0-9]+)/i) {
          $SCANINFO .="fsecure: $2/$3/";
        } elsif (/sign.def version ([0-9\.]+-[0-9\.]+-[0-9\.]+)/) {
          $SCANINFO .= "$1/";
        } elsif (/fsmacro.def version ([0-9\.]+-[0-9\.]+-[0-9\.]+)/) {
          $SCANINFO .= "$1/";
        } elsif (/sign2.def version ([0-9\.]+-[0-9\.]+-[0-9\.]+)/) {
          $SCANINFO .= "$1. ";
        } elsif (/F-PROT database version (.*)$/) {
          $SCANINFO .= "fprot($1)/";
        } elsif (/AVP FPI Engine database version (.*)$/) {
          $SCANINFO .= "avp($1). ";
        }
      }
      close(FS);
      $SCANINFO .= ". " if ($SCANINFO !~ /\. $/);
    } elsif ($scanner eq "fprot") {
      open(FP,"$fprot_binary \?|")||die "failed to call $fprot_binary --version -
$!";
      while (<FP>) {
       chomp;
       if (/(F-PROT|Program version:) ([0-9\.]+)/) {
         $SCANINFO .="f-prot: $2/";
       } elsif (/Engine version: ([0-9\.]+)/) {
         $SCANINFO .= "$1";
       }
      }
      $SCANINFO .= ". ";
      close(FP);
    } elsif ($scanner eq "hbedv") {
      open(IS,"$hbedv_binary --version 2>&1 |")||die "failed to call  $hbedv_binary -
-version - $!";
      while (<IS>) {
       chomp;
       if (/engine version:\s+([0-9\.]+)/) {
         $SCANINFO .= "hbedv: $1";
       } elsif (/vdf version:\s+([0-9\.]+)/) {
         $SCANINFO .= "/$1. ";
       }
      }
      close(IS);
    } elsif ($scanner eq "avp") {
      open(AVP,"$avp_binary -Y -VL 2>&1 |")||die "failed to call  $avp_binary -Y -VL
- $!";
      while (<AVP>) {
       chomp;
       if (/Version (([0-9\.]+)\s+build ([0-9\.]+)|([0-9\.]+))/) {
         if ($2) {
           $SCANINFO .= "avp: $1/$2. ";
         } else {
           $SCANINFO .= "avp: $1. ";
         }
       }
      }
      close(AVP);
    } elsif ($scanner eq "ravlin") {
      open(RAV,"$ravlin_binary --version 2>&1 |")||die "failed to call
$ravlin_binary --version  - $!";
      while (<RAV>) {
       chomp;
```

```
       if (/^Version: ([0-9\.]+)\./) {
         $SCANINFO .= "ravlin: $1. ";
       }
      }
      close(RAV);
    } elsif ($scanner eq "vexira") {
      open(VEX,"$vexira_binary --version 2>&1 |")||die "failed to call
$vexira_binary --version  - $!";
      while (<VEX>) {
       chomp;
       if (/^engine version:\s+([0-9\.]+)/) {
         $SCANINFO .= "vexira: $1. ";
       }
      }
      close(RAV);
    } elsif ($scanner eq "sophie") {
      open(SOP,"$sophie_binary -v 2>&1|")||die "failed to call $sophie_binary -v -
$!";
      while (<SOP>) {
       chomp;
       if (/Sophos engine version (.*)$/) {
         $sweep_eng=$1;
       } elsif (/Sophos IDE version ([0-9\.]+)/) {
         $sweep_product=$1;
       } elsif (/Sophie version\s+:\s+([0-9\.]+)/) {
         $sophie_eng=$1;
       }
      }
      $SCANINFO .= "sophie: $sophie_eng/$sweep_eng/$sweep_product. ";
      close(SOP);
    } elsif ($scanner eq "sweep") {
      open(SOP,"$sweep_binary -v|")||die "failed to call $sweep_binary -v - $!";
      while (<SOP>) {
       chomp;
       if (/Engine version\s+:\s+(.*)$/) {
         $sweep_eng=$1;
       } elsif (/Product version\s+:\s+(.*)$/) {
         $sweep_product=$1;
       }
      }
      $SCANINFO .= "sweep: $sweep_eng/$sweep_product. ";
      close(SOP);
    } elsif ($scanner eq "inocucmd") {
      open(IOP,"$inocucmd_binary -HEL|")||die "failed to call $inocucmd_binary -HEL -
$!";
      while (<IOP>) {
       chomp;
       if (/Engine version:\s+(.*) ([0-9\/]+)$/) {
         $inocucmd_eng=$1;
       } elsif (/Data version:\s+(.*) ([0-9\/]+)$/) {
         $inocucmd_product=$1;
       }
      }
      $SCANINFO .= "inocucmd: $inocucmd_eng/$inocucmd_product. ";
      close(IOP);
    } elsif ($scanner eq "clamscan") {
      open(CLAMS,"$clamscan_binary --stdout -V|")||die "failed to call
$clamscan_binary --stdout -V - $!";
      while (<CLAMS>) {
       chomp;
       if (/ersion ([0-9\.\-a-z]+)/i) {
         $SCANINFO .="clamscan: $1. ";
       }
      }
      close(CLAMS);
    } elsif ($scanner eq "clamdscan") {
      open(CLAMS,"$clamdscan_binary --version 2>&1|")||die "failed to call
$clamdscan_binary --version - $!";
      while (<CLAMS>) {
       chomp;
       if (/ersion ([0-9\.\-a-z]+)/i) {
         $SCANINFO .="clamdscan: $1. ";
       }
      }
```

```
      close(CLAMS);
    } elsif ($scanner eq "spamassassin") {
      #X-Spam-Checker-Version: SpamAssassin 2.01
      open(SPAS,"$spamassassin_binary  -V |")||die "failed to call
$spamassassin_binary -V - $!";
      $spamassassin_eng="2.x";
      while (<SPAS>) {
       chomp;
       if (/^SpamAssassin version (.*)$/i) {
         $spamassassin_eng=$1;
       }
      }
      close(SPAS);
      $SCANINFO .= "spamassassin: $spamassassin_eng. ";
    } else {
      #Catch-all for other ones
      $SCANINFO .= "$scanner: ???. ";
    }
  }
  $SCANINFO =~ s/ \. / /g;
  open(VER,">$versionfile.tmp")||die "cannot write to $versionfile.tmp - $!";
  print VER $SCANINFO;
  close(VER);
  rename("$versionfile.tmp","$versionfile");

  &debug("s_q: cleaning up files older than 2 days via $find_binary $scandir/tmp -
mtime +2 -exec $rm_binary -rf {} \;");
  &minidebug("s_q: cleaning up files older than 2 days via $find_binary $scandir/tmp
-mtime +2 -exec $rm_binary -rf {} \;");
  my ($OLDFILES)=`$find_binary $scandir/tmp -mtime +2 -exec $rm_binary -rf {} \\;
2>/dev/null`;
}

sub write_quarantine_report {
  my ($temp,$desc,$report,$subj);
  $subj=$headers{'subject'};
  $subj =~ s/\t/ /g;
  $desc=$quarantine_description;
  $desc =~ s/\n\t/ /g;
  $nowtime = strftime("%a, %d %b %Y %H:%M:%S %Z", localtime(time));
  $report = "$nowtime\t$returnpath\t$recips\t$subj\t$desc\t$SCANINFO\n";
  open(QUARANTINELOG,">>$scandir/$quarantinelog");
  print QUARANTINELOG $report;
  close QUARANTINELOG;
  &debug("w_v_r: writing quarantine log report of: $report");
}

sub scanner_info {
  open(SC,"<$versionfile")||&error_condition("cannot open $versionfile - did you
initialise the system by running \"$prog -z\"? - $!");
  $SCANINFO = <SC>;
  $SCANINFO =~ s/\n|\r|\0/ /g;
  close(SC);
}

sub generate_quarantine_db {
  use DB_File;
  use vars qw( %h);
  my ($line,%array,$count,$match,$type,$descr,$entry,$descrip,$size);
  if ($opt_g) {
    print "perlscanner: generate new DB file from $db_filename.txt\n";
    unlink("$db_filename.db.tmp");
    tie (%array, 'DB_File', "$db_filename.db.tmp", O_CREAT|O_RDWR, 0640, $DB_HASH )
|| &error_condition("cannot open for write $db_filename.db.tmp - $!");

    open(TXT,"<$db_filename.txt")||&error_condition("cannot read $db_filename.txt -
$!");

    #Remeber: all filenames are lowercased, but headers aren't...
    while (<TXT>) {
      $line++;
      next if (/^\#|^\s+$/); #ignore lines starting with hashes
      chomp;
      $count++;
```

```
      ($match,$type,$descr)=split(/\t+/,$_,3);
      if ( $match eq "" || ($type !~ /^[0-9]+$/ && $type !~ /^Virus-[0-9a-z\_\-
]+:$/i) ) {
       print "ERROR: incorrect format on line \"$line\"\n";
       &error_condition("ERROR: incorrect format on line \"$line\"");
      } else {
       #Strip off any regex endings
       if ($type =~ /^[0-9]+$/) {
         #this is a filename/attachment
         if ( $match =~ /^\.dat$/i ) {

           print "ERROR: on line \"$line\".\nCannot block all .dat files. Will block
too many normal messages.\n";
           &error_condition("ERROR: on line \"$line\".\nCannot block all .dat files.
Will block too many normal messages.");
           next;
         }
         $match = tolower($match);
       } else {
         #this is for header matches
         $match =~ s/^\^|\$$//g;
         #Now make unique
         $match = "$line:$match";
         $type =~ s/:$//;
         $type =~ /^Virus\-(.*)/;
         if ($1 !~ /^(MAILFROM|RCPTTO|TCPREMOTEIP)$/) {
           $type="Virus-".tolower($1);
         }
       }
       $array{"$match"}="$type\t$descr";
      }
    }
    close(TXT);
#    $array->sync;
    untie %array ;
    rename("$db_filename.db.tmp","$db_filename.db");
    print "perlscanner: total of $count entries.\n";
  } else {
    print "perlscanner: reading from $db_filename.db\n";
    tie (%array, 'DB_File', "$db_filename.db", O_RDONLY, 0600) ||
&error_condition("cannot open $db_filename.db - $!");
    foreach $entry (keys %array) {
      $count++;
      ($type,$descrip)=split(/\t/,$array{$entry},2);
      if ( $type =~ /^([0-9]+|Any)/) {
       if ($type eq "0") {
         $type="Any";
       } elsif ($size =~ /^[0-9]+$/) {
         $type="$type bytes";
       }
       print "File: \t$entry\n\t\t\tSize: $type\n\t\t\tDescription: $descrip\n\n";
      }
      if ($type =~ /^Virus-(.*)$/i) {
       $type=$1;
       #Strip off numeric uid...
       $entry =~ s/^[0-9]+://;
       if ($type =~ /^(MAILFROM|RCPTTO|TCPREMOTEIP)$/) {
         print "Envelope Header: \t$type\n\t\t\tContent:
^$entry\$\n\t\t\tDescription: $descrip\n\n";
       } else {
         print "Email Header: \t$type\n\t\t\tContent: ^$entry\$\n\t\t\tDescription:
$descrip\n\n";
       }
      }
    }
    untie %array;
    print "perlscanner: total of $count entries found.\n";
  }
}



sub show_version {
```

```
  my ($scanner);
  &scanner_info;
  print "

$prog

Version: $VERSION

Perl:    Summary of my perl5 (revision 5.0 version 8 subversion 0) configuration:

Scanners: perlscanner";
  foreach $scanner (@scanner_array) {
    print ", $scanner";
  }

  print "\n\nScanner versioning: $SCANINFO\n";
  print "
Operating System: Linux, 2.4.20-30.9
Hardware:         i686";
  print "\n\n\n";
}


sub email_sender {
  #Don't e-mail bounced mail messages/etc!
  return if (&is_unreplyable_email('sender'));
  my($addr_type)=@_;
  my ($HDR,$hdr,$tmpsndrs,$tmpsubj,$polstring)='';
  my ($tmpmsgid)= &uniq_id() . "-" . $V_FROM;
  $polstring='policy' if (&notify_addr('nmlvadm'));

  open(SM,"|$qmailinject -h -f ''")||&error_condition("cannot open $qmailinject for
sending quarantine report - $!");
  print SM "From: \"$V_FROMNAME\" <$V_FROM>\n";
  if ($addr_type =~ /sender/) {
    $addr_type='psender' if ($NOTIFY_ADDRS =~ /psender/);
    if ($addr_type eq "sender") {
      if (!&is_unreplyable_email('sender') && &notify_addr('sender')) {
       &debug("e_s: sending quarantine report via: $qmailinject to sender address
($returnpath)");
       print SM "To: $returnpath\n";
       $tmpsndrs = "$returnpath";
      } else {
       &debug("e_s: don't notify sender");
      }
    }elsif ($addr_type eq "psender") {
      if (!&is_unreplyable_email('sender') && &notify_addr('sender') &&
($quarantine_event =~ /^(policy|perlscan)/i && $quarantine_event !~ /virus/i)) {
       &debug("e_s: sending policy quarantine report via: $qmailinject to psender
address ($returnpath)");
       &minidebug("e_s: sending policy quarantine report via: $qmailinject to psender
address ($returnpath)");
       print SM "To: $returnpath\n";
       $tmpsndrs = "$returnpath";
      } else {
       &debug("e_s: don't notify psender");
      }
    } else {
      return;
    }
  } else {
    if (&notify_addr('admin') || (&notify_addr('nmladm') &&
!&is_unreplyable_email('sender')) || (&notify_addr('nmlvadm') && ($quarantine_event
=~ /^(policy|perlscan)/i && $quarantine_event !~ /virus/i) &&
!&is_unreplyable_email('sender'))) {
      &debug("e_s: sending $polstring quarantine report via: $qmailinject to admin
address ($QUARANTINE_CC)");
      print SM "To: $QUARANTINE_CC\n";
      $tmpsndrs .= "$QUARANTINE_CC";
    } else {
      &debug("e_s: don't notify admin");
    }
  }
  $tmpsubj="$destring found in sent message \"$headers{'subject'}\"";
```

```
  $tmpsubj =~ s/(\r|\0|\n)/ /g;
  print SM "Subject: $tmpsubj\n";
  print SM "Message-ID: <".&uniq_id."\@$hostname>\n";
  print SM "X-Tnz-Problem-Type: 40\n";
  print SM "MIME-Version: 1.0\n";
  print SM "Content-type: text/plain\n";
  if ( $descriptive_hdrs ) {
    print SM "${V_HEADER}-Mail-From: $returnpath via $hostname\n";
    print SM "${V_HEADER}-Rcpt-To: $recips\n" if ($descriptive_hdrs eq "2");
    print SM "$V_HEADER: $VERSION ($SCANINFO $destring Found. \n";
    print SM " Processed in ",tv_interval($start_time,[gettimeofday])," secs)\n";
  }
  print SM "\n";
  if (&is_unreplyable_email('sender')) {
    print SM "
Attention: $V_FROMNAME.\n";
    print SM "
[This warning message is *not* being sent to the apparent originator
of the original message.  This address appears to be that of a
mailing list or other automated email system.]\n";
    print SM "\n----------------------------------------\n\n";
  } else {
    print SM "
Attention: $returnpath\n";
  }
  print SM "\n
A $destring was found in an Email message you sent.
This Email scanner intercepted it and stopped the entire message
reaching its destination.

The $destring was reported to be:

$quarantine_description\n";
  if (($addr_type ne "sender") && ($quarantine_description =~ /spam/i) && $sa_report)
{
    print SM "\nSA_REPORT hits = $sa_score/$required_hits\n$sa_report\n\n";
  }
  if ($destring eq "virus") {
    print SM "\n
Please update your virus scanner or contact your IT support
personnel as soon as possible as you may have a virus on your system.\n";
  } else {
    print SM "\n
Please contact your IT support personnel with any queries regarding this
policy.\n";
  }
  print SM "\n
Your message was sent with the following envelope:

MAIL FROM: $returnpath
RCPT TO:   $recips

... and with the following headers:\n\n";
  print SM "---\n";
  print SM "MAILFROM: $headers{'MAILFROM'}\n";
  print SM "$HEADERS\n";
  print SM "---\n";
  if ($addr_type ne "sender" ) {
    print SM "\n

The original message is kept in:

  $hostname:$scandir/$vmaildir/new/$file_id

where the $V_FROMNAME can further diagnose it.

The Email scanner reported the following when it scanned that message:

---
$description
---\n";
  }
  close(SM);
  if ($log_details) {
```

```
    &log_msg("qmail-
scanner","Clear:$tag_score",$elapsed_time,1100,$V_FROM,$tmpsndrs,$tmpsubj,$tmpmsgid,"
quarantine-event.txt:1000");
  }
}

sub email_recips {
  my($recip)=@_;
  return if (!&notify_addr('recips') || $recip eq "");
  my($HDR,$hdr,$tmprecips,$tmpsubj)='';
  my($tmpmsgid)= &uniq_id() . "-" . $V_FROM;

  open(SM,"|$qmailinject -h -f ''")||&error_condition("cannot open $qmailinject for
sending quarantine report - $!");
  print SM "From: \"$V_FROMNAME\" <$V_FROM>\n";
  if (!&is_unreplyable_email('recips')) {
    &debug("e_r: sending quarantine report via: $qmailinject to recip address
($recip)");
    print SM "To: $recip\n";
  }
  $tmpsubj= "$destring found in received message \"$headers{'subject'}\"";
  $tmpsubj =~ s/(\r|\0|\n)/ /g;
  print SM "Subject: $tmpsubj\n";
  print SM "Message-ID: <".&uniq_id."\@$hostname>\n";
  print SM "X-Tnz-Problem-Type: 40\n";
  print SM "MIME-Version: 1.0\n";
  print SM "Content-type: text/plain\n";
  if ( $descriptive_hdrs ) {
    print SM "${V_HEADER}-Mail-From: $returnpath via $hostname\n";
    print SM "${V_HEADER}-Rcpt-To: $recip\n" if ($descriptive_hdrs eq "2");
    print SM "$V_HEADER: $VERSION ($SCANINFO $destring Found. \n";
    print SM " Processed in ",tv_interval($start_time,[gettimeofday])," secs)\n";
  }
  print SM "\n";
  print SM "
Attention: $recip\n";
  if (!&is_unreplyable_email('recips')) {
    if (&notify_addr('sender')) {
      print SM "
[A message has been sent to the originator, stating there is a virus
in the Email they just sent to you. No further action is required on
your part.]\n";
    }
  } else {
    print SM "

[This message was _not_ sent to the originator address, as that appears to
be a mailing-list or some other automated Email message]\n";
  }
  print SM "\nA $destring was found in an Email message sent to you.
This Email scanner intercepted it and stopped the entire message
before it reached you. No further action is required on your part.\n";
  print SM "\nThe $destring was reported to be:

$quarantine_description

Please contact your IT support personnel with any queries regarding this
policy.

The message sent to you had the following envelope:

MAIL FROM: $returnpath
RCPT TO:   $recips

... and with the following headers:\n\n";
  print SM "---\n";
  print SM "MAILFROM: $headers{'MAILFROM'}\n";
  print SM "$HEADERS\n";
  print SM "---\n";
  #print SM "\nLxOCALE_recips_quarantine\n";
  close(SM);
  if ($log_details) {
```

```
    &log_msg("qmail-
scanner","Clear:$tag_score",$elapsed_time,1100,$V_FROM,$recip,$tmpsubj,$tmpmsgid,"qua
rantine-event.txt:1000");
  }
}

sub notify_addr {
  my($addr_type)=@_;
  #&debug("n_a: notify_addr (set to $NOTIFY_ADDRS) called with $addr_type");
  if (($NOTIFY_ADDRS =~ /$addr_type/ || $NOTIFY_ADDRS =~ /all/) && ($NOTIFY_ADDRS !~
/none/)) {
    return 1;
  } else {
    return 0;
  }
}

sub unzip_file {
  my($zipfile)=@_;
  my ($MAYBEZIP,$ztmp,$zfile,$zline,$zsize,$zip_status,$passwd_protected_zip);

  &debug("u_f: potential zip archive file found ($zipfile).");
  &debug ("u_f: it is possibly a zip file, run unzip $unzip_options -t
$ENV{'TMPDIR'}/$zipfile");
  $MAYBEZIP=`$unzip_binary $unzip_options -t $ENV{'TMPDIR'}/$zipfile 2>&1`;
  $zip_status=($? >> 8);

  if ( ($zip_status > 0) && ($zip_status !~ /^(1|2|3|51|81|82)$/) && ($MAYBEZIP !~
/skipping: /) ) {
    &debug("u_f: not a recognisable zip file ($MAYBEZIP)");
  } else {
    if ($MAYBEZIP =~ /skipping:.*password/) {
      &debug ("u_f: it is a password-protected zip file");
      &minidebug ("u_f: it is a password-protected zip file");
      $passwd_protected_zip++;
      $CRYPTO_TYPE="CR:ZIP(encrypted)" if ($CRYPTO_TYPE eq "");
    } else {
      &debug ("u_f: it is a zip file - unpack it!");
    }
    if ($BLOCK_PASSWORD_PROTECTED_ARCHIVES && $passwd_protected_zip) {
      #Quarantine it!
      $quarantine_description="Disallowed password-protected zip files ($zipfile) -
potential virus";
      &debug("u_f: $quarantine_description");
      &minidebug("u_f: $quarantine_description");
      $destring='problem';
      $quarantine_event="Policy:Passwd_ZIP";
      $description .= "\n---perlscanner results ---\n$destring
'$quarantine_description' found in file $ENV{'TMPDIR'}/$zipfile";
    } else {
      if ($force_unzip) {
       &debug("u_f: run $unzip_binary $unzip_options -oj $ENV{'TMPDIR'}/$zipfile
2>&1");
       open(ZIPPED,"$unzip_binary $unzip_options -oj  $ENV{'TMPDIR'}/$zipfile
2>&1|")||&error_condition("u_f: cannot open $ENV{'TMPDIR'}/$zipfile - $!");
       while (<ZIPPED>) {
         if (/^\s+\w+:\s+(.*)$/) {
           ($ztmp=$1)=~s/^.*\///g;
           #Grrr, I don't know if this'll be exploited, but I have to remove the
whitespace...
           #$ztmp=~s/\s+$//g;
           #if ($ztmp ne "" && !grep(/^${ztmp}$/,@zipfile_list)) {
             #&debug("u_f: adding file \"$ztmp\" to list of zipped files");
             #push(@zipfile_list, $ztmp);
           #}
         }
         if (/^\s+skipping:\s(.*)\s+(shrink|encrypted|incorrect password)/) {
           $passwd_protected_zip++ if (!/^\s+skipping:\s(.*)\s+shrink/);
           #grab these protected filenames for reports anyway.
           $zfile = $1;
           $zfile =~ s/^.*\///g;
           $zfile =~ s/(^\s+|\s+$)//g;
           #$file_desc .= "$zfile:$zsize\t";
         }
```

```
        }
        close(ZIPPED);
        $zip_status=($? >> 8);
        if ($zip_status > 0 && ($zip_status !~ /^(1|2|3|51|81|82)$/ &&
!$passwd_protected_zip)) {
          &error_condition("u_f: cannot close unzip (error code:
$zip_status,$passwd_protected_zip) - $!");
        }
      }
    }
    #Only delete original zip file if it happily unpacked.
    if ( $zip_status eq 0 && -f "$ENV{'TMPDIR'}/$zipfile") {
       #system $rm_binary,"-f","$ENV{'TMPDIR'}/$zipfile";
       &debug("u_f: $zip_status, and successfully unzipped");
       #It may have been deleted, but you still want to see if
       #it matches the perlscanner DB...
       #$zipfile=tolower($zipfile);
       #push(@zipfile_list, $zipfile) if (!grep(/^$zipfile$/,@zipfile_list));
       my
($dev,$ino,$mode,$nlink,$uid,$gid,$rdev,$zsize,$atime,$mtime,$ctime,$blksize,$blocks)
;

($dev,$ino,$mode,$nlink,$uid,$gid,$rdev,$zsize,$atime,$mtime,$ctime,$blksize,$blocks)
= stat("$zipfile");
       $file_desc .= "$zipfile:$zsize\t";
    }
  }
}

sub deltatime {
  my ($delta,$current_time,$last_time);
  $current_time = [gettimeofday];
  $delta =  tv_interval ($last_time, $current_time);
  $last_time=$current_time;
  return $delta;
}

sub qmail_parent_check {
  my $ppid=getppid;
  #&debug("q_s_c: PPID=$ppid");
  if ($ppid == 1)  {
    &debug("q_s_c: Whoa! parent process is dead! (ppid=$ppid) Better die too...");
    &minidebug("q_s_c: Whoa! parent process is dead! (ppid=$ppid) Better die
too...");
    &cleanup;
    &close_log;
    #Exit with temp error anyway - just to be real anal...
    exit 111;
  }
}

sub check_and_grab_uuencoding {};

sub log_msg() {};


################################
##
##  END of standard subroutines
##  Virus-scanner specific subroutines automatically added below by setup.sh
##
################################

################################################
# Subroutines added by ST
################################################

sub minidebug {
  my $dnowtime = strftime("%a, %d %b %Y %H:%M:%S %Z", localtime(time));
  print LOG "$dnowtime:$nprocess: ",@_,"\n" if ($MINIDEBUG && !$DEBUG);
}

sub close_log {
```

```
  ($sec,$min,$hour,$mday,$mon,$year) = localtime(time);
  # Format: dd/mm/yyyy hh:mm:ss....
  #$nowtime = sprintf "%02d/%02d/%02d %02d:%02d:%02d", $mday, $mon+1, $year+1900,
$hour, $min, $sec;
  # Format: day, dd month yyyy hh:mm:ss +TZ...
  #$nowtime = strftime("%a, %d %b %Y %H:%M:%S %z", localtime(time));

  &debug("--- all finished. Total of ",tv_interval ($start_time, [gettimeofday])," 
secs");
  &minidebug("------ Process $nprocess finished. Total of ",tv_interval ($start_time,
[gettimeofday])," secs");
  close(LOG);
}

sub reject_email {
  my ($exit_string,$exit_code)=@_;
  $exit_code=111 if (!$exit_code);

  # st: tell qmail-smtpd why the message is rejected,
  # so it can be written to the qmail-smtpd log
  warn "$V_HEADER-$VERSION: $exit_string\n";

  &debug("r_e: $V_HEADER-$VERSION: $exit_string");
  &minidebug("r_e: $V_HEADER-$VERSION: $exit_string");

  &cleanup;

  &close_log;
  exit $exit_code;
}

################################################
# END of subroutines added by ST
################################################

sub clamdscan_scanner {
  #Clamdscan scanner
  &debug("clamdscan: starting scan of directory \"$ENV{'TMPDIR'}\"...");

  my ($start_clamdscan_time)=[gettimeofday];
  my ($DD,$clamdscan_status,$stop_clamdscan_time,$clamdscan_time);
  my ($clamdscan_verbose,$clamdscan_status);
  $clamdscan_verbose="-v" if ($DEBUG);

  &debug("run $clamdscan_binary $clamdscan_options  $ENV{'TMPDIR'} 2>&1");

  $DD=`$clamdscan_binary $clamdscan_options $ENV{'TMPDIR'} 2>&1`;
  $clamdscan_status=($? >> 8);

  &debug("--output of clamdscan was:\n$DD--");

  if ( $clamdscan_status > 0 ) {
    if ($clamdscan_status == 1 && $DD =~ /\:\s(.*)\sFOUND$/m) {
      $quarantine_description=$+;
      &debug("There be a virus! ($quarantine_description)");
      &minidebug("clamdscan: there be a virus! ($quarantine_description)");
      ($quarantine_event=$quarantine_description)=~s/\s/_/g;
      $quarantine_event="CLAMDSCAN:".substr($quarantine_event,0,$QE_LEN);
      $description .= "\n---clamdscan results ---\n$DD";
    } else {
      #This implies a corrupt set of DAT files or resource problems...
      &error_condition("clamdscan: corrupt or unknown clamd scanner error or
memory/resource/perms problem - exit status $clamdscan_status");
    }
  } else {
    if ($DD =~ /Recursion limit exceeded/) {
      $quarantine_description="Resource attack - $1";
      &debug("clamdscan: $quarantine_description");
      &minidebug("clamdscan: $quarantine_description");
      $quarantine_event="CLAMDSCAN:Resource_attack";
      $description .= "\n---clamdscan results ---\n$DD";
    }
  }
```

```
  #Bugs in clamdscan sometimes shows up as zero output. Always error on such
2645 conditions
  $DD=~s/\n//g;
  if ($DD eq "" ) {
    &error_condition("clamdscan: corrupt or unknown clamd scanner error or
memory/resource/perms problem - exit status $clamdscan_status, but no output!");
2650 }

  $stop_clamdscan_time=[gettimeofday];
  $clamdscan_time = tv_interval ($start_clamdscan_time, $stop_clamdscan_time);
  &debug("clamdscan: finished scan of dir \"$ENV{'TMPDIR'}\" in $clamdscan_time
2655 secs");
  &minidebug("clamdscan: finished scan of dir \"$ENV{'TMPDIR'}\" in $clamdscan_time
secs");
}
sub spamassassin {
2660 #Only run SA if mail is from a "remote" SMTP client, or QS_SPAMASSASSIN
  #is defined via tcpserver...
  if (defined($ENV{'RELAYCLIENT'}) && !defined($ENV{'QS_SPAMASSASSIN'})) {
    &debug("spamassassin: don't scan as RELAYCLIENT implies this was sent by a local
user");
2665   &minidebug("SA: don't scan as RELAYCLIENT implies this was sent by a local
user");
    return;
  }
  if ( $SA_SKIP_MD ne "0" && $returnpath eq "" && $headers{'from'} =~ /mailer-
2670 daemon|postmaster|bounce/i ) {
  &debug("SA: skipping message from MAILER-DAEMON");
  &minidebug("SA: skipping message from MAILER-DAEMON");
  return;
  }
2675
  #SpamAssassin client scanner
  my ($start_spamassassin_time)=[gettimeofday];
  my
($sa_tag,$spamassassin_status,$stop_spamassassin_time,$spamassassin_time,$cmdline_rec
2680 ip,$sa_fast);
  my ($sa_status)=0;
  ($sa_score,$required_hits)=('0','0');

  #Cleanup $one_recip so it's usable from the commandline...
2685 #any char that isn't supported to changed into an '_'
  ($cmdline_recip=$one_recip)=~s/[^0-9a-z\.\_\-\=\+\@]/_/gi;
  $cmdline_recip=~/^([0-9a-z\.\_\-\=\+\@]+)$/i;
  $cmdline_recip=tolower($1);

2690 # st: a dirty way of cleaning up the chars that can be interpreted as an option by
spamc
  # Problem pointed by Jonas Thomsen.

  if ($cmdline_recip=~/\-[crydefhpstux]/) {
2695   &debug("SA: option char in $cmdline_recip don't pass it to spamc");
    &minidebug("SA: option char in $cmdline_recip don't pass it to spamc");
    $cmdline_recip="";
  }

2700 $sa_fast=1 if ($spamc_options =~ / \-c /);
  $spamc_options="$spamc_options -u \"$cmdline_recip\"" if ($cmdline_recip ne "");
  &debug("SA: run $spamc_binary $spamc_options < $scandir/$wmaildir/new/$file_id");
  open(SA,"$spamc_binary $spamc_options  <
$scandir/$wmaildir/new/$file_id|")||&error_condition("cannot run $spamc_binary  <
2705 $scandir/$wmaildir/new/$file_id - $!");
  open(SOUT,">$scandir/$wmaildir/new/$file_id.spamc")||&error_condition("cannot open
for write $scandir/$wmaildir/new/$file_id.spamc - $!");
  while (<SA>) {
    if ($sa_fast) {
2710   chomp;
      ($sa_score,$required_hits)=split(/\//,$_,2);
      $sa_tag++;
      last;
    } else {
2715   #X-Spam-Status: No, hits=2.8 required=5.0
      if (/^X-Spam-Status: (Yes|No), hits=(-?[\d\.]*) required=([\d\.]*)/) {
        $sa_tag++;
```

```
        $sa_status=1 if ($1 eq "Yes");
        $sa_score=$2;$required_hits=$3;
      }
    }
    print SOUT;
  }
  close SA ;
  $spamassassin_status=($? >> 8);
  $sa_status=$spamassassin_status if ($sa_fast);
  close(SOUT);

  $sa_score='?' if (!$sa_score);
  $required_hits='?' if (!$required_hits);

  if (!$sa_fast && -s "$scandir/$wmaildir/new/$file_id.spamc" && $spamassassin_status
== 0) {
    &debug("SA: overwriting $scandir/$wmaildir/new/$file_id with
$scandir/$wmaildir/new/$file_id.spamc");
    rename
("$scandir/$wmaildir/new/$file_id.spamc","$scandir/$wmaildir/new/$file_id");
  } else {
    unlink("$scandir/$wmaildir/new/$file_id.spamc");
  }
  if ($required_hits > $sa_score || ($sa_score == 0) || ($sa_score eq "\?")) {
    $tag_score .= "SA:0($sa_score/$required_hits):";
    $sa_comment = "No, hits=$sa_score required=$required_hits";
  } else {
    # If sa_quarantine/sa_delete are set, then compare them to the current score and
    # quarantine/delete it if necessary,
    # otherwise tag the message as spam.

    # Control the values of sa_delete and sa_quarantine
    if ($sa_delete && ($sa_quarantine>$sa_delete)) {
      &debug("SA: WARNING, sa_delete is lower than sa_quarantine, spam could be
quarantined, but not deleted");
      &minidebug("SA: WARNING, sa_delete is lower than sa_quarantine, spam could be
quarantined, but not deleted");
      $sa_delete='0';
    }

    if ( $sa_delete && (($sa_delete+$required_hits)<$sa_score)) {
       if ( $sa_reject) {
          &debug("SA: yup, this smells like SPAM - hits=$sa_score - rejecting
message...");
          &minidebug("SA: yup, this smells like SPAM - hits=$sa_score - rejecting
message...");
          &reject_email("We have reasons to believe this mail is SPAM",31);
       } else {
          # st: mark the message to delete it
          $del_message='1';
          # st: maybe these three lines are useful for those who wants the
'log_details'...
          # But if the message is rejected nothing remains
          $destring="problem";
          $quarantine_description="SPAM exceeds \"delete\" threshold -
hits=$sa_score" if (!$quarantine_description);
          $quarantine_event="SA:SPAM-DELETE";
          &debug("SA: yup, this smells like SPAM - deleting message...");
          &minidebug("SA: yup, this smells like SPAM - deleting message...");
          $description .= "\n---spamassassin results ---\n$destring
'$quarantine_description'\n found in message $ENV{'TMPDIR'}";
       }
    } else {
       if ( $sa_quarantine && (($sa_quarantine+$required_hits)<$sa_score)) {
          $destring="problem";
          $quarantine_description="SPAM exceeds \"quarantine\" threshold -
hits=$sa_score" if (!$quarantine_description);
          $quarantine_event="SA:SPAM-QUARANTINE";
          &debug("SA: yup, this smells like SPAM - quarantine message...");
          &minidebug("SA: yup, this smells like SPAM - quarantine message...");
          $description .= "\n---spamassassin results ---\n$destring
'$quarantine_description'\n found in message $ENV{'TMPDIR'}";
       } else {
          $tag_score .= "SA:1($sa_score/$required_hits):";
```

```
          $sa_comment = "Yes, hits=$sa_score required=$required_hits" if ($sa_fast);

          #st: if $spamc_subjec and $sa_delta are set, add in the subject the spam-
2795 level
          if ($spamc_subject ne "" && $sa_delta) {
             if ($sa_score < ($required_hits+$sa_delta)) {
                $spamc_subject .= " LOW * ";
             } elsif ($sa_score > ($required_hits+(2 * $sa_delta))) {
2800            $spamc_subject .= " HIGH * ";
             } else {
                $spamc_subject .= " MEDIUM * ";
             }
          }
2805      &debug("SA: yup, this smells like SPAM");
          &minidebug("SA: yup, this smells like SPAM");
       }
    }
  }
2810  my $sa_hits=$sa_score;
  if ($sa_score > 0) {
    $sa_score=int($sa_score);
    #Keep it RFC compliant
    $sa_score=100 if ($sa_score > 100);
2815    my $si=0;
    if ($sa_fast) {
      while ($si < $sa_score) {
        $si++;
        $sa_level .= $sa_symbol;
2820    }
    }
  }
  $stop_spamassassin_time=[gettimeofday];
  $spamassassin_time = tv_interval ($start_spamassassin_time,
2825 $stop_spamassassin_time);

  &debug("SA: required_hits $required_hits / sa_quarantine +$sa_quarantine /
sa_delete +$sa_delete") if ($sa_quarantine || $sa_delete);
  &minidebug("SA: required_hits $required_hits / sa_quarantine +$sa_quarantine /
2830 sa_delete +$sa_delete") if ($sa_quarantine || $sa_delete);

  &debug("SA: finished scan of dir \"$ENV{'TMPDIR'}\" in $spamassassin_time secs");
  &minidebug("SA: finished scan in $spamassassin_time secs - hits=$sa_hits");
}
2835

################################################
# Spamassassin subroutine added by ST
################################################

2840 sub spamassassin_alt {
  # st: Alternative routine for spamassassin, lighter and can logs the report...

  #Only run SA if mail is from a "remote" SMTP client, or QS_SPAMASSASSIN
  #is defined via tcpserver...
2845  if (defined($ENV{'RELAYCLIENT'}) && !defined($ENV{'QS_SPAMASSASSIN'})) {
    &debug("spamassassin: don't scan as RELAYCLIENT implies this was sent by a local
user");
    &minidebug("SA: don't scan as RELAYCLIENT implies this was sent by a local
user");
2850    return;
  }
  if ( $SA_SKIP_MD ne "0" && $returnpath eq "" && $headers{'from'} =~ /mailer-
daemon|postmaster|bounce/i ) {
  &debug("SA: skipping message from MAILER-DAEMON");
2855  &minidebug("SA: skipping message from MAILER-DAEMON");
  return;
  }

  #SpamAssassin client scanner
2860  my ($start_spamassassin_time)=[gettimeofday];
  my ($sa_tag,$spamassassin_status,$stop_spamassassin_time,$spamassassin_time);
  my ($sa_status)=0;
  ($sa_score,$required_hits)=('0','0');
  $sa_report='';
2865
```

```
  if ( $sa_debug eq "1" ) {
      $spamc_options=" -R -f ";
  } else {
      $spamc_options=" -c -f ";
  }

  open(SA,"$spamc_binary $spamc_options  <
$scandir/$wmaildir/new/$file_id|")||&error_condition("cannot run $spamc_binary  <
$scandir/$wmaildir/new/$file_id - $!");
  while (<SA>) {
    if (!$sa_tag) {
       chomp;
       ($sa_score,$required_hits)=split(/\//,$_,2);
       $sa_tag++;
       next;
    }
    if ( /^(\s+|-)\d.*$/ ) {
       $sa_report .= $_ ;
    }
  }

  $spamassassin_status=($? >> 8);
  $sa_status=$spamassassin_status if ($spamc_options =~ /\-c/);

  close SA ;

  $sa_score='?' if (!$sa_score);
  $required_hits='?' if (!$required_hits);

  &debug("SA: REPORT hits = $sa_score/$required_hits\n$sa_report") if ( $sa_debug eq
"1" );
  &minidebug("SA: REPORT hits = $sa_score/$required_hits\n$sa_report") if ( $sa_debug
eq "1" );

  if ($required_hits > $sa_score || ($sa_score == 0) || ($sa_score eq "\?")) {
    $tag_score .= "SA:0($sa_score/$required_hits):";
    $sa_comment = "No, hits=$sa_score required=$required_hits";
  } else {
    # If sa_quarantine/sa_delete are set, then compare them to the current score and
    # quarantine/delete it if necessary,
    # otherwise tag the message as spam.

    # Control the values of sa_delete and sa_quarantine
    if ($sa_delete && ($sa_quarantine>$sa_delete)) {
      &debug("SA: WARNING, sa_delete is lower than sa_quarantine, spam could be
quarantined, but not deleted");
      &minidebug("SA: WARNING, sa_delete is lower than sa_quarantine, spam could be
quarantined, but not deleted");
      $sa_delete='0';
    }

    if ( $sa_delete && (($sa_delete+$required_hits)<$sa_score)) {
       if ( $sa_reject) {
          &debug("SA: yup, this smells like SPAM - hits=$sa_score - rejecting
message...");
          &minidebug("SA: yup, this smells like SPAM - hits=$sa_score - rejecting
message...");
          &reject_email("We have reasons to believe this mail is SPAM",31);
       } else {
          # st: mark the message to delete it
          $del_message='1';
          # st: maybe these three lines are useful for those who wants the
'log_details'...
          # But if the message is rejected nothing remains
          $destring="problem";
          $quarantine_description="SPAM exceeds \"delete\" threshold -
hits=$sa_score" if (!$quarantine_description);
          $quarantine_event="SA:SPAM-DELETE";
          &debug("SA: yup, this smells like SPAM - deleting message...");
          &minidebug("SA: yup, this smells like SPAM - deleting message...");
          $description .= "\n---spamassassin results ---\n$destring
'$quarantine_description'\n found in message $ENV{'TMPDIR'}";
       }
    } else {
```

```
       if ( $sa_quarantine && (($sa_quarantine+$required_hits)<$sa_score)) {
          $destring="problem";
          $quarantine_description="SPAM exceeds \"quarantine\" threshold -
hits=$sa_score" if (!$quarantine_description);
          $quarantine_event="SA:SPAM-QUARANTINE";
          &debug("SA: yup, this smells like SPAM - quarantine message...");
          &minidebug("SA: yup, this smells like SPAM - quarantine message...");
          $description .= "\n---spamassassin results ---\n$destring
'$quarantine_description'\n found in message $ENV{'TMPDIR'}";
       } else {
          $tag_score .= "SA:1($sa_score/$required_hits):";
          $sa_comment = "Yes, hits=$sa_score required=$required_hits";
          #
          if ($spamc_subject ne "" && $sa_delta) {
             if ($sa_score < ($required_hits+$sa_delta)) {
                $spamc_subject .= " LOW * ";
             } elsif ($sa_score > ($required_hits+(2 * $sa_delta))) {
                $spamc_subject .= " HIGH * ";
             } else {
                $spamc_subject .= " MEDIUM * ";
             }
          }
          &debug("SA: yup, this smells like SPAM");
          &minidebug("SA: yup, this smells like SPAM");
       }
    }
  }
  my $sa_hits=$sa_score;
  if ($sa_score > 0) {
    $sa_score=int($sa_score);
    #Keep it RFC compliant
    $sa_score=100 if ($sa_score > 100);
    my $si=0;
    while ($si < $sa_score) {
      $si++;
      $sa_level .= $sa_symbol;
    }
  }
  $stop_spamassassin_time=[gettimeofday];
  $spamassassin_time = tv_interval ($start_spamassassin_time,
$stop_spamassassin_time);

  &debug("SA: required_hits $required_hits / sa_quarantine +$sa_quarantine /
sa_delete +$sa_delete") if ($sa_quarantine || $sa_delete);
  &minidebug("SA: required_hits $required_hits / sa_quarantine +$sa_quarantine /
sa_delete +$sa_delete") if ($sa_quarantine || $sa_delete);

  &debug("SA: finished scan of dir \"$ENV{'TMPDIR'}\" in $spamassassin_time secs");
  &minidebug("SA: finished scan in $spamassassin_time secs - hits=$sa_hits");
}

################################################
# END of Spamassassin subroutine added by ST
################################################
#########################
## END of scanner definitions
##
#########################
```