CENTRO UNIVERSITÁRIO DE BRASÍLIA - UNICEUB

TENILLE ALMEIDA DE MORAES

ICAP COMO SOLUÇÃO PARA ADAPTAÇÃO DE CONTEÚDO EM VERIFICAÇÃO DE MALWARE

Brasília - DF

TENILLE ALMEIDA DE MORAES

ICAP COMO SOLUÇÃO PARA ADAPTAÇÃO DE CONTEÚDO EM VERIFICAÇÃO DE MALWARE

Trabalho apresentado à banca examinadora da Faculdade de Ciências Exatas e Tecnológicas, para conclusão do curso de Engenharia da Computação.
Professor Orientador: Fabiano Mariath.

Brasília - DF

TENILLE ALMEIDA DE MORAES

ICAP COMO SOLUÇÃO PARA ADAPTAÇÃO DE CONTEÚDO EM VERIFICAÇÃO DE MALWARE

COMISSÃO EXAMINADORA

_____

Prof. Fabiano Mariath D' Oliveira

_____

Prof. Antônio Gonçalves J. Pinto

_____

Prof. Miguel

Brasília, 26 de junho de 2006.

“Ao meu namorado Rafael,
que apoiou o meu sucesso profissional com todo o amor e carinho que precisei ao longo dos últimos meses.

Aos meus Pais,
que continuam lutando até hoje em prol do bem-estar dos seus filhos.”

Agradeço a todos que diretamente ou não, contribuíram para o desenvolvimento deste Projeto, e em particular:

Ao Professor Fabiano Mariath, pela orientação, incentivo, cobrança e sugestões de conteúdos que enriqueceram este Projeto.

Aos professores do Curso de Engenharia da Computação do Centro Universitário de Brasília – UniCEUB que são responsáveis por parte do sucesso profissional de todos os seus alunos.

Aos meus pais, Marco Antônio de Moraes e Maria Nazareth Almeida de Moraes, e à minha irmã Etienne Almeida de Moraes que sempre estiveram do meu lado, ajudando, incentivando, confortando e me fortalecendo para que eu conseguisse concluir com êxito este Projeto.

Ao Rafael Gomes, pela compreensão, carinho e apoio diário.

Ao mestre Rômulo Setúbal e ao DSc. Roberto Campos da Rocha Miranda pelo auxílio e esclarescimento de dúvidas.

Aos amigos, em especial, a Priscila Cortez, Wolmer Godoi, Rafael Dohms, Rodolfo Rodrigues e Cid Antinues que sempre estavam dispostos a me auxiliar de alguma forma no desenvolvimento desse Projeto.

SUMÁRIO

RESUMO

Com o constante crescimento da *Internet*, a necessidade de implementação de serviços escalonáveis se tornou imprescindível. Portais de serviços *Web* atendem diariamente centenas de milhares de requisições. A estrutura de servidores monolíticos centralizados é a principal responsável pelo atraso e demora na resposta às requisições dos clientes. Os provedores de conteúdo necessitavam atender de forma mais eficiente a crescente demanda e melhorar a qualidade de seus serviços. Para atender este requisito, várias soluções foram adotadas: instalação de servidores de replicação, servidores de *cluster*, *caches* e balanceamento de carga, são algumas delas. Toda essa infra-estrutura aumentou o custo e a manutenção do ambiente tecnológico. A proposição de um novo protocolo para fazer o controle e a alteração desse conteúdo, de forma dinâmica, veio em boa hora. O ICAP veio atender à demanda dos serviços e servidores de conteúdo. Esse Projeto efetua uma análise aprofundada desse protocolo e sua utilização como agregador de segurança no ambiente de borda de rede.

Palavras-Chave: Protocolo, *Internet*, *malware*, vírus de computador, ICAP, serviços de borda, controle de conteúdo, *cache*, *Proxy* e segurança.

ABSTRACT

The constant Internet growth and the necessity of new services became essential. Web services portals daily answers hundreds of thousand of solicitations. The centered monolithic servers structure is the main responsible for the delay in the customers responses. The content suppliers needed a more efficient way to increasing demand and to improve the quality of its services. To answer this requirement, some solutions had been adopted: replication servers installation, cluster servers, caches and load balancing scheme, is some of them. All this infrastructure increased the cost and the maintenance of the technological environment. The proposal of a new protocol to make the control and the content alteration, in a dynamic form, came just about time. The ICAP came to respond the demand of the services and content servers. This Project does a deepened analysis of this protocol and its use as a security provider in the net edge environment.

Key-Word: Protocol, Internet, malware, virus of computer, ICAP, services of edge, control of content, cache, Proxy and security.

LISTA DE QUADROS

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

API - *Application Programming Interface* ou Interface de Programação de Aplicativos

CD - *Compact Disc* ou Disco Compacto

CERT.br - Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil

CRLF - *Carrier Return Line Feed* ou comando que cria uma nova linha.

DES - *Data Encryption Standard* é um algoritmo de encriptação.

DMZ – *DeMilitarized Zone* ou Zona Desmilitarizada

FBI - *Federal Bureau of Investigation* ou Escritório Federal de Investigação

FTP - *File Transfer Protocol* ou Protocolo de Transferência de Arquivos

GPL - *General Public License* ou Licença Pública Geral

HTML - *Hypertext Markup Language*

HTTP - *Hypertext Transfer Protocol* ou Protocolo de Transferência de Hipertexto

ICAP – *Internet Content Adaption Protocol* ou Protocolo de Adaptação de Conteudo da Internet

ID – *Identification* ou Identificação.

IP - *Internet Protocol* ou Protocolo da Internet

MB – *Megabytes* ou um milhão de *bytes*.

MBR - *Master Boot Record* ou Primeiro Setor de Boot

MS-DOS - *Microsoft Disk Operating System*

P2P - *Peer-to-Peer*

PC - *Personal Computer* ou Computador Pessoal

PDA - *Personal Digital Assistants* ou Assistente Pessoal Digital

RAT – *Remote Access Trojans* ou Trojans de Acesso Remoto

RFC - *Request for Comments*

RPC - *Remote Procedure Call* ou Chamada de Procedimento Remoto

SMTP - *Simple Mail Transfer Protocol*

SQL - *Structured Query Language* ou Linguagem de Consulta Estruturada

TCP - *Transmission Control Protocol*

TI - Tecnologia da Informação e Comunicação

UDP - *User Datagram Protocol* ou Protocolo de Datagramas de Usuário

URL - *Universal Resource Locator* ou Localizador Uniforme de Recursos

USB - *Universal Serial Bus*

WAP - *Wireless Application Protocol* ou Protocolo para Aplicações Sem Fio

XDR - *Extreme Data Rate*

XML - *eXtensible Markup Language*

CAPÍTULO 1. INTRODUÇÃO

1.1 MOTIVAÇÃO

O relatório (*http://www.cert.br/stats/incidentes*) do Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil - CERT.br informa que, dos incidentes reportados durante o período de janeiro a dezembro de 2005, do total de 68.000 incidentes, 44.785 destes foram causados por *Worms* e fraudes. Os dados constantes desse Relatório estão refletidos no artigo intitulado “A Mudança na Natureza do Crime”, realizada pela IBM, na qual altos executivos de Tecnologia da Informação da América Latina afirmam ser o investimento em soluções de antivírus, seguindo a tendência global, uma das iniciativas mais importantes a ser tomada durante o próximo ano.

Atualmente, o termo "vírus de computador" tornou-se genérico para denominar os vários tipos de códigos maliciosos que podem, da mesma forma que um visitante indesejado, causar grandes danos a uma rede coorporativa. E assim como vírus biológicos, os “espécimes” de vírus de computador que podem atacar uma rede são incontáveis e o propósito dos ataques é o de explorar os pontos fracos das várias plataformas de computação, ambientes de rede e "culturas" do usuário.

Só o conhecimento dos principais tipos de vírus, a manutenção de uma política preventiva, o uso de uma ferramenta específica e o constante investimento em treinamento do usuário final podem ajudar os gerentes de TI (Tecnologia da Informação e Comunicação) a proteger seus ambientes corporativos e parques tecnológicos como um todo.

Diante do cenário de constante surgimento de novos tipos de vírus e prevenção, gerentes de TI adotaram como medidas de prevenção a utilização das chamadas ferramentas de antivírus em seus parques tecnológicos, de modo a conter o avanço das ameaças e diminuir o

downtime[1], devido às epidemias relacionadas a vírus.

Um fato importante a ser destacado é que diante de um ambiente interligado e conectado à *Internet*, tornam-se constantes e intensas as ameaças oriundas da rede global. Muitas das infecções, reportadas nos relatórios do CERT.br já mencionado, foram originadas da *Internet*.

Atualmente considera-se como um ambiente corporativo, bem planejado e protegido, aquele que implementa proteção em camadas, ou seja, um ambiente em que um código malicioso, para infectar uma estação de trabalho, precisaria superar as barreiras de vários dispositivos colocados no caminho, como por exemplo: servidores[2], roteadores[3] e *Proxies*.

Este Projeto tem o propósito de estudar e certificar a melhoria dos serviços prestados pelos computadores, com função de servidores, num ambiente corporativo, com a utilização do *Internet Content Adptation Protocol*, ICAP, quando será utilizada uma solução de antivírus, baseada no Sistema Operacional *Linux*, operacionalizada com protocolo ICAP, como solução para redução do consumo de recursos dos servidores. Para a implantação desta solução, utilizamos um servidor *Proxy*, baseado em software livre *Squid*, por meio do protocolo de intercomunicação, ICAP.

1.2 OBJETIVOS

Este Projeto têm como objetivo validar a redução do consumo de recursos de um *Proxy* que esteja utilizando ICAP. Para tanto se efetuará um estudo comparativo - *BENCHMARK*, analisando o desempenho dos serviços providos por um servidor *Proxy* com proteção de antivírus e do mesmo ambiente, utilizando o protocolo ICAP para fazer a proteção contra vírus. Este estudo visa certificar a real redução do nível de utilização do processamento e alocação de memória neste *Proxy*, conforme é proposto

1 *Downtime* é o período de indisponibilidade dos serviços de rede.

2 Servidor é computador que fornece serviços a uma rede de computadores.

3 Roteador é um equipamento usado para fazer a comunicação entre diferentes redes de computadores.

pelos desenvolvedores do protocolo ICAP.

1.3 METODOLOGIA

Capítulo 02: apresentação do referencial teórico que dará sustentação ao desenvolvimento do Projeto. Neste Capítulo será fornecido uma visão geral sobre vírus de computador e alguns outros tipos de *malware*, bem como outros códigos maliciosos e as tecnologias de antivírus. Abordaremos ainda o Protocolo de Adaptação de Conteúdo da Internet (ICAP), focalizando o uso deste Protocolo para serviço de verificação de vírus;

Capítulo 03: serão apresentadas as tecnologias escolhidas, especificações técnicas e ambientes a serem analisados;

Capítulo 04: relato dos processos de montagem, instalação e configuração do ambiente a ser executado o *benchmarking*;

Capítulo 05: serão apresentados os testes e as simulações realizadas, assim como os resultados obtidos e suas análises; e

Capítulo 06: neste Capítulo será apresentado as conclusões, dificuldades enfrentadas e as perspectivas de evolução do Projeto.

CAPÍTULO 2. REFERENCIAL TEÓRICO

2.1 APRESENTAÇÃO

Neste Capítulo será abordado o assunto dos vírus de computadores e sua evolução até os *softwares* maliciosos ou *malwares*, como são atualmente conhecidos. Serão definidos os vários tipos de *malwares* e suas técnicas de ação, bem como as formas de propagação.

Também serão definidos e discutidos outros programas maliciosos que não são classificados como *malware*, tais como: os *spams* (*e-mails* não solicitados), e *adware* (programas que fazem propagandas).

Será ainda apresentado o protocolo ICAP, com suas definições, funcionamento e arquitetura para auxiliar os serviços de borda de rede, com foco no uso do protocolo como ferramenta auxiliar no combate a vírus de computador e suas variantes.

2.2 VÍRUS DE COMPUTADOR E CÓDIGOS MALICIOSOS

2.2.1 Evolução dos Vírus de Computador

Os primeiros vírus de computador foram criados no início da década de 80. Nessa abordagem inicial, os vírus eram relativamente simples, com arquivos de auto-replicação que, quando executados, mostravam frases ou piadas.

Em 1986 foi detectado o primeiro ataque de vírus em computadores pessoais com Sistema Operacional *Microsoft* MS-DOS, este vírus foi denominado de *Brian*. [COHEN1984]

Alguns autores, entretanto, consideram outros vírus como pioneiros: *Virdem*, como sendo o primeiro vírus de arquivo e o *PC-Write*, como o primeiro cavalo de tróia (*Trojan horse*) - programa que aparentemente parece ser útil e inofensivo, mas que contém códigos maliciosos escondidos para, quando executado, causar dano ao sistema.

Quanto mais os técnicos exploram as tecnologias dos vírus, seu número ou suas plataformas, mais cresce a complexidade e a diversidade deles, o que demanda pesquisa constante.

A ação dos vírus permaneceu focada nos setores de *boot* por um tempo, depois eles passaram a atuar em arquivos executáveis. Em 1988 surgiu o primeiro *worm* de *Internet*, um tipo de *malware* que utiliza códigos maliciosos que se auto-replicam e podem automaticamente ser distribuídos de um computador para outro. O *worm Morris* acarretou a primeira lentidão substancial na *Internet*. Em resposta ao acréscimo do número de paradas, foi criado o Centro de Coordenação (CERT[4]), com o intuito de orientar o uso e garantir a estabilidade da *Internet,* por meio da coordenação das respostas às paradas e incidentes [SZOR2005].

Desde então, os vírus de computador têm se tornado cada vez mais sofisticados, passando a acessar as agendas de endereço de *e-mails* e acionando o envio de *e-mails* para todos os contatos do usuário. Além disso, os chamados macro vírus se anexam a diversas aplicações para ataques. Há também vírus criados especificamente para atacar as vulnerabilidades dos sistemas operacionais. *E-mail*, *peer-to-peer* (P2P), compartilhamento de arquivos, sites *web*, *drives* compartilhados e vulnerabilidades são alguns dos pontos explorados por vírus replicáveis. *Backdoors* são pontos de entrada escondidos que são abertos pelos *malwares* e são criados nos sistemas infectados, permitindo a entrada de vírus e *hackers*[5], para entrar remotamente e assumir o controle da máquina, executando qualquer *software* que desejar.

Alguns vírus se auto-reproduzem e se enviam por *e-mail*, ignorando qualquer configuração definida no programa instalado no cliente ou no servidor de *e-mail*. Desenvolvedores de vírus passaram a arquitetar seus ataques da forma mais cuidadosa, utilizando para isso, técnicas para que o

4 CERT.br - Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil – Página oficial: *www.cert.br*

5 *Hacker* é um programador ou um usuário de computador que tenta acessar de forma ilegal um sistema computacional ou uma rede.

conteúdo das mensagens de *e-mails* pareça autêntico. Estes artifícios são utilizados para fazer com que o usuário abra os anexos do *e-mail* que contenham vírus, e assim, ocorra à infecção em larga escala.

Com a evolução dos vírus, os *softwares* de antivírus também foram obrigados a evoluir. Entretanto, a maioria dos *softwares* de antivírus existente no mercado é baseada em assinaturas ou na detecção de características de *softwares* maliciosos, para identificação de possíveis códigos maliciosos. Observa-se então a seguinte dinâmica: entre o lançamento de um vírus, o tempo de desenvolvimento e distribuição da respectiva assinatura, existe um lapso de tempo, uma fase de não proteção. Como resultados, muitos dos vírus criados atualmente, alcançam rapidamente um significativo volume de contaminação nos primeiros dias de infecção, seguidos de rápido declínio após o lançamento da assinatura.

Com a evolução, os vírus de computador ganharam variantes e usos específicos. Por causa desta vasta classificação, hoje, todos os códigos maliciosos são classificados como *malwares*.

2.2.2 Malware

Utiliza-se o termo *malware* (uma abreviação para o termo "*software* malicioso") como um substantivo coletivo para se referir ao vírus, *worms* e "cavalos de Tróia", que intencionalmente executam tarefas maliciosas em um sistema computacional.

Desta forma, é que, atualmente, têm-se no ambiente de rede, diversos códigos maliciosos, que dificultam a identificação do vírus, *worm* ou cavalo de Tróia, bem como, de qual o antivírus é eficaz para combatê-los. A variedade desses códigos torna difícil a aplicação de uma definição específica para cada categoria de *malware*.

A *Microsoft* divide os *malwares* em três categorias:

- Cavalo de Tróia: programas aparentemente úteis e inofensivos, mas

que contém códigos escondidos e projetados para, quando executados, explorar ou danificar o sistema. O vírus do tipo “cavalo de tróia” chega aos usuários em forma de anexos, através de mensagens de *e-mail*, sendo omitida a finalidade original do programa. Também chamados de *Trojan*, os cavalos de Tróia agem instalando um pacote malicioso quando executados.

- *Worm*: certos códigos maliciosos, auto-replicáveis, que se distribuem automaticamente de um computador a outro, através das conexões de rede. O *worm* pode causar danos, como por exemplo, consumo de banda ou recursos do sistema. Enquanto outros códigos maliciosos requerem obrigatoriamente a intervenção do usuário para execução, *worms* atuam diretamente, podendo se auto-executar e espalhar sem a intervenção de terceiros. *Worms* também podem causar problemas como lentidão ou parada do equipamento, além da replicação.

- Vírus: o objetivo principal dos vírus é a replicação e seu código é criado inicialmente com esse intuito. Eles se espalham de computador para computador, se anexando a programas hospedeiros. Diversos vírus causam danos em *hardware*, *software* ou dados. Quando o hospedeiro é executado, o vírus também é executado, infectando novos hospedeiros e às vezes, incorporando um custo de infecção adicional.

As definições das várias categorias de *malware* fazem com que seja possível ilustrar estas diferenças em um simples fluxograma. A Figura 2.1 apresenta os elementos auxiliares que define a que categoria o *malware* se enquadra:

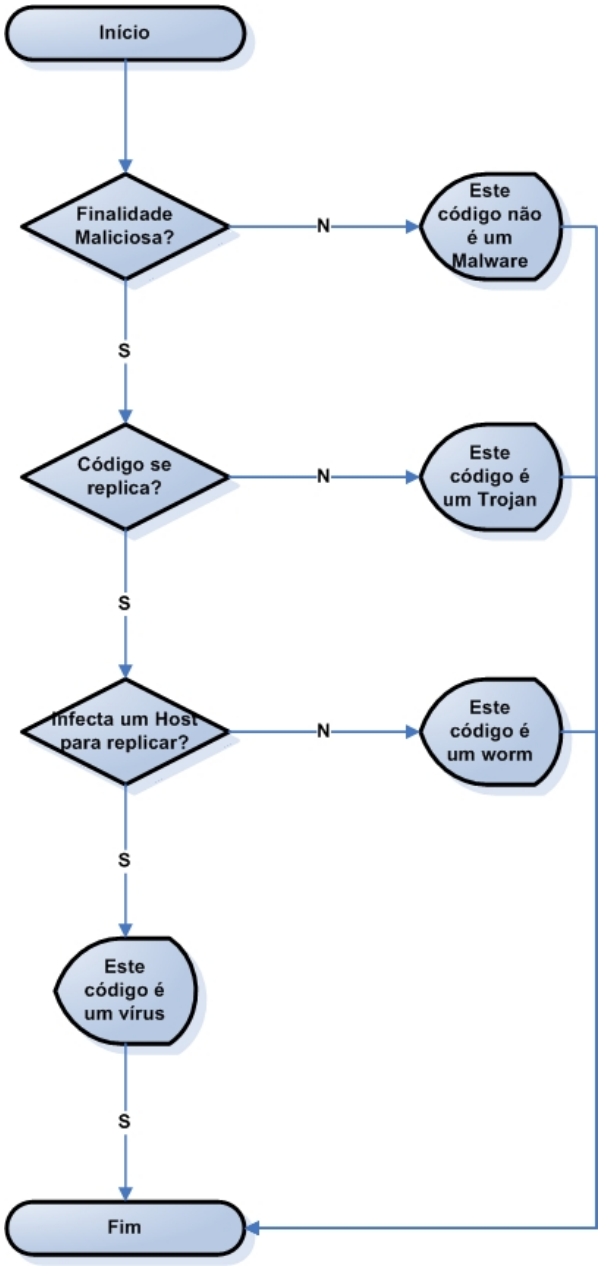

Figura 2.1: Fluxograma de classificação de *malware* [HARRISON2004].

A Figura 2.1 mostra como se torna possível distinguir a diferença entre as categorias dos códigos maliciosos mais comuns. Entretanto, é importante salientar que um ataque pode se adequar em mais de uma categoria. Estes tipos de ataques (códigos que utilizam técnicas de múltiplos vetores de ataques[6]) podem se espalhar rapidamente. Por esta razão, estes ataques diversificados aumentam a dificuldade de definir os métodos de defesa.

6 Vetor de ataque é uma rota que o *malware* pode utilizar para executar um ataque.

Nos próximos tópicos, será abordada, detalhadamente, cada categoria de *malware*, para auxiliar na ilustração dos elementos chaves de cada um [HARISSON2004].

2.2.2.1 Cavalos de Tróia

O Cavalo de Tróia ou *Trojan* como também é chamado, não pode ser considerado um vírus de computador ou um *worm*, porque ele não se propaga sozinho. Entretanto, um vírus e um *worm* podem ser utilizados para copiar um *Trojan* no sistema, como parte do ataque. Esse processo é denominado *dropping*. A intenção básica do *Trojan* é atrapalhar o usuário e as operações normais do sistema. Como por exemplo, um *Trojan* pode instalar uma porta escondida no sistema para que um *hacker* possa roubar dados ou mudar as configurações remotamente. Assim, são atividades características de *Trojans*:

1. *Trojans* de acesso remoto: alguns permitem ao hacker controlar um sistema remotamente. Estes programas são chamados de *Remote Access Trojans* (RATs) ou *Trojans* de Acesso Remoto. Exemplos de RATs: *Back Orifice, Caffene* e *SubSeven*.

2. Rootkits: são coleções de programas que um usuário mal intencionado pode utilizar para obter acesso remoto não autorizado e executar ataques adicionais em computadores. Estes programas podem utilizar um número variado de técnicas, incluindo o monitoramento de teclado, alteração dos arquivos de *logs* [7]do sistema ou das aplicações do sistema. Essas ferramentas permitem a criação de uma porta de entrada, escondida no sistema, que possibilita ataques contra outros computadores na rede.

É importante ressaltar que os RATs e algumas das ferramentas que compõe os *Rootkits* possuem propósitos legítimos e são utilizados para controle remoto de estação de trabalho e monitoramento do ambiente coorporativo.

7 Log de dados é o termo utilizado para descrever o processo de registro de eventos relevantes num sistema computacional.

Entretanto, com o uso dessas ferramentas, os riscos de violação da segurança e da privacidade podem aumentar nos ambientes em que estas são utilizadas. Uma estratégia empresarial que adote uma política bem definida para uso dos recursos computacionais, aliada ao constante investimento em treinamento do corpo técnico, pode reduzir de forma significativa o risco de violação.

2.2.2.2 Worm

Os códigos maliciosos que não se replicam, são denominados de *Trojan*. Um *worm* é um código que se replica sem a necessidade de um hospedeiro e sem a necessidade de infectar um arquivo executável.

A maioria dos *worms* tem como principal característica copiar-se em um computador hospedeiro e com isto, usar os canais de comunicação da rede para replicação. Por exemplo, o *worm Sasser* utiliza uma vulnerabilidade para iniciar a infecção no sistema e então faz com que o sistema infectado utilize a rede de comunicação para se replicar. Se as últimas atualizações de segurança estiverem instaladas ou ainda se o *firewall* estiver habilitado para bloquear as portas que os *worms* utilizam, o *worm* não infectará o equipamento. Por exemplo, num ambiente utilizando o *Windows XP* com o *Service Pack* 2 instalado, ambos os métodos de infecção e replicação irão falhar. Isto ocorre, porque a vulnerabilidade foi removida e o *Windows Firewall* foi habilitado por *default*.

2.2.2.3 Vírus

Se um código malicioso adiciona uma cópia de si mesmo em um arquivo, documento ou no setor de *boot* de um disco, a fim de se replicar, este código então será considerado um vírus. Esta cópia pode ser uma cópia do vírus original ou uma versão modificada. Como mencionado anteriormente, um vírus sempre irá conter um *payload* que será aplicado a um computador local, assim como o *Trojan*, o que então irá acarretar em um ou mais atos maliciosos, como por exemplo, a remoção de dados. Um vírus que tem a

função primária de replicação, não possuindo nenhum *payload*, é considerado um *malware*, porque pode agregar funções de corrupção de dados, controlador dos recursos do sistema e consumidor de banda na replicação.

2.2.2.4 Spyware

Este tipo de programa é, às vezes, chamado de *spybot* ou programa de rastreamento. O *spyware* utiliza outras formas de programas e aplicativos fraudulentos que conduzem certas atividades no computador, sem o consentimento do usuário. Estas atividades podem incluir a coleta de informações pessoais, bem como a modificação da configuração do navegador da *Internet*. Além de ser um incômodo, *spywares* resultam em uma variedade de problemas que vão desde queda do desempenho total do computar até violação da sua privacidade.

2.2.3 Característica dos Malwares

As várias características que cada categoria de *malware* apresenta são muito similares. Por exemplo, os vírus e os *worm* utilizam a rede como meio de transporte. Entretanto, os vírus têm interesse em infectar arquivos, enquanto o *worm* simplesmente se copia.

Nos próximos tópicos serão mencionadas as principais características dos *malwares* [SZOR2005].

2.2.3.1 Pré-Requisitos

Como os *malwares* se preocupam em atacar um sistema, pode existir um número específico de componentes que são pré-requisitos para que um ataque seja bem sucedido. Os pontos abaixo são exemplos típicos do que os *malwares* necessitam para atacar um equipamento (*host*):

- **Dispositivos:** alguns *malwares* efetuam seus ataques, dependendo do dispositivo físico, como por exemplo: os computadores pessoais, um *Apple Machintosh* ou até um *Personal Digital Assistent* (PDA).

- **Sistemas Operacionais:** os *malwares* podem requerer sistemas operacionais específicos para sua infecção. Por exemplo, o vírus *Chernobyl*, criado no início da década de 90, atacava apenas computadores que utilizavam sistema operacional *Microsoft Windows* 95 ou 98.

- **Aplicativos:** Os *malwares* podem, ainda, requerer que aplicativos específicos estejam instalados em seus alvos, antes de conseguirem sucesso na infecção ou replicação. Por exemplo, o vírus *LFM.926*, lançado no ano de 2002, só conseguia efetivar seu ataque se, no computador local, os arquivos *Shockwave Flash* (.swf) pudessem ser executados diretamente.

2.2.3.2 Carrier Objects

Considerando que o *malware* é considerado um vírus, o mesmo necessita de um hospedeiro para se infectar. O número e o tipo de hospedeiros variam de acordo com o tipo de *malware*. A lista a seguir, exemplifica os tipos mais comuns de *carrier objects* ou também chamados de hospedeiros [SZOR2005]:

- **Arquivos Executáveis:** este é o principal alvo da maioria dos vírus. Além da tradicional extensão dos arquivos executáveis (.exe), arquivos com as extensões .com, .sys, .dll, .ovl e .prg também são visados.

- ***Scripts*:** ataques que utilizam *scripts*[8], como arquivos hospedeiros além de linguagens do tipo *Visual Basic Script*, *Java Script*, *Apple Script* ou *Perl Script*. Extensões destes tipos de arquivos incluem: .vbs, .js, .wsh e .prl.

- **Macros:** são arquivos que suportam a linguagem macro *Script,* como por exemplo, arquivos de documentos, planilhas eletrônicas ou banco

8 *Scripts* são arquivos contendo linguagem interpretada.

de dados. Os vírus podem utilizar a linguagem macro no *Microsoft Word* e *Lotus Ami Pro* para produzir um número de efeitos indesejáveis.

- **Setor de *boot*:** atua especificamente em áreas de disco (disco rígido e mídias "bootáveis"). Os *Master Boot Record*[9] (MBR) ou DOS *boot record* podem ser considerados hospedeiros, porque são capazes de executar os códigos maliciosos na inicialização.

2.2.3.3 Mecanismo de Transporte

Um vírus pode utilizar um ou mais métodos para replicação. Nesta sessão serão abordados os mais comuns mecanismos de transporte que os *malwares* utilizam [SZOR2005].

- **Mídias**: o original e mais fácil método de proliferação dos vírus de computador e outros *malwares* é pela transferência de arquivos. No início, se utilizavam dos discos removíveis, posteriormente, partiram para rede e, atualmente, utilizam-se de mídias como os dispositivos *Universal Serial Bus* (USB) e *Firewire*. A taxa de infecção não é tão rápida quanto os *malwares* de rede, contudo ela está sempre presente e é de difícil erradicação por causa da necessidade de modificação dos dados entre os sistemas.

- **Compartilhamento de rede:** após o advento das redes, os computadores ganharam a capacidade de intercomunicação rápida. Os desenvolvedores de *malware*, percebendo a oportunidade, focaram seus esforços em utilizar essa nova tecnologia na disseminação de suas pragas virtuais. As antigas mídias removíveis (disquetes e CDs) perderam espaço e os códigos maliciosos evoluíram. Hoje, mecanismos de proteção de rede precisam estar presentes e constantemente atualizados para combater essa nova

9 *Master boot record* é primeiro setor que contém informação da estrutura organizacional do disco (partições, código de arranque do sistema operativo, e assinatura desse código).

forma de ataque.

- **Scaneamento de Rede**: os desenvolvedores de *malware* utilizam este método, a busca de vulnerabilidades na rede, para atacar, de forma randômica, endereços de IP. Esse mecanismo pode, por exemplo, enviar um pacote em uma porta especifica para um *range* de endereços IP, com o intuito de achar uma vulnerabilidade em algum computador que possa ser atacado.

- **Redes *peer-to-peer* (P2P):** em geral, para a transferência de arquivos P2P ocorrer, um usuário precisa primeiro, instalar um componente da aplicação P2P, que usará uma das portas liberadas pelo *firewall* de uma organização. A aplicação usa esta porta para passar pelo *firewall* e transferir arquivos diretamente de um computador a outro. Estas aplicações estão disponíveis na *Internet* e permitem um mecanismo de transporte que os desenvolvedores podem usar, diretamente, para ajudar a espalhar arquivos infectados nos discos rígidos.

- ***E-mail*:** o *e-mail* se tornou o mecanismo de transporte mais utilizado para ataque de *malwares*. A facilidade com que milhares de centenas de pessoas são contatados, por meio de correio eletrônico, faz do *e-mail* um dos meios de transporte mais eficazes para disseminação de vírus. É relativamente simples fazer com que os usuários abram os anexos contidos nos *e-mails*, utilizando técnicas de engenharia social. Existem basicamente, dois tipos de *malware* que utilizam o *e-mail* com este mecanismo:

    - *Mailer*: este tipo de *malware* envia *e-mail* a um número limitado de endereços de *e-mails*, utilizando-se de *softwares* instalados no hospedeiro (como por exemplo, o *Microsoft Outlook Express*) ou usando seu próprio protocolo (*Simple Mail Transfer Protocol* – SMTP).

- *Mass Mailer*: este tipo de *malware* que cria *e-mails* em massa e os envia para centenas de milhares de pessoas, utiliza-se de *softwares* de *e-mail* instalados ou o protocolo SMTP.

- ***Remote Exploit*:** os *malwares* podem explorar uma determinada vulnerabilidade em serviços ou aplicações, ao invés de se replicarem. Esta característica é principalmente vista nos *worms*; por exemplo, o *worm Slammer* tira vantagem sobre uma vulnerabilidade no *Microsoft SQL Server* 2000. O *worm* gera um *buffer*[10] *overrun*[11] que permite a uma pequena parcela da memória do sistema ser reescrita com um código que pode ser executado no mesmo contexto do *SQL Server*. A *Microsoft* identificou e solucionou esta vulnerabilidade, meses antes do *Slammer* ser liberado, mas poucos sistemas foram atualizados e o vírus se espalhou.

2.2.3.4 Payloads

Uma vez o computador infectado, o *malware* causará um *payload,* ou como o próprio nome significa, há um custo de infecção. Esse *payload* pode se apresentar de várias formas. Alguns dos tipos de *payload* mais comuns são identificados abaixo [SZOR2005]:

- ***Backdoor*:** este tipo de *payload* permite acessos não autorizado, completo ou limitado, no computador, assim que habilita o acesso ao *File Transfer Protocol* (FTP), via porta 21. Se o ataque foi para habilitar o *Telnet*, um *hacker* pode usar o computador infectado como plataforma para ataques remotos em outros computadores. Além disso, *backdoor* é também considerado um *Trojan* de Acesso Remoto.

- **Dados corrompidos ou deletados:** um dos mais destrutivos tipos de *payload* é o código malicioso que corrompe e apaga dado, deixando o computador inutilizável. O desenvolvedor de *malware* tem duas

10 *Buffer* é uma região de memória temporária utilizada para escrita e leitura de dados.

11 *Buffer overrun* é uma condição que resulta na adição de informações em um *buffer*.

opções para programar códigos que agem dessa forma. A primeira é desenvolver um código de execução rápida. Como o código é extremamente destrutivo, ele precisa se disseminar rapidamente caso contrário ele vai eliminar a chance de auto-replicação. A segunda opção é armazenar o código no sistema afetado por um período (por exemplo, na forma de um *Trojan*), permitindo assim, sua disseminação antes de executar o código destrutivo ou o usuário perceber sua presença.

- **Roubo de Informações**: um tipo preocupante de *payload* é o que tem como principal função o roubo de informações. Se o *payload* puder comprometer a segurança do hospedeiro, é possível que ele estabeleça um roubo de informações. Isso pode ocorrer de diversas maneiras, como por exemplo: a transferência pode ser automática, assim o *malware* simplesmente captura o local dos arquivos ou das informações, como o nome ou as senhas de usuários. Outro mecanismo é fornecer um ambiente local no hospedeiro que permita ao *hacker* controlar o hospedeiro remotamente ou permitir o acesso a arquivos diretamente do sistema.

2.2.3.5 Trigger

Trigger ou mecanismo de acionamento é uma característica dos *malwares* que o programa malicioso utiliza para iniciar réplicas ou *payload*. Mecanismos *trigger* incluem os seguintes [SZOR2005]:

- **Execução manual:** este tipo de mecanismo é simplesmente a execução do *malware* diretamente pela vitima.
    - **Engenharia social:** o código malicioso frequentemente usará alguma idéia para ajudar a enganar a vitima a executar manualmente o *malware*. A aproximação pode ser relativamente simples, como aquelas utilizadas em malas diretas com *worms,* em que o texto contém argumentos para

convencer o usuário a agir conforme solicitado. Para um usuário desavisado, essa mensagem de *e-mai,l* provavelmente, será aberta. Os desenvolvedores de *malware* também podem utilizar-se de *e-mails* fraudulentos (*spoofing*[12]) para fazer com que a vítima acredite que o remetente do *e-mail* é uma fonte confiável.

- **Execuções semi-automáticas:** esse tipo de mecanismo de acionamento é primeiro iniciado pela vítima e então executado de forma automática daquele ponto em diante.

- **Execuções automáticas:** este mecanismo não requer nenhuma execução manual. O *malware* executa seu ataque sem necessidade da vítima acionar qualquer programa malicioso no computador alvo.

- **Bomba relógio:** esse mecanismo executa uma ação, após um período determinado. Este período pode ser um atraso da primeira execução do *bug*, alguma data predeterminada ou uma escala variada.

- **Condicional:** esse tipo de mecanismo utiliza algumas condições predeterminadas para entrega de *payload*.

2.2.3.6 Mecanismos de Auto-Defesa

Vários exemplos de *malware* utilizam algum tipo de mecanismo de defesa para ajudar a reduzir a probabilidade de detecção e remoção. A seguinte lista fornece exemplos de algumas destas técnicas empregadas [SZOR2005]:

- **Armadura (*Armor*):** esse tipo de mecanismo de defesa aplica algumas técnicas que tentam impedir a análise do código mal-intencionado. Tais técnicas incluem detectar quando um *debugger*[13]

12 *Spoofing* é o ato de imitar um Web site ou uma transmissão de dados para fazê-la parecer original.

13 *Debugger* é um programa de depuração, ou seja, é utilizado para encontrar *bugs* numa aplicação ou mesmo em *hardware*.

está funcionando e tentar prevenir para que funcione corretamente; ou ainda adiciona vários códigos sem sentido para tornar difícil determinar o propósito do código mal-intencionado.

- **Invisível (*Stealth*):** o *malware* utiliza esta técnica para esconder-se, interceptando pedidos de informação e retornando dados falsos.

- **Criptografia:** o código malicioso que utiliza este tipo de mecanismo codifica-se ao *payload* para impedir a detecção ou a recuperação dos dados. O código malicioso codificado contém uma rotina estática de decodificação, uma tecla codificada e a codificação mal-intencionada (que inclui uma rotina de codificação). Quando executado, o *malware* utiliza a rotina de decodificação e tecla para decodificar o código mal-intencionado. O *malware* cria então, uma cópia do seu código e gera uma nova tecla de codificação. Ele utiliza esta tecla e sua rotina de codificação para codificar a sua nova cópia, adicionando à nova tecla a rotina de decodificação ao inicio da nova cópia. Ao contrário dos vírus polimorfos, o *malware* codificado sempre utiliza a mesma rotina de decodificação, assim, embora o valor chave (e, deste modo, a assinatura do código mal-intencionado codificada) mude de infecção para infecção, os programas de antivírus podem buscar a rotina estática decodificada para detectar *malwares* que utilizam este mecanismo de defesa.

- **Oligomorfos*:*** *malwares* que exibem essa característica utilizam a codificação como um mecanismo de defesa e são capazes de modificar a rotina de codificação somente em uma determinada quantidade (geralmente um número pequeno). Por exemplo, um vírus que pode gerar duas rotinas diferentes de codificação.

- **Polimorfos:** o *malware* desse tipo utiliza a codificação como um mecanismo de defesa para modificar-se e prevenir a detecção, normalmente codificando o próprio código malicioso com uma rotina codificada, e então fornecendo uma chave decodificada diferente para

cada mutação. Assim, os *malwares* polimorfos utilizam um número ilimitado de rotinas codificadas para prevenir a detecção. Conforme o *malware* se reproduz, uma parcela do código decodificado é modificada. Dependendo do código especifico do *malware,* o *payload* ou outras ações executadas podem ou não usar codificação. Normalmente há um motor, que é um componente já contido no *malware* codificado, que gera rotinas ao acaso de codificação. Este motor e o *malware* são, então, codificados e a nova decodificação é passada junto com eles.

2.2.4 O que não é um Malware

De acordo com a *Microsoft*, existe ainda uma variedade de ameaças, mas que não são consideradas *malware*, pois não são programas de computadores desenvolvidos com uma intenção maliciosa. Contudo, essas ameaças possuem implicações de segurança e finanças para uma organização. Por essas razões, o administrador deve entender o que essas ameaças representam para a infra-estrutura de TI de uma corporação e para a produtividade dos usuários [SZOR2005].

2.2.4.1 Joke Programs

Joke Programs são projetados para produzir uma brincadeira, ou na pior das hipóteses, fazer com que o usuário perca seu tempo. Estas aplicações existem desde o início da utilização de computadores pessoais. Isto porque eles não são criados com intenções maliciosas, sendo claramente identificados como piadas.

2.2.4.2 Hoaxes

Atualmente, um grande número de *hoaxes* tem surgido na comunidade de TI. Como algumas formas de *malware*, um *hoax* utiliza a engenharia social para tentar enganar usuários e levá-los a fazer algo. Contudo, neste caso, o *hoax* não tem código a ser executado. O *hoax* geralmente tenta enganar a vítima com certos procedimentos, como por exemplo, uma mensagem de *e-*

mail “informando” sobre o descobrimento de um novo tipo de vírus. O que acontece é que, com a finalidade de alertar os amigos, os usuários reenviam o mesmo tipo de “alerta” a outros usuários. Esses *hoaxes* utilizam recursos do servidor de *e-mail* e consomem banda de rede.

2.2.4.3 Scams

Várias formas de comunicação vêm sendo usadas por criminosos para tentar enganar usuários, divulgação de informações falsas que dão ao usuário a impressão de que o mesmo pode obter algum ganho financeiro. A *Internet*, *Websites* e *e-mails* não são exceções. Uma mensagem que tenta enganar o endereçado, a fim de obter informações pessoais, pode ser utilizada para diversos propósitos, como por exemplo para obtenção de informações de conta bancária. Um outro tipo de *scam* que ficou conhecido é o *phishing*, pronunciado como “*fishing*”, que também é conhecido como “*brand spoofing or carding*”. Exemplos de *phishing* incluem casos em que o criminoso cria um espécie de réplica de alguns *sites* já conhecidos, na tentativa de enganar os usuários e adquirir informações confidenciais.

2.2.4.4 Spam

Spam são *e-mails* não solicitados, gerados para anunciar algum serviço ou produto. Este fenômeno é geralmente considerado um incômodo, mas o *spam* não é um *malware*. Contudo, o crescimento dramático no número de mensagens de *spam* enviadas tem sido um problema para a infra-estrutura da *Internet*, bem como para empresas, visto que, entre outras coisas, esse tipo de *e-mail* resulta na diminuição da produtividade do empregado que se vê forçado a lidar com tais mensagens todos os dias. A origem do termo *spam* é disputada, mas independente da sua origem, não há dúvida que o *spam* se tornou uma das irritações mais persistentes nas comunicações que têm como base a *Internet*. Muitos consideram o *spam* um assunto significativo, que agora ameaça a saúde da comunicação, via *e-mail,* ao redor do mundo. Deve ser notado, entretanto, que exceto pela carga tolerada pelos servidores de *e-mail* e de programas *anti-spam*, o *spam* não é

capaz de replicar ou ameaçar a saúde e operação de um sistema organizacional de TI.

O *spam* tem sido freqüentemente utilizado por seus criadores, os chamados *spammers*, que instalam um servidor simples de *e-mail* (SMTP) em um computador anfitrião e então, este é utilizado para remeter mensagens de *spam* para outros receptores de *e-mails*.

2.2.4.5 Adware

O *adware* está, normalmente, combinado com uma aplicação hospedeira que é fornecida sem nenhum custo (*payload*), desde que o usuário concorde em aceitar o *adware*. Já que a aplicação do *adware* é instalada, após o usuário aceitar o "acordo de uso" que indica a finalidade da aplicação, nenhuma ofensa é cometida. Entretanto, as propagandas *pop-up* podem se tornar um incômodo e em alguns casos degradar a desempenho do sistema.

2.2.4.6 *Cookies* de Internet

Cookies são arquivos de texto inseridos no computador do usuário por *Web sites* que o usuário visita. Os *cookies* contêm e fornecem informações de identificação sobre o usuário ao *Web site* que os aloca no computador do usuário, junto com qualquer informação que o *site* quer reter sobre a visita do mesmo. *Cookies* são ferramentas legitimas que muitos *Web sites* utilizam para rastrear informações sobre o visitante. Um exemplo poderia ser o caso do usuário que ao visitar uma loja *on-line* e tendo colocado um ou mais itens em seu "carrinho de compras" virtual, decide ir a outro *site*, por qualquer motivo. A loja pode decidir por "salvar" a informação sobre quais produtos estavam no carrinho de compra em um *cookie* no computador do usuário, permitindo que ao retornar ao site, o usuário encontre no "carrinho" os itens que ele selecionou previamente, o que facilita, caso ele deseje finalizar a compra.

Teoricamente, os desenvolvedores de *Web sites* podem recuperar somente informações armazenadas no *cookie* que eles criaram. Este acesso deve

assegurar a privacidade do usuário prevenindo que qualquer pessoa, que não seja o desenvolvedor do *site*, possa acessar o *cookie* alocado no computador do usuário.

Sabe-se que alguns desenvolvedores de *Web sites* usam os *cookies* para coletar informações sem o conhecimento do usuário. Alguns podem enganar os usuários ou omitir suas políticas de uso. Podem, por exemplo, rastrear os hábitos de uso da *Internet* em vários *Web sites* sem avisar o usuário. De posse das informações, os desenvolvedores podem então utilizar os dados ou informações para personalizar a propaganda que o usuário vê no *Web site*. É difícil identificar este tipo de propaganda foco e outras formas de abuso no uso de *cookies*, o que torna difícil decidir se, quando, e como bloqueá-los do seu sistema.

Além disso, o nível de aceitação do compartilhamento de informações varia entre os usuários de computador, tornando difícil criar um programa anti-*cookie* que irá atingir a necessidade de todos os computadores em um ambiente.

2.2.5 Software de Antivírus

Os programas de antivírus são especialmente criados para defender o sistema contra as ameaças de códigos maliciosos presentes. É recomendado o uso de programas de antivírus porque defenderá o sistema do computador contra todas as formas de códigos maliciosos e não somente de vírus.

Há um número de técnicas que os programas de antivírus utilizam para detectar os *malwares*. Esta seção discute o funcionamento de algumas destas técnicas [SZOR2005]:

- **Verificação de assinatura:** a maioria dos programas de antivírus usa atualmente esta técnica, que envolve a busca de alvos (computador hospedeiro, *drive* de disco ou arquivos) por um modelo que pode representar um código malicioso. Este modelo, geralmente, é

guardado em arquivos chamados de arquivos de assinatura que são atualizados pelos vendedores de programas regularmente, para assegurar que o *scanner* do antivírus reconheça a maior quantidade de ataques de *malwares* conhecidos. O principal problema com esta técnica, é que o programa de *software* já deve estar atualizado para contra-atacar o código malicioso antes que o *scanner* possa reconhecer.

- **Verificação heurística:** técnica que tenta detectar as novas e conhecidas formas de *malwares* buscando pelas suas características gerais. A principal vantagem desta técnica é que ela não se baseia em arquivos de assinatura para identificar e contra-atacar o código malicioso. Entretanto, a varredura heurística possui alguns problemas específicos, incluindo:

    - **Falso-positivo: t**écnica que usa as características gerais, e é, portanto, propensa a reportar programas legítimos como *malwares,* se a característica for similar em ambos os casos.

    - **Verificação lenta:** o processo de analisar características é um procedimento mais difícil para o programa realizar do que buscar um padrão conhecido de *malware*. Por esta razão, a verificação heurística pode demorar mais do que a verificação por assinatura.

    - **Novas características:** se um novo ataque de *malware* apresentar uma característica que não tenha sido previamente identificada, a varredura heurística provavelmente não identificará até o programa ser atualizado.

- **Obstrução de comportamento:** técnica que foca no comportamento de ataque do *malware,* ao invés do próprio código. Por exemplo, se um aplicativo tentar abrir um portal de rede um antivírus de obstrução de comportamento, isso pode ser detectado como uma atividade

típica de um código malicioso e então, o comportamento ser catalogado como um possível ataque.

2.2.6 Tempo de Vida de um Malware

Um padrão surgiu para definir o tempo de vida dos novos ataques de *malware* disponíveis na rede pública de computadores bem como definir em quanto tempo novos ataques irão para a rede. Uma revisão deste padrão ajuda a entender o risco que novos ataques de *malwares* provocam após serem liberados.

Uma nova linha do tempo inicia quando o código malicioso é desenvolvido e termina quando todos os rastros são removidos das redes monitoradas. Os estágios da linha do tempo são:

1. **Concebimento:** o desenvolvimento de um *malware* geralmente começa quando um novo método de ataque ou exploração é sugerido e então compartilhado entre as comunidades de *hackers*. Por algum tempo esses métodos são discutidos e explorados, até que uma maneira de abordagem, que torne possível desenvolver o ataque, seja descoberta.

2. **Desenvolvimento:** anteriormente para a criação de um *malware* era necessário entender a linguagem do conclave de computadores e o complicado trabalho do sistema que era atacado. Entretanto, com a chegada de *kits* e salas de bate-papo na *Internet* tornou-se possível para quase todos com más intenções criar um código malicioso.

3. **Replicação:** após um *malware* ter sido desenvolvido e solto na rede, normalmente tem que se replicar para um dispositivo hospedeiro em potencial, por algum tempo, antes que possa realizar suas funções primárias ou realizar a entrega de seu *payload*

4. **Entrega de *payload*:** após um *malware* ter infectado o hospedeiro, tem início a entrega de seu *payload*. Se o código tem um acionador

condicional para o seu *payload*, esse estágio é o ponto nos quais as condições para a entrega do mecanismo são localizadas. Por exemplo, alguns *payloads* são acionados quando o usuário realiza certa ação ou quando o relógio da máquina do hospedeiro chega a certa data. Se o *malware* possuir uma ação direta ativa, simplesmente irá iniciar a entrega do *payload* no ponto onde a infecção estiver completa.

5. **Identificação:** neste ponto, o *malware* é identificado pela comunidade de antivírus. Na maioria dos casos esse passo ocorrerá antes do quarto estágio ou mesmo antes do terceiro estágio, mas nem sempre.

6. **Detecção:** após a ameaça ser identificada, os desenvolvedores do programa de antivírus precisam analisar o código para determinar a confiabilidade do método de detecção. Após terem determinado o método, eles então atualizam a assinatura dos arquivos do antivírus para permitir que a aplicação do antivírus detecte este novo *malware*. O tempo que este processo toma é crucial para ajudar no controle de uma eclosão.

7. **Remoção:** após a atualização ser disponibilizada ao público, é da responsabilidade dos usuários do programa de antivírus fazer a atualização em tempo hábil para proteger seus computadores contra o ataque ou para limpar seus sistemas, no caso de já terem sido infectados.

À medida que mais usuários atualizam seus programas de *software,* o *malware* irá lentamente deixar de ser uma ameaça. Este processo raramente remove todos os casos de *malware* na rede, porque existem sempre alguns computadores conectados à Internet, que dispõe de pouca ou nenhuma proteção de antivírus, nos quais o *malware* pode residir. Entretanto, a ameaça de ataque como um todo é minimizada.

Embora esta linha de tempo se repita para cada novo ataque desenvolvido,

não é caracterizada uma linha típica de todos os ataques. Muitos ataques são simples versões modificadas de uma parcela original de *malware*. Então o código base e a maneira de abordar são iguais, mas pequenas mudanças são feitas para ajudar a prevenir que o ataque seja detectado e removido. Normalmente, um ataque bem sucedido de *malware* vai iniciar um número de revisões durante as próximas semanas ou meses. Essa situação leva a um tipo de “corrida de braço”, no qual os autores de *malware* tentam evitar a detecção para o seu próprio ganho, seja este por propósitos financeiros, má reputação ou simplesmente curiosidade. As defesas do antivírus são, então, novamente atualizadas, reformadas ou modificadas, conforme a necessidade de diminuir o novo ataque.

2.3 TECNOLOGIAS DE SUPORTE AO ICAP

Esta seção abordará as Chamadas de Procedimento Remoto (RPC) como mecanismo de funcionamento do protocolo ICAP, e as ferramentas e programas (*Proxy*) que utilizam o protocolo ICAP para prover serviço de adaptação de conteúdo.

2.3.1 Proxy-Cache

O serviço de *Proxy* tem o papel de concentrar todas as requisições das mais diversas origens, canalizando-as para uma mesma saída. Ele é que, efetivamente, faz a requisição ao destino, funcionando como um intermediário entre o cliente e o servidor de destino. Esse intermediário efetua tais requisições, segundo regras ou filtros, implementados pela ferramenta de *proxy*. Tais filtros têm as funções de proibir ou liberar acessos a sites, endereços identificadores de máquinas, redes, *strings*[14] e até limitar velocidade de acesso. Além de ser capaz de coibir o acesso através de regras que atuam sobre os clientes da rede interna: nomes de usuários, grupos de usuários, endereços identificadores de máquinas, etc.

O serviço de *Cache*, por sua vez, funciona como um repositório das páginas já visitadas. Ele armazena todo o tráfego, permitindo economia na utilização do *link* de comunicação de dados. Ele faz com que um conteúdo remoto, fique armazenado para visitas futuras, evitando que o conteúdo seja requisitado novamente.

Soluções presentes no mercado contemplam a utilização dos serviços de *proxy* e *cache* juntos, sendo que o tratamento da informação entre tais serviços é realizado automaticamente pela aplicação, sem a necessidade de intervenção do administrador, conforme figura abaixo.

14 *String* é uma cadeia de caracteres.

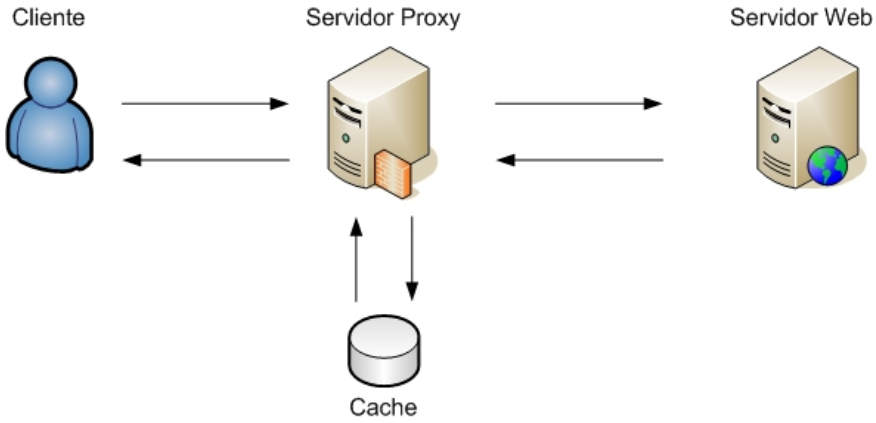

Figura 2.2 :Funcionamento de *Proxy* e *Cache*.

De acordo com a figura acima (Figura 2.2), o cliente deseja acessar um serviço *Web*, por exemplo, HTTP, GOPHER ou FTP. Cada usuário poderia fazer um acesso único, porém, banda desnecessária seria consumida sendo que mais de um usuário acessa o mesmo recurso durante todo o dia. Para facilitar esse acesso, o cliente faz uma requisição ao servidor *Proxy*, este provê acesso ao recurso desejado pelo cliente. Caso o cliente acesse um *Web site* (por exemplo, www.uniceub.br), o *Proxy* faz o *download* do site e o armazena em seu *cache*. Caso outro usuário deseje acessar o mesmo *Web site*, não haverá assim, a necessidade de acessá-lo novamente, basta apenas o *proxy* entregar o conteúdo existente em seu *cache*, acelerando a resposta ao cliente.

2.3.2 RPC – Remote Procedure Call

O RPC (*Remote Procedure Call*) é um protocolo que provê o paradigma de comunicação de alto-nível e é usado em sistemas operacionais.

O RPC pressupõe que tenha sido previamente instalado um protocolo de comunicação de baixo nível, como o TCP/IP (*Transmission Control Protocol/Internet Protocol*) ou UDP (*User Datagram Protocol*) para transmitir os pacotes de dados entre as aplicações que se comunicam. O RPC implementa um sistema de comunicação lógica entre cliente-servidor, projetado especificamente para trabalhar sobre redes.

O protocolo RPC foi constituído sobre o protocolo XDR (*eXternal Data*

Representation), que define os padrões de troca de dados em aplicações que se comunicam remotamente. O XDR converte os parâmetros e resultados para cada requisição RPC.

Este protocolo permite que usuários possam trabalhar com procedimentos remotos, como se estes fossem procedimentos locais. Por exemplo, um usuário na máquina A pode solicitar que a máquina B execute a função soma (a+b) nela existente e esta então, repassa o resultado novamente para a máquina A. As chamadas de procedimentos remotos são definidas por meio de rotinas existentes no protocolo RPC. Para cada requisição, existe uma mensagem de retorno. O protocolo RPC é um protocolo de troca de mensagens, que implementa outros protocolos, como por exemplo, chamadas de procedimentos remotos *broadcasting*[15].

O protocolo RPC provê um mecanismo de autenticação, que identifica o servidor e o cliente para ambos, sendo suportados vários tipos de sistemas de criptografia, como por exemplo, o DES - *Data Encryption Standard*. No protocolo RPC, cada servidor executa um programa, que é um conjunto de procedimentos remotos. A combinação de endereço do servidor, número do programa e do procedimento, identificam uma *procedure* remota. Quando um pacote com uma requisição chega ao servidor, este aciona uma função que executa o procedimento requisitado e retorna outro pacote com a requisição feita.

A *interface* do protocolo RPC é, geralmente, usada para a comunicação entre processos de máquinas remotas. Contudo, o protocolo RPC trabalha da mesma forma, com diferentes processos/programas, rodando na mesma máquina.

15 *Broadcast* é um endereço IP que permite que a informação seja enviada para todas as maquinas de uma LAN, MAN, WAN e TANS, redes de computadores e sub-redes.

2.4 INTERNET CONTENT ADAPTION PROTOCOL – ICAP

O *Internet Content Adaption Protocol* (ICAP) foi introduzido pelo ICAP Fórum, em 1999. O ICAP Fórum é uma coalizão de empresas com soluções e serviços voltados para a Internet e foi co-fundado pelas empresas *Network Appliances*[16] e *Akamai Technologies*[17].

O ICAP é um protocolo criado, originalmente, para executar chamadas remotas em mensagens HTTP. Ele faz parte de uma arquitetura que permite às grandes corporações, serviços de provedor de *Internet* e companhias de comunicação de dados, verificarem dinamicamente determinado tráfego, permitindo a alteração do mesmo, quando estes passam pelos servidores ICAP. O Protocolo permite que os clientes ICAP enviem solicitações para os servidores ICAP objetivando a "adaptação de conteúdo", como por exemplo, a verificação de vírus em um dado arquivo.

O ICAP é um protocolo para ser utilizado em servidores na primeira camada de proteção da rede, ou seja, na DMZ[18] (*DeMilitarized Zone*). Ele permite uma infra-estrutura única de comunicação, integrando equipamentos de diferentes fabricantes e possibilitando a gerência do conteúdo do tráfego entre os diversos serviços.

A implantação de serviços agregados é um enorme passo no ambiente de servidores *cache* e gerenciadores de conteúdo. Com essa nova abordagem nos serviços, as operadoras e provedores de *Internet* poderão disponibilizar para seus clientes arquiteturas com maior escalabilidade e serviços com maior confiabilidade.

2.4.1.1 Características

O conceito de prover novos serviços para atrair novos usuários, não é uma

16 Página oficial: *http://www.netapp.com/*

17 Página oficial: *http://www.akamai.com/*

18 DMZ é uma pequena rede situada entre uma rede confiável e uma não confiável, geralmente entre a rede local e a *Internet*.

idéia nova, aqueles que tentaram se aventurar por este caminho descobriram que a solução não era escalável. E também, a tecnologia normalmente afetava a confiabilidade dos serviços oferecidos, fazendo com que os usuários procurassem outros métodos. A maioria das soluções sofria da falta de uma infra-estrutura única. Alguns serviços eram executados exclusivamente em API's[19] proprietárias, que eram customizadas para uma aplicação em particular. Alguns serviços provêem aplicações para os clientes, porém alguns serviços em específico consomem muitos recursos, tais como banda e processamento, gerando assim, um atraso na resposta do servidor.

De acordo com essas necessidades, surgiu o ICAP Fórum. O seu intuito foi definir o novo protocolo de adaptação de conteúdo. Esse grupo foi criado para estudar e gerar um produto que atendesse aos seguintes pré-requisitos: [NA2001]:

- "Ser simples;
- Ser escalável;
- Usar a infra-estrutura física existente;
- Ser modular em seus serviços;
- Usar os métodos básicos e existentes de comunicação; e
- Prover uma economia de recursos, apenas otimizando os serviços de borda elevados (*extranet* DMZ)".

ICAP é um padrão aberto de protocolo, isto é, um protocolo que permite acesso ao código fonte para realização de mudanças e adaptação a outros protocolos, facilitando assim, a comunicação com dispositivos de outros fabricantes, semelhante ao TCP/IP. Este provê o protocolo de comunicação

19 API (*Application Programming Interface*) é um conjunto de rotinas e padrões estabelecidos por um *software* para utilização de suas funcionalidades por programas aplicativos.

para criar serviços customizados para que a aplicação receba da melhor forma a resposta. ICAP é uma chamada de procedimento remoto (RPC - *Remote Procedure Call*), baseado em "pergunta/resposta" (*request/response*) que permite que clientes enviem requisições HTTP para "adaptação" de conteúdo, do tipo, análise de vírus em um arquivo.

O protocolo ICAP é especificado na RFC 3507, publicada em abril de 2003, como um memorando informativo. A RFC 3507 faz uma introdução sobre o protocolo ICAP, nos seguintes termos:

> *"ICAP, Protocolo de Adaptação de Conteúdo da Internet, é um protocolo que provê um off-load20 de conteúdos específicos para serviços HTTP. ICAP é, em sua essência, um protocolo leve para executar uma 'chamada de procedimento remoto' em mensagens HTTP. Permite aos clientes ICAP passar mensagens HTTP para servidores ICAP com alguma transformação ou outros processamentos (adaptação). Os servidores executam estes serviços de transformações em mensagens e as enviam em respostas aos clientes, usualmente são mensagens modificadas. Tipicamente, estas mensagens adaptadas são requisições HTTP ou respostas HTTP."*

O ICAP é, portanto, um protocolo desenvolvido para fazer com que serviços específicos de análise/adaptação de conteúdo passem a funcionar em servidores dedicados, com isso, recursos de *hardware* poderão ser otimizados e também existirá um padrão de comunicação de como as funcionalidades serão implementadas.

2.4.2 Serviços de Implementação

Os serviços que podem ser implementados com a utilização do ICAP são [NA2001]:

1. **Verificação de Vírus**

A verificação de vírus sempre foi deixada a cargo da rede receptora ou PC e cada objeto pode ser verificado muitas vezes, o que gera um desperdício de

20 *Off-load* tem como função, assumir parte da necessidade de processamento de outro dispositivo.

recursos. A verificação de vírus em tempo real do ICAP permite que objetos que já passaram por uma verificação prévia e não estão contaminados sejam arquivados e depois despachados sem vírus.

2. **Filtro de conteúdo**

Essa é a habilidade de redirecionar uma requisição não autorizada ou restrita para uma outra página.

3. **Tradução de linguagens (PDA's e Celulares)**

É a habilidade de permitir a dispositivos sem suporte a HTML, como os celulares, de falar com dispositivos HTML, como os PCs e vice-versa. Pode-se dizer que isso seja uma ponte de tradução WAP/XML/HTML. Sobre o ICAP, as entradas do ponto de presença podem residir em qualquer lugar que o ponto de transmissão pode estar na rede. Um "armazenador" pode manipular todas as requisições dos clientes e redirecioná-las aos servidores de tradução do ICAP e manter armazenadas cópias de objetos, em vários formatos, para uma resposta melhor para seus clientes.

4. **Inserção de propaganda**

Essa é a habilidade de inserir propaganda em páginas *Web* ou de propagar páginas nas preferências, histórico ou localização de um cliente, quando o mesmo requisitar o *Web site* com uma ferramenta de busca. É baseado no endereço IP originário do servidor *Proxy*, palavras-chave adicionadas pelo cliente ou a coleta de informações do cliente (perfil).

5. **Tradução de linguagem humana**

Essa é a habilidade de traduzir um conteúdo formatado em HTML, de uma linguagem em outra, por exemplo, de inglês para japonês. Sobre o ICAP, algumas requisições serão originadas pela geografia ou a introdução direta será traduzida pelos servidores ICAP, via redirecionamento a servidores *Proxy*.

6. **Compressão de dados**

Essa é a habilidade de compactar páginas HTML de um servidor de origem. As respostas do servidor de origem podem ser comprimidas, permitindo a economia de banda de rede.

2.4.2.1 Vantagens

As vantagens do ICAP são: [NA2001]

- Uma infra-estrutura utilizando ICAP para gerenciamento de conteúdo, permite à operadora oferecer serviços otimizados, com economia na utilização da banda do *link* de comunicação de dados. Além de permitir redução na utilização do processamento dos servidores, o ICAP possibilita o gerenciamento centralizado do conteúdo trafegado.

- O ICAP é um protocolo aberto, possui código fonte disponível para acesso (*open source*) e sua utilização não possui custos. Os servidores de acesso à *Internet* e as empresas podem escolher exatamente o provedor de aplicação de valor agregado mais apropriado.

- O ICAP pode coletar informações de interesse do cliente e utilizá-las para propagandas mais focadas nesses indivíduos. Por exemplo: quando um usuário acessa vários *sites* que contem informações sobre o assunto “carros esportivos”, o ICAP percebe e utiliza essa informação coletada para vincular propagandas sobre “carros esportivos” nas novas páginas que aquele usuário está visualizando. Um exemplo prático disto é o que o *Google* tem feito no serviço de correio eletrônico dele (*Gmail*). Na abertura da mensagem, ele faz a verificação de contexto e apresenta no lado direito da tela, as propagandas relacionadas ao conteúdo.

2.4.3 Arquitetura ICAP

O ICAP executa suas adaptações de conteúdo HTTP, utilizando uma das quatro técnicas de adaptação existentes. Duas delas efetuam as modificações atuando no cabeçalho HTTP: **Modificação de Requisição** e **Satisfação de Requisição**. Já as outras duas agem modificando a resposta HTTP: **Modificação de Resposta** e **Modificação de Resultado**. [NA2001]

O protocolo ICAP funciona na mesma arquitetura que a grande maioria dos protocolos na *Internet*, a arquitetura cliente-servidor. A porção cliente é chamada de Cliente ICAP, ele se apresenta como uma camada de apresentação, recebendo as solicitações dos usuários. O segundo componente, é o Servidor ICAP, que é responsável pelo processamento de conteúdo e aplicação da política definida pelo administrador. A RFC que define o ICAP faz referência a um terceiro componente, para que as quatro técnicas de adaptação funcionem de forma satisfatória. Esse terceiro componente é chamado de ICAP *switch-box*, ele atua como um interceptador e redirecionador, de requisições ICAP, fazendo papel intermediário de transações entre o Cliente ICAP e o Servidor ICAP.

Para completar a arquitetura, outros dois elementos tomam forma, a estação do usuário final, aqui definido simplesmente como usuário. Ele é que realiza as solicitações de conteúdo, tais como página *Web*, arquivo etc. O segundo elemento é o Servidor *Web*, referenciado aqui como Servidor da *Internet*. Ele é o responsável por disponibilizar o conteúdo.

Apesar da RFC fazer referência ao ICAP *switch-box,* como um componente de software separado, nas implementações do protocolo nas soluções comercial, esse componente de *software* é parte integrante do Servidor ICAP e neste projeto será apresentado desta forma.

A seguir será detalhado cada uma das quatro possibilidades de utilização do ICAP.

2.4.3.1 Modificação de Requisição

Pelo método de **Modificação de Requisição** (Figura 2.3)**,** um usuário envia uma requisição para o Cliente ICAP (*Proxy/cache*) (1 - Figura 2.3) que por sua vez encaminha o pedido ao Servidor ICAP (2 - Figura 2.3). O Servidor ICAP efetua a alteração da requisição original e devolve a requisição com o conteúdo alterado para o Cliente ICAP (3 - Figura 2.3). O Cliente ICAP encaminha essa requisição modificada para o Servidor de Internet (4 - Figura 2.3). Como última parte no processo, o Servidor de Internet recebe a requisição e responde para o Cliente ICAP (5 - Figura 2.3) que por fim, encaminha a requisição ao usuário que a originou (6 - Figura 2.3).

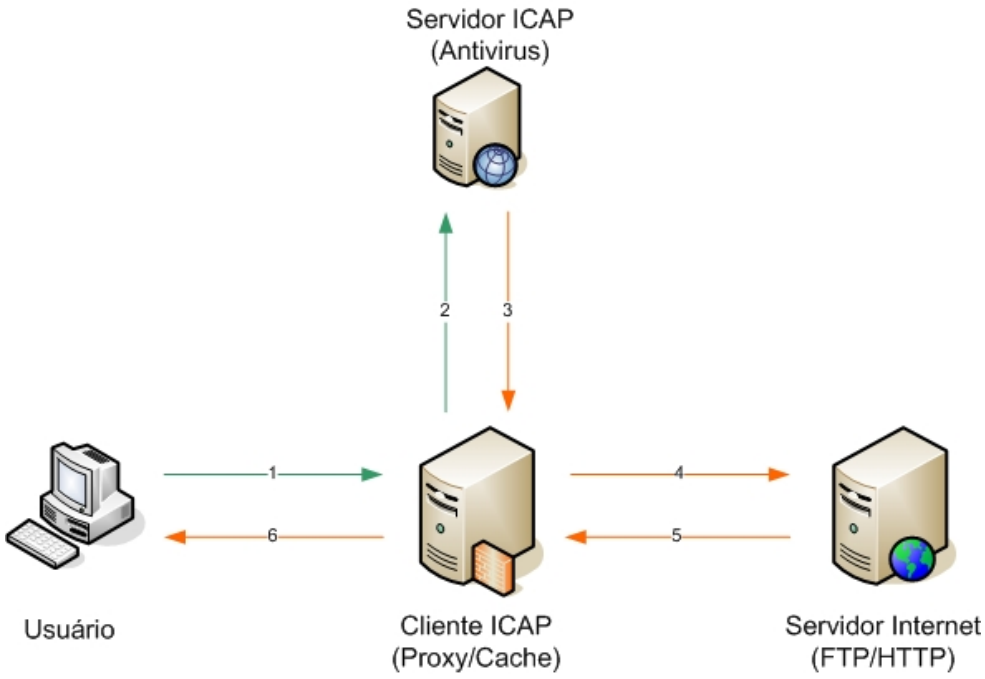

Figura 2.3: Método de Modificação de Requisição

Exemplo 1: Filtro de Conteúdo

Um usuário deseja enviar uma requisição para acessar uma página *Web*. para isto, o servidor *Proxy* (Cliente ICAP) recebe esta requisição e redireciona ao Servidor ICAP. O Servidor ICAP analisa esta requisição HTML e a compara com uma lista de URLs proibidas. Se a URL requirida

estiver na lista das URLs proibidas, a requisição então é modificada para uma requisição de mensagem de erro do Servidor de *Internet* ou do Servidor *Proxy* (*cache*). Esta mensagem de erro, é então entregue ao cliente. Se a URL não for proibida, o Servidor ICAP envia a requisição ao Servidor de Internet, via Servidor *Proxy,* e a requisição será processada.

2.4.3.2 Satisfação de Requisição

Na **Satisfação de Requisição** (Figura 2.4) um usuário envia uma requisição para o Cliente ICAP (*Proxy/cachê*) (1 - Figura 2.4). O Cliente ICAP encaminha a requisição ao Servidor ICAP onde é efetuada a alteração no conteúdo da requisição original (2 - Figura 2.4). O Servidor ICAP direciona esta requisição modificada diretamente para o Servidor de Internet (3 - Figura 2.4). Após o processamento da requisição, o Servidor de Internet retorna a resposta ao Servidor ICAP (4 - Figura 2.4). O Servidor ICAP entrega a resposta ao Cliente ICAP (5 - Figura 2.4), que por sua vez, encaminha a resposta ao originador da requisição (6 - Figura 2.4).

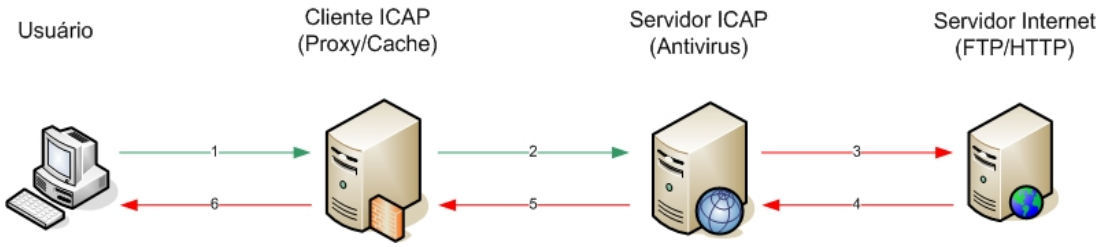

Figura 2.4: Método de Satisfação de Requisição.

Exemplo 2: Filtro de Conteúdo

Usando o mesmo exemplo do Filtro de Conteúdo, o processo muda em relação à Requisição de Modificação. A requisição do usuário é ainda examinada pelo Servidor ICAP, porém se o conteúdo não for autorizado, o Servidor ICAP não encaminhará a requisição para o Servidor Internet e retornará uma resposta de erro para o Servidor *Proxy* (Cliente ICAP). Entretanto, se a requisição estiver autorizada a acessar o Servidor de Internet, o Servidor ICAP efetuará a requisição ao Servidor de Internet e a

retornará ao Servidor *Proxy* que por sua vez fará a entrega final para o usuário.

2.4.3.3 Modificação de Resposta

Na **Modificação de Resposta** (Figura 2.5) um usuário envia uma requisição para o Cliente ICAP (*Proxy/cahe*) (1 - Figura 2.5), que por sua vez encaminha o pedido ao Servidor na Internet (2 - Figura 2.5). O Servidor de Internet recebe a requisição e responde para o Cliente ICAP (3 - Figura 2.5), este por sua vez encaminha a resposta para o Servidor ICAP (4 - Figura 2.5), onde é feita a adaptação/modificação de conteúdo. O Servidor ICAP redireciona esta resposta para o Cliente ICAP (5 - Figura 2.5). O cliente ICAP envia a resposta para ser armazenada em cache. Como última parte no processo, o Cliente ICAP entrega a resposta para o usuário que originou a requisição (6 - Figura 2.5).

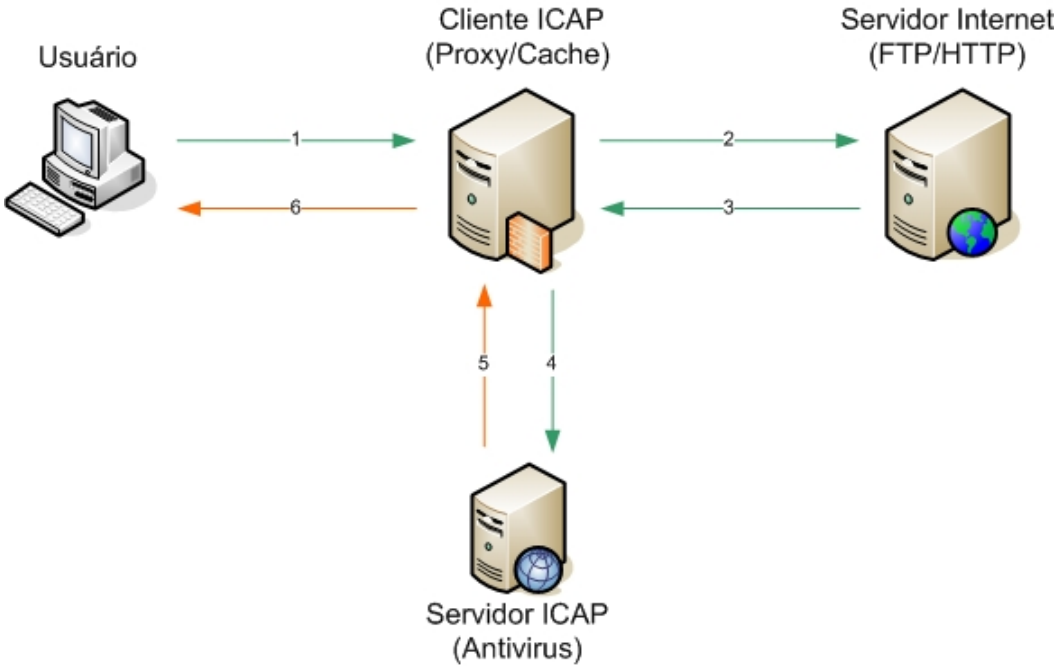

Figura 2.5: Método de Modificação de Resposta

Exemplo 3: Tradução de Gateway – Formato HTML

Uma requisição é efetuada, por meio de um aparelho celular, para o estoque de uma empresa. O pedido é enviado ao Servidor de Internet que processa

a requisição. A resposta do Servidor de Internet, entretanto, é redirecionada ao Servidor ICAP que modifica a resposta para permitir que o aparelho celular receba a resposta corretamente modificada.

2.4.3.4 Modificação de Resultado

No método de **Modificação de Resultado** (Figura 2.6), o usuário envia uma requisição para o Cliente ICAP (*Proxy/cahe*) (1 - Figura 2.6), que a encaminha ao Servidor de Internet (2 - Figura 2.6). A requisição é aceita normalmente pelo Servidor de Internet e a resposta é enviada para o Cliente ICAP (3 - Figura 2.6). Após receber a resposta, o Cliente ICAP efetua cache da resposta, em seguida encaminhando-a para o Servidor ICAP (4 - Figura 2.6). O Servidor ICAP modifica o resultado e o entrega para o Cliente ICAP (5 - Figura 2.6). O Cliente ICAP, por sua vez, efetua a entrega ao usuário (6 - Figura 2.6).

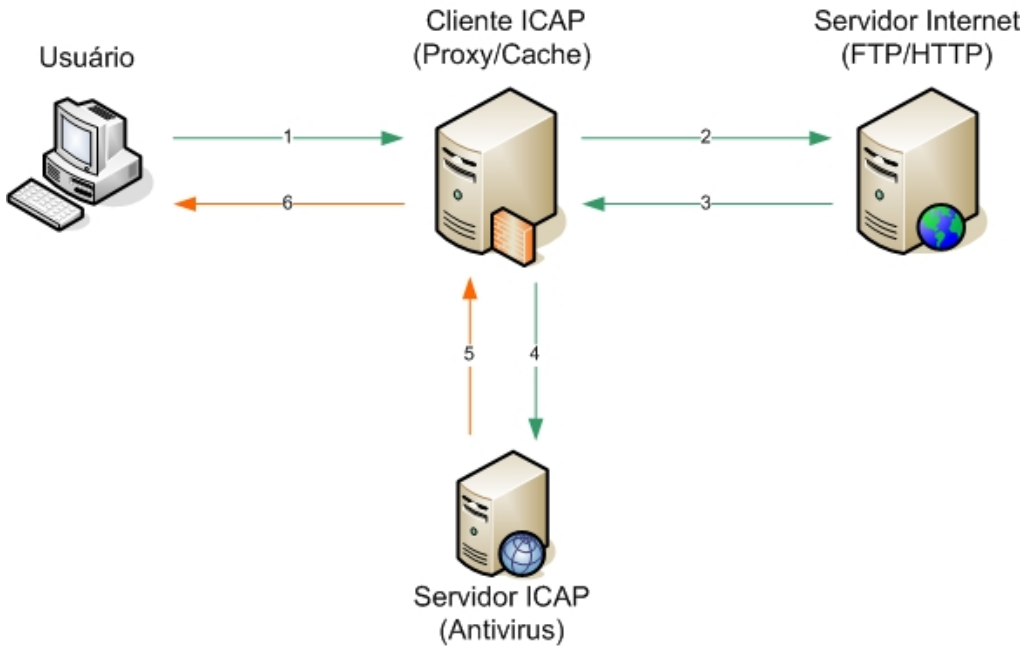

Figura 2.6: Método de Modificação de Resultado

Exemplo 4: Inserção de Dados

Um usuário solicita uma requisição de acesso a uma página *Web*. O Servidor *Proxy (Cliente ICAP)* direciona esta requisição ao Servidor de

Internet. O Servidor de Internet processa a requisição e retorna a resposta ao servidor *Proxy*. O *Proxy* armazena, em *cache*, os objetos e direciona a resposta ao Servidor ICAP. O Servidor ICAP analisa a resposta HTML e executa o *profiling* do cliente. O ICAP Server adiciona *targets* e envia a resposta ao usuário.

O método de Modificação de Resultado (2.4.3.4) se diferencia do método de Modificação de Resposta (2.4.3.3), pois em implementações da Modificação de Resultado, o serviço de *cache* fica antes do Servidor ICAP, coibindo a execução de *cache* de respostas já verificas pelo Servidor ICAP. Diferentemente do método de Modificação de Resposta, que o serviço de *cache* fica após a verificação das políticas do servidor ICAP.

O quadro seguinte (Quadro 2.1) resume os serviços a serem aplicados para cada um dos quatro tipos de técnicas de aplicação, que envolve a arquitetura ICAP.

SERVIÇOS	MODIFICAÇÃO DE REQUISIÇÃO	SATISFACÃO DE REQUISIÇÃO	MODIFICAÇÃO DE RESPOSTA	MODIFICAÇÃO DE RESULTADO
FILTRAGEM DE CONTEÚDO	X	X	X	X
TRADUÇÃO DE *GATEWAY*	X		X	
TRADUÇÃO DE LINGUAGEM	X	X	X	
VERIFICAÇÃO DE VÍRUS			X	
INSERÇÃO DE PROPAGANDA	X	X	X	X
COMPRESSÃO DE DADOS			X	X

Quadro 2.1: Quadro de serviços para cada Arquitetura [NA2001].

CAPÍTULO 3. INFRA-ESTRUTURA DO PROJETO

3.1 INTRODUÇÃO

Este Capítulo trata das especificações e desenvolvimento deste Projeto.

Inicialmente, será apresentada a topologia do Projeto, em seguida será explicada a arquitetura desenvolvida para a execução do *benchmark*, proposta nesta monografia.

3.2 TOPOLOGIA

O Projeto físico será dividido em dois cenários, com e sem a utilização do ICAP.

3.2.1 Cenário I

A Figura 3.1 mostra a estrutura física montada para o Cenário I, onde não é utilizado o protocolo ICAP.

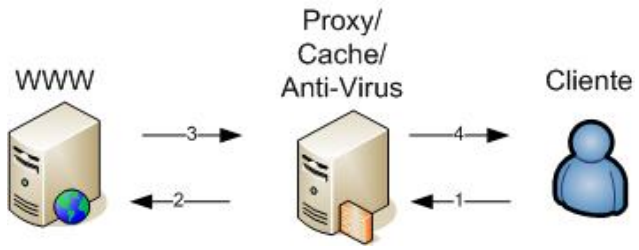

Figura 3.1: Cenário I – Situação sem serviço ICAP.

No Cenário I (Figura 3.1), o cliente envia uma requisição para *download* de arquivo ao servidor *Proxy* (1 - Figura 3.1). O *Proxy* altera o cabeçalho do pacote e efetua a conexão no servidor *Web* remoto (2 - Figura 3.1). O servidor *Web* responde à solicitação do *Proxy* e envia o arquivo requisitado ao mesmo (3 - Figura 3.1). Neste momento, o servidor *Proxy* aguarda a conclusão do *download* para em seguida efetuar a busca por vírus. Neste momento podem ocorrer duas situações: a primeira, se o arquivo não estiver

contaminado, o *Proxy* armazenará o mesmo em cache, com objetivo primário de acelerar *downloads* futuros; e a segunda, se o arquivo estiver contaminado, o *Proxy* irá deletar o arquivo por completo, isto devido a configuração setada no antivírus *Clamav*. Finalmente, o servidor *Proxy* responde a requisição de *download* ao cliente (4 - Figura 3.1).

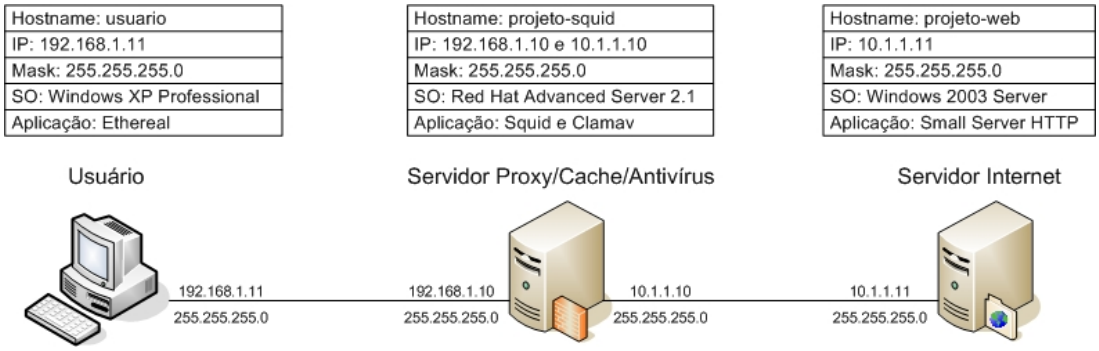

Figura 3.2: Topologia Física do Cenário I.

Já a Figura 3.2 define a topologia física utilizada no Cenário I.

3.2.2 Cenário II

A Figura 3.3 mostra a estrutura física montada para o Cenário II, onde é utilizado o protocolo ICAP.

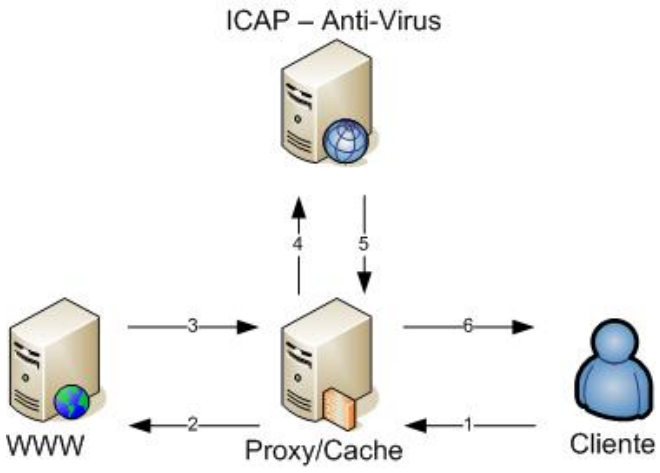

Figura 3.3: Cenário II - Serviço ICAP habilitado.

Já no Cenário II (Figura 3.3), o cliente envia uma requisição para *download* de arquivo para o servidor *Proxy* (1 - Figura 3.3). O *Proxy* altera o cabeçalho do pacote e efetua a conexão no servidor *Web* remoto (2 - Figura 3.3). O

servidor *Web* remoto responde à solicitação e envia o conteúdo ao servidor *Proxy* (3 - Figura 3.3). Neste momento, o servidor *Proxy* recebe a solicitação e envia o arquivo para verificação de vírus em um servidor ICAP – Antivírus (4 - Figura 3.3). Após a verificação, se o arquivo não contiver conteúdo malicioso, o servidor ICAP – Antivírus devolve o arquivo para o servidor *Proxy* (5 - Figura 3.3). O servidor *Proxy* efetuará o armazenamento do arquivo em *cache* para atender requisições futuras do mesmo conteúdo. Se o arquivo contiver conteúdo malicioso, o servidor ICAP irá deletar o arquivo por completo, isto devido à configuração setada no antivírus *Symantec Scan Engine*. Finalmente, após o servidor *Proxy* responde a requisição de *download* ao cliente (6 - Figura 3.3).

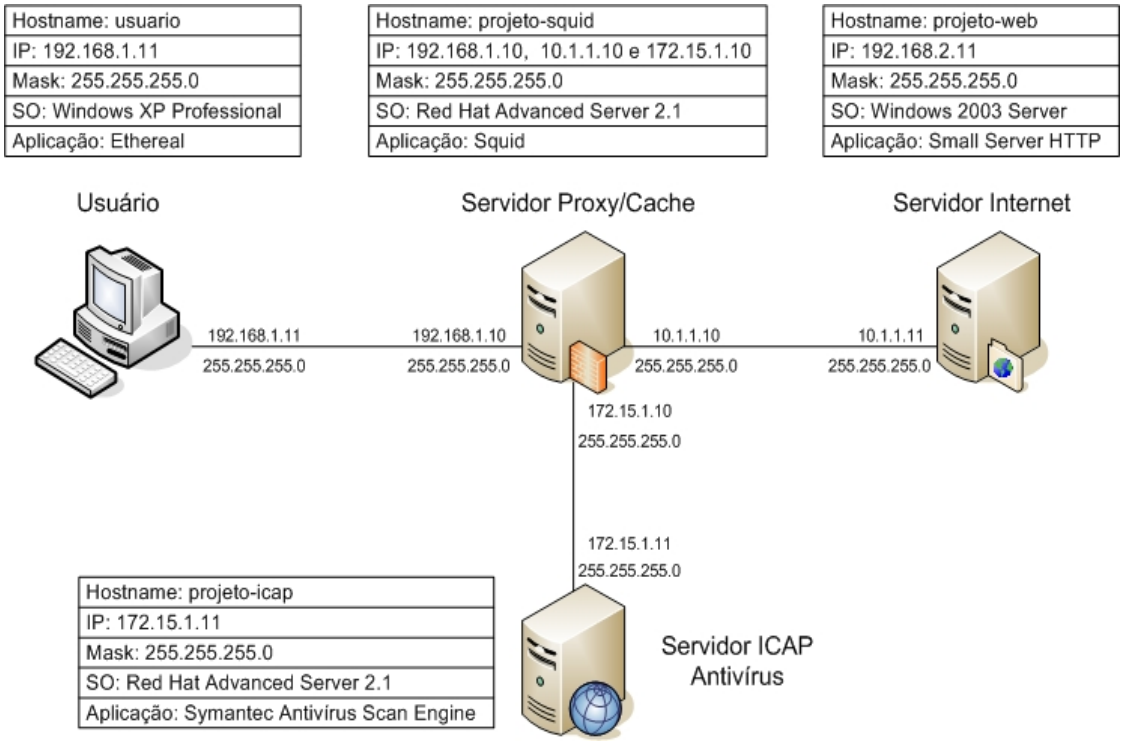

Figura 3.4: Topologia Física do Cenário II.

Já a Figura 3.4 define a topologia física utilizada no Cenário II.

3.3 HARDWARE

Para a execução dos testes deste Projeto, foram utilizados os seguintes *hardwares*:

3.3.1 Solução Usuário

Nome do Computador: projeto-usuário
Endereço IP: 192.168.1.11
Tipo: Torre
CPU: Mobile *Unknown*, 1800 MHz (9 x 200)
Memória: 1024 Mb
Disco Rígido: *Maxtor* 6Y120L0 (120 GB, 7200 RPM, Ultra-ATA/133)
Placa de Rede: *Marvell Yukon* 88E8001/8003/8010 PCI *Gigabit Ethernet Controller*
Sistema Operacional: *Microsoft Windows XP Professional*

3.3.2 Solução Servidor Cliente ICAP – Squid e ClamAV

Nome do Computador: *projeto-squid*
Endereço IP: 192.168.1.10, 10.1.1.10 e 172.15.1.10
Tipo: Torre
CPU: *AuthenticAMD AMD Athlon(tm) Processor1 1001 MHz i686 Processor*
Memória: 128 Mb
Disco Rígido: *Maxtor* 6Y120L0 (120 GB, 7200 RPM, Ultra-ATA/133)
Placa de Rede 01: *Accton*|SMC2-1211TX
Placa de Rede 02: *Realtek RTL8139 Family PCI Fast Ethernet*
Placa de Rede 03: *VIA VT6105 Rhine III Fast Ethernet Adapter*
Sistema Operacional: *Red Hat Advanced Server 3.0*

3.3.3 Solução Servidor ICAP – Symantec Antivírus Scan Engine

Nome do Computador: projeto-icap
Endereço IP: 172.15.1.11
Tipo: *Notebook*
CPU: *ALi Corporation*|*M5229 IDE*
Memória: 256Kb
Disco Rígido: IC25N030ATCS04-0 20Gb
Placa de Rede: *Realtek*|*RTL-8139/8139C/8139C+*
Sistema Operacional: *Red Hat Advanced Server 3.0*

3.3.4 Solução Servidor Internet - HTTP

Nome do Computador: projeto-web
Endereço IP: 10.1.1.11
Tipo: Torre
CPU: *Intel Pentium* II, 400 MHz (4x100), A-Trend ATC-6220
Memória: 256 Mb
Disco Rígido: WDC WD100EB-34BHFO 10Gb
Placa de Rede: *Realtek RTL8139 Family PCI Fast Ethernet*
Sistema Operacional: *Microsoft Windows Server 2003, Enterprise Edition*

3.4 SOFTWARE

Nesta seção serão apresentados os principais *softwares* aplicados nas soluções: Usuário, Cliente ICAP, Servidor ICAP e Servidor Internet.

3.4.1 Solução Usuário

Com base no artigo *"Windows Dominates on the Desktop*[21]*"*, pesquisas realizadas com os dados estatísticos das requisições, efetuadas nos servidores com maior tráfego na *Internet*, afirmam que 1% destas requisições são originadas por máquinas, utilizando sistema operacional *Linux*, 3% utilizando sistema operacional *Apple Macintosh* e o restante, 96% das requisições, são originadas de máquinas utilizando *Microsoft Windows*.

De acordo com esse cenário, o sistema operacional adotado para simular o lado "cliente" da solução foi o sistema operacional *Microsoft Windows XP Professional*, a explicação dessa decisão, se dá ao fato de que a maior parte dos usuários finais, no mundo, utilizarem em seus computadores tal sistema operacional.

3.4.1.1 Ferramenta para cronometrar o tempo de *download*

Para execução do *benchmark* proposto neste Projeto (5.2), foi necessária a medição do tempo de *download* dos vários arquivos pelo cliente. A primeira ferramenta utilizada foi o *software Get Right Download Manager*[22] (*GetRight*).

A escolha de um aplicativo, para funcionar como gerenciador de *download*, era a mais interessante, uma vez que a medição comparativa de desempenho da solução ICAP com a solução convencional, exige que seja cronometrado o tempo de *download* de arquivo no lado "cliente". O *Internet Explorer* não atende esta demanda.

21 Laura Rohde, IDG News Service, 2003

22 Página oficial: *www.getright.com/*

Entretanto todos os *donwloads* eram efetuados em um tempo muito curto, dando resultados imprecisos e invalidando as medições. Para resolver o problema, a solução encontrada foi utilizar um capturador de pacotes, que escutava a placa de rede do cliente. Cada medição foi efetuada usando o início do estabelecimento da conexão TCP entre o cliente-servidor S*quid* (*three-way-handshacking*) e a finalização dessa conexão. O capturador utilizado foi o *Ethereal*.

3.4.2 Solução Servidor Cliente ICAP

Para a implementação do Cliente ICAP foram utilizados os *softwares: Squid Web Proxy Cache* e o antivírus *ClamAV,* de acordo com o Cenário I (3.2.1).

3.4.2.1 Squid Web Proxy Cache

Squid é um *Proxy-cache* de alta performance para clientes *Web* que suporta protocolos como o FTP, *Gopher* e HTTP.

Pode-se dizer, que o *Squid* consiste em um programa principal - *squid* -, um sistema de busca e resolução de nomes - *dnsserver* - e alguns programas adicionais para reescrever requisições, fazer autenticação, gerenciar ferramentas de clientes e fazer *cache*.

O *Squid* tem como principais características: manter metadados[23] e especialmente objetos armazenados na RAM; armazena em *cache* buscas de DNS; e implementa *cache* negativo de requisições não-respondidas. Ele suporta SSL[24], listas de acesso complexas e *logging* completo. Por utilizar o *Internet Cache Protocol* (ICP), o *Squid* pode ser configurado para trabalhar de forma hierárquica ou mista para melhor aproveitamento da banda.

O *Squid* pode ser executado nas principais plataformas do mercado, como *Linux*, *Unix* e *Windows*.

23 Metadados são dados capazes de descrever outros dados, ou seja, dizer do que se tratam, dar um significado real e plausível a um arquivo de dados,são a representação de um objeto digital.

24 SSL (*Secure Sockets Layer*) é um protocolo desenvolvido pela *Netscape* para promover o tráfego seguro de dados na *Internet*.

O *Squid* tem como principais vantagens:

- A continuidade no aprimoramento de seu desempenho. Além de adicionar novas características, possui excelente estabilidade em condições extremas.

- Sua compatibilidade com várias plataformas e a imensa gama de *software* para analisar *logs*, gerar relatórios, melhorar o desempenho e adicionar segurança provido pela comunidade *open source*, combinados com ferramentas de administração simplificada e baseadas em *Web* agregam grande valor ao produto.

- Pode-se, ainda, citar a capacidade de *clustering*[25], *Proxy* transparente, *cache* de FTP e, é claro, seu baixo custo. Por fim, o sistema é totalmente aberto, possibilitando a sua otimização no nível de código, além da otimização via configuração.

3.4.2.2 Clam Antivírus - ClamAV

Para atender às requisições do Cenário I (3.2.1), será necessária a utilização de um *software* antivírus. Como existe uma variedade de antivirus para sistemas operacionais *Linux* disponíveis no mercado, tanto comerciais como livres, será utilizado para este Projeto uma solução livre, o *Clam*[26] Antivírus (*ClamAV*).

O *ClamAV* é uma ferramenta de antivírus GPL[27] para *UNIX*. O objetivo principal desse *software* é a sua integração com servidores de *e-mail* (varredura de anexos). O pacote Clam provê um servidor multitarefa bastante flexível, um *scanner* de linha de comando e uma ferramenta para atualização automática via *Internet*.

Apesar do objetivo principal do *ClamAV* ser a sua integração com servidores

25 *Cluster* é um tipo especial de sistema distribuído construído a partir de computadores convencionais (*desktops*).

26 Página oficial: *http://www.clamav.net/*

27 GNU GPL ou simplesmente GPL, é a Licença Pública Geral é a designação da licença para *software* livre.

de *e-mail*, este também pode ser usado em *on-access scanning*[28], em sistemas *Linux* e *FreeBSD*, utilizando-se uma *thread*[29] do *clamd,* chamada *Clamuko*.

Os programas são baseados em uma biblioteca compartilhada, distribuída com o pacote *ClamAV*, a qual pode ser usada em qualquer *software*. O mais importante, é que o banco de dados do *ClamAV* está sempre sendo atualizado, garantindo assim, freqüentes atualizações da biblioteca de vírus e conseqüentemente maior proteção a seus usuários.

Entre as características do *ClamAV*, encontram-se:

- *Scanner* de linha de comando;
- Servidor rápido e muti-tarefa;
- *Interface* para *Sendmail*;
- Atualizador de bando de dados;
- Detecção de mais de 20000 vírus, *worms* e *trojans*;
- Suporte interno a arquivos RAR (2.0), Zip, Gzip, Bzip2, Tar e outros; e
- Suporte interno a arquivos de *e-mails* dos tipos *mbox*, *Maildir* e *raw*.

3.4.2.3 SquidClamav

Para a comunicação entre o processo de *Proxy* e o processo de varredura de arquivo (antivírus), é necessária a inserção de uma camada de *software* para funcionar como *interface* entre os dois processos principais. Essa camada de *software* é o *SquidClamav*. Ele é um *script* redirecionador do antivírus *ClamAV* para o *Squid* que permite fácil verificação de vírus nos arquivos trafegados pelo *proxy*.

O *Squid* utiliza os recursos do *SquidClamav* para redirecionar todos os acessos dos usuários à *Internet* ao *ClamAV*. O *ClamAV* verifica se existe

28 On-access scanning

29 *Thread* é uma maneira de um processo dividir a si mesmo em duas ou mais linhas de tarefas simultâneas.

algum vírus nesses acessos e retorna a requisição ao *SquidClamav,* que por sua vez, retorna somente os acessos válidos ao *Squid*.

3.4.3 Solução Servidor ICAP

Para este Projeto, poderia ser utilizado um dos quatro antivírus, conforme o Quadro 3.1 que mostra os servidores ICAP antivírus existentes.

Produtos	***Windows***	***Solaris***	***Linux***
Symantec Antivirus Scan Engine	**X**	**X**	**X**
WebWasher (NAI/CAI Engine)	**X**	**X**	**X**
Finjan SurfinGate for Web	**X**	**X**	
Trend Micro InterScan WebProtect for ICAP		**X**	

Quadro 3.1: Servidores ICAP Antivírus.

Dos produtos disponíveis, o *Finjan* e o *TrendMicro* foram descartados por não estarem implementados em plataforma aberta, a utilização destes, inviabilizaria o Projeto, pois o custo desta solução está fora do orçamento disponível para este Projeto. Sobraram as soluções da *McAfee* e da *Symantec*, as maiores do mercado. A solução da *McAfee* utiliza *hardware* proprietário de custo maior, do que as duas outras soluções juntas. Já a solução da *Symantec*, foi cedida uma licença, pela representante da *Symantec* em Brasília, para este uso acadêmico. O nome do produto utilizado foi o *Symantec Antivirus Scan Engine* (SAVSE), como ferramenta para a verificação de vírus.

3.4.3.1 Symantec Antivirus Scan Engine

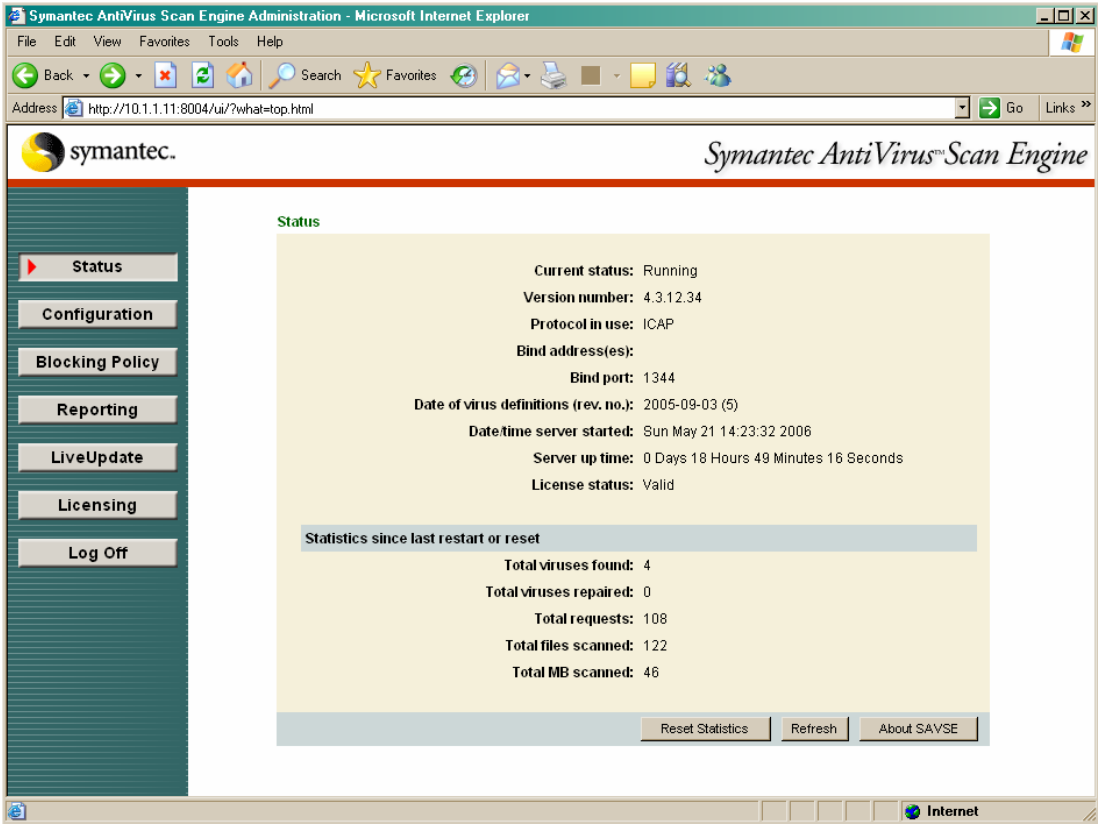

Figura 3.5: *Console* Inicial do *Symantec Antivirus Scan Engine*

Para os testes e desenvolvimento do Servidor ICAP, foi utilizado o *Symantec Antivirus Scan Engine* (SAVSE) 5.x. Uma requisição de verificação pode ser enviada, via SAVSE, pelo seu protocolo nativo, pelo Protocolo de Adaptação de Conteúdo da Internet (ICAP) ou pela Chamada de Procedimento Remoto (*Remote Procedure Call* – RPC) [SAVSE2002]. De acordo com [SAVSE2002], *"o Symantec Antivirus Scan Engine provê verificação de vírus e capacidade de reparação de qualquer aplicação na rede IP, independente de plataforma. Qualquer aplicação pode ter seus arquivos verificados no Symantec Antivirus Scan Engine, o que no caso de arquivos infectados serão limpos se necessário"*.

Principais características e configurações:

- O ICAP verifica políticas que permitem configurar a ação que deve ser tomada pelo SAVSE, isto é, o arquivo pode ser verificado, verificado e deletado ou verificado e reparado [SAVSE2002].

- O SAVSE possui um sistema de arquivos proprietário, que aloca um espaço da memória RAM. Este disco virtual é utilizado para a extração e análise de arquivos compactados e empacotados, a conseqüência desta ação é uma análise mais rápida, exatamente por não ser obrigado a utilizar o disco rígido e perder tempo na utilização do barramento, nem no tempo de gravação do dispositivo rígido [SAVSE2002].
- O SAVSE implementa um tempo limite para verificação de vírus, em arquivos compactados. Essa limitação se dá, por exemplo, em relação ao tamanho do arquivo, tempo máximo para descompactação e análise, bem como ao número de níveis a ser analisado [SAVSE2002].

3.4.4 Solução Servidor Internet

Para implementação do servidor *Web*, decidiu-se utilizar o produto *Small Server*, um servidor HTTP *Freeware*, leve e fácil de ser utilizado. Ele é um simples arquivo executável, possuindo uma pasta de acesso e permitindo gerenciamento simplificado e rápido.

O sistema operacional adotado foi o *Microsoft Windows 2003 Server*, pois é um sistema operacional robusto, com *interface* amigável de rápida configuração. A escolha dessa plataforma operacional foi baseada na facilidade de seu uso e principalmente pela rapidez de sua instalação e configuração como servidor *Web*.

Um sistema dedicado foi utilizado para armazenar os arquivos para *download*, pois caso os arquivos fossem armazenado em algum dos outros servidores propostos (*Squid* ou ICAP) as medições seriam prejudicadas, afetando seu desempenho e invalidando o resultado deste Projeto. Portanto, cada máquina possui seu serviço dedicado, para que nenhum fator interfira na análise e nos resultados propostos e obtidos.

3.5 ESPECIFICAÇÕES TÉCNICAS

Atualmente, a RFC que trata do protocolo ICAP menciona apenas os métodos: OPTIONS (opção de "*query*" do Servidor ICAP), REQMOD (Modificação de Requisição - 2.4.3.1) e RESPMOD (Modificação de Resposta - 2.4.3.3). O escopo deste Projeto tem foco no uso de antivírus para verificação de arquivos, portanto, apenas o RESPMOD será utilizado.

O ICAP possui várias semelhanças com o HTTP, tanto de semântica quanto de utilização. O ICAP não é HTTP *"tão pouco um protocolo de aplicação que roda sobre HTTP [..] sua porta default é a 1344, porém outras portas podem ser utilizadas."* [ELSON2003]. Uma requisição ICAP é uma mensagem enviada de um cliente para o Servidor ICAP, uma resposta ICAP consiste em um cabeçalho e um corpo ICAP, que conterá uma mensagem HTTP encapsulada. Neste caso, a mensagem HTTP conterá uma requisição HTTP e uma resposta HTTP. Como usamos ICAP, não apenas para propósitos HTTP, o Cliente ICAP deve criar uma requisição HTTP e uma resposta HTTP forjadas. Mensagens ICAP devem estar conforme os formatos de mensagens da RFC2822 [ELSON2003].

Maiores detalhes serão explicados por meio de exemplo.

3.5.1 Exemplo de Requisição ICAP HTTP - Vírus

Na requisição, o arquivo */tmp/test.txt* deverá ser verificado por um Servidor ICAP Antivírus. O arquivo *test.txt* contém apenas a linha "TESTE TESTE TESTE" (12). O número de linhas foi colocado à frente de cada passo para demonstrar de forma mais clara o significado de cada linha da requisição.

```
01: RESPMOD icap://localhost/avscan ICAP/1.0
02: Host: localhost
03: Encapsulated: req-hdr=0, res-hdr=30, res-body=109
04:
05: GET /tmp/test.txt HTTP/1.1
06:
07: HTTP/1.1 200 OK
08: Content-Type: application/octet-stream
09: Content-Length: 14
```

```
10:
11: e
12: TESTE TESTE TESTE
13: 0
14:
```

As linhas de 01 a 14 mostram uma mensagem de requisição ICAP, que consiste de duas partes: o cabeçalho, linhas 01 a 03; e o corpo ICAP, linhas 05 a 14. A linha 04 é vazia, pois indica o final do cabeçalho ICAP. O cabeçalho inicia-se com uma linha de requisição (01), que contém: o método utilizado, neste caso, o RESPMOD; a URL do recurso ICAP e uma *string* da versão do ICAP [ELSON2003]. A URL ICAP é especificada como:

```
ICAP_URL = Scheme ":" Net_Path [ "?" Query ]
Scheme = "icap"
Net_Path = "//" Authority [ Abs_Path ]
Authority = [ userinfo "@" ] host [ ":" port ]
```

A linha de requisição é seguida por dois ou mais cabeçalhos (02-03) onde pode ser necessário conter o cabeçalho de "*Host*" [ELSON2003]. Já o cabeçalho encapsulado deve estar contido em cada mensagem ICAP [ELSON2003] e será explicado posteriormente.

O corpo ICAP contém mensagens HTTP encapsuladas. O modelo de encapsulamento ICAP é simplesmente a ação de empacotar todo o número de seções da mensagem HTTP em mensagem de encapsulamento ICAP (Corpo), a fim reservar as requisições de vetoramento, respostas, e requisição/resposta para um Servidor ICAP. Os corpos encapsulados devem ser transferidos, usando-se o método "*chunked*" que será descrito posteriormente [ELSON2003].

Antes de detalhar o corpo ICAP, segue a sintaxe para o método de transferência "*chunked*" [ELSON2003].

```
Chunked-Body = *chunk
                        last-chunk
                        trailer
                       CRLF

chunk = chunk-size CRLF
            chunk-data CRLF

chunk-size = 1*HEX
```

```
last-chunk = 1*("0") CRLF

chunk-data = chunk-size(OCTET)
trailer = *(entity-header CRLF)
```

Então, o corpo ICAP consiste em:

- Cabeçalho de requisição HTTP (linha 05). Normalmente, esta é a requisição de um *browser Web*, mas de acordo com esta aplicação, o Cliente ICAP forja um cabeçalho;
- A linha vazia (06) indica o fim da requisição HTTP;
- O cabeçalho de resposta HTTP (07-09);
- A linha vazia (10) indica o final do cabeçalho de resposta HTTP;
- O *chunk size*, representado na linha 11 [ELSON2003] é uma *string* de dígitos hexadecimal e tem o mesmo valor especificado em "*context-lengths*" para requisições HTTP;
- Os “*chunk-data*” são os dados, referentes à linha 12, a serem verificados (neste caso, o conteúdo do arquivo *test.txt*).
- O último "*chunk*", é simplesmente "0" (seguido por CRLF); e
- A linha vazia (CRLF) como um "*chunked-body*" definição da [FIELDING1999] (linha13).

Depois de analisado como é feito o encapsulamento, será verificado agora o cabeçalho encapsulado. Ele indica uma mensagem ICAP que encapsulará o cabeçalho da requisição, um grupo de cabeçalhos de resposta e um corpo de resposta.

3.5.2 Exemplo de Resposta ICAP HTTP – Vírus

Uma resposta ICAP pode ser considerada de três formas:

- Um erro;
- Uma mensagem de cabeçalho e corpo de resposta HTTP encapsulada e modificada; ou

- Uma resposta HTTP 204, o qual indica que a requisição do cliente não necessita de adaptação [ELSON2003].

A Resposta ICAP “deve iniciar com uma linha de status ICAP, similar a forma usada pelo HTTP, incluindo a versão do ICAP e o código do status” [ELSON2003].

A seguir, os códigos de erro do ICAP, que diferem dos HTTP:

- 204 - Sem necessidade de modificação;
- 400 - *Bad request*; e
- 404 - Serviço ICAP não encontrado

Segue um exemplo de Resposta ICAP como mencionado anteriormente:

```
01: ICAP/1.0 200 OK
02: ISTag: "1052324700"
03: Encapsulated: res-hdr=0, res-body=127
04:
05: HTTP/1.1 200 OK
06: Content-Type: application/octet-stream
07: Content-Length: 14
08: Via: 1.1 Symantec AntiVirus Scan Engine (ICAP)
09:
10: e
11: TESTE TESTE TESTE
12: 0
```

As linhas de 01 a 12 demonstram uma Resposta ICAP completa, que consiste em: cabeçalho ICAP (01-03), um cabeçalho da resposta encapsulado HTTP (05- 08) e finalmente, os dados que formam o corpo da resposta (10-12).

3.5.3 Extensões do ICAP

Normalmente, quando o ICAP é utilizado para verificação de vírus em tráfego HTTP, uma notificação de vírus é retornada em HTML (Resposta HTTP), que é então simplesmente mostrada no *Browser Web*. O Cliente ICAP usa o cabeçalho de informação que é enviado pelo *Symantec Antivirus*

Engine, para pegar a informação desejada. A seguir, os cabeçalhos utilizados [STECHER2003]:

```
X-Infection-Found-Header = "X-Infection-Found: Type=" TypeID
"; Resolution=" ResolutionID
"; Threat=" ThreadDescription ";"
TypeID = 0 | 1 | 2
ResolutionID = 0 | 1 | 2
ThreadDescription = TEXT
```

O *TypeID* é:

- 0 = Infecção de vírus;
- 1 = Violação da política de e-mail; e
- 2 = Violação de recipiente.

ResolutionID é:

- 0 = Arquivo não-reparado;
- 1 = Arquivo reparado; e
- 2 = Arquivo deve ser bloqueado ou rejeitado.

A "*threaddescription*" é a descrição da ameaça, como por exemplo, o nome do vírus [STECHER2003]. O Cliente ICAP desenvolvido utiliza o cabeçalho *X-Infection-Found* para adquirir o nome do vírus.

```
X-Virus-ID-Header = "X-Virus-ID:" OneLineUSTEXT
OneLineUSText = 1*( <any CHAR except CTLs> )
```

```
X-Violations-Found-Header = "X-Violations-Found:" count
1*( CR LF Filename CR LF ThreadDescription CR LF
ProblemID CR LF ResolutionID )
count = 1*DIGIT
Filename = TEXT
ThreadDescription = TEXT
ProblemID = 1*DIGIT
ResolutionID = 0 | 1 | 2
```

CAPÍTULO 4. INSTALAÇÃO DOS SERVIDORES

4.1 INTRODUÇÃO

Este Capítulo descreverá os processos de montagem, instalação e configuração dos Servidores e seus respectivos serviços.

4.2 INSTALAÇÃO DO SERVIDOR ICAP – *SCAN ENGINE*

4.2.1 Instalação Sistema Operacional Linux

Para instalação do *Linux* foram escolhidas as opções de:

- Linguagem de instalação em inglês;
- Configurações do teclado ABNT2;
- Configurações de *mouse*: *"Wheel Mouse (PS/2)";*
- *"Automatically Partition"*. Nesta opção o instalador apagou todos os dados da partição;
- *"Remove all linux partitions on this system"*, com isso o instalador removeu qualquer instalação prévia do disco. O instalador informou que ao selecionar essa opção todas as partições existentes nesse disco serão apagadas;
- Selecionado o esquema de partição automática;
- Opções *default* de *"boot loader"*;
- Em *network configurations*, a opção DHCP estava habilitada por default. Foi configurado o IP estático, mascara de rede, nome do computador, *gateway* e DNS;
- Desabilitar o *firewall*;
- Linguagem *default* do sistema em inglês;

- Horário do sistema de São Paulo;
- Configuração do *password* de *root;*
- *"Customize the set of packages to be installed"* onde foram escolhidos os seguintes pacotes:
  - *Editors;*
  - *Text Based Internet;*
  - *Development tools; e*
  - *Kernel development. e*
- Finalmente, os discos foram formatados e o sistema operacional instalado.

4.2.2 Instalação *Symantec Scan Engine*

- Inserção do CD contendo o produto *Symantec Scan Engine* e montagem do *drive* de CD:

  [root@ScanEngineLAB2 RedHat]# mount /dev/cdrom /mnt/cdrom/

- Após montagem da unidade, foi copiado o conteúdo do CD para um diretório local:

  [root@ScanEngineLAB2 RedHat]#cd /mnt/cdrom/

  [root@ScanEngineLAB2 RedHat]#cp Scan_Engine_4313-RedHat.zip /root

- Extração dos arquivos do zip:

  [root@ScanEngineLAB2 RedHat]#unzip Scan_Engine_4313-RedHat.zip

- O instalador foi executado:

  [root@ScanEngineLAB2 RedHat]#cd root/Scan_Engine/Scan_Engine/RedHat

  [root@ScanEngineLAB2 RedHat]# sh ScanEngine.sh

- Após a execução dos instaladores, foi necessário responder algumas

perguntas para configuração do SAVSE, tais como:

- Aceitação dos termos da licença;

Do you agree with the terms of this license? (default: n) [y,n,?,q] ***y***

- Especificação do diretório de instalação (*default*);

Specify the installation directory. InstallDir (default: /opt/SYMCScan) [?,q]

- Especificação do diretório de *log* (*default*);

Specify the log file directory. LogDir (default: /var/log) [?,q]

- Especificação do protocolo utilizado - Protocolo ICAP;

Specify the protocol to use:

1. NATIVE protocol

2. ICAP protocol

ProtocolChoice (default: 1) [?,q] 2

- Especificação da porta de Administração Web (*default*);

Specify the port the Administrator Web Interface will listen on AdminPort (default: 8004) [?,q]

- Especificação da senha para o administrador da *interface Web*;

Specify the password for the Administrator Web Interfac AdminPassword [?,q]

Confirm AdminPassword [?,q]

Após as respostas para configuração, a instalação foi inicializada. Para evitar o erro a seguir, é necessário que seja instalado a dependência *compat-libstdc++-7.3-2.96.126.i386.rpm*:

> ***ERROR: This copy of the SYMCScan software did not install correctly. Review the error or warning messages that were issued, take the corrective action suggested, then try again. A transcript of this installation has been***

saved as /var/log/SYMCScan-install.log.

De acordo com o erro gerado durante a instalação do produto, foi necessário satisfazer esta dependência antes de prosseguir com a instalação. Para isso instalou-se o pacote *compat-libstdc++-7.3-2.96.126.i386.rpm*:

[root@ScanEngineLAB2 root]# rpm -ivh compat-libstdc++-7.3-2.96.126.i386.rpm

Após a instalação da biblioteca, foi necessário repetir os passos de instalação do SAVSE:

[root@ScanEngineLAB2 RedHat]# sh ScanEngine.sh

Por fim, a instalação com configurações *default,* foi realizada com sucesso:

The installation of SYMCScan was successful.

SEE THE INSTALLATION GUIDE FOR FURTHER INSTRUCTIONS, including how to activate and configure the software.

A transcript of this installation has been saved as /var/log/SYMCScan-install.log.

A console *web* já pode ser acessada (*http://10.1.1.11:8004*). A Figura 4.1 mostra a console *web* com a janela de *logon* do *Symantec Scan Engine*.

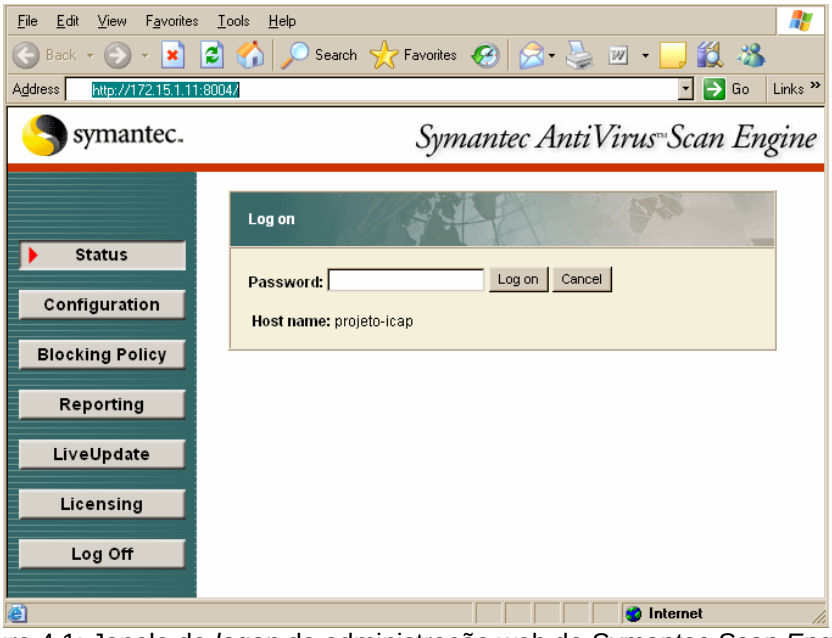

Figura 4.1: Janela de *logon* da administração web do *Symantec Scan Engine.*

4.3 INTALAÇÃO DO SERVIDOR SQUID – CLIENTE ICAP

4.3.1 Instalação do Sistema Operacional Linux

Para instalação e configuração do Sistema Operacional Linux foram adotados os mesmos passos do Servidor ICAP, referenciado na seção 4.2.1 (Instalação Sistema Operacional Linux).

4.3.2 Instalação do Servidor *Squid*

Foi feito o *download* do pacote pré-compilado, com suporte ao protocolo ICAP (*link* para *download*: *http://prdownloads.sourceforge.net/icap-server/squid-icap-client-1.2.1-src.tgz?download*) e copiado o arquivo para o servidor via WinSCP (*link* para *download*: *http://winscp.net/eng/index.php*).

Após a copia, o arquivo foi descompactado e instalado com os seguintes comandos:

```
[root@Squid root]# tar xzvf squid-icap-client-1.2.1-src.tar

[root@Squid root]# cd squid-2.6-DEVEL-20020324
```

Nesta instalação foi habilitado o suporte ao ICAP, SNMP, pois foi necessário gerar um *benchmark* do serviço e da máquina, e liberação do ICMP:

```
[root@Squid root]# ./configure --prefix=/usr/local/squid --enable-icap-support --enable-snmp --enable-icmp

[root@Squid root]# make

[root@Squid root]# make install
```

Foi necessário criar um usuário, para que este seja o responsável pela inicialização do *cache*:

```
[root@Squid root]# adduser squid
```

Logo após, foi feita a criação das pastas de *cache* e *log* para que o *Squid*

pudesse ser inicializado:

[root@Squid root]# mkdir /usr/local/squid/cache

[root@Squid root]# mkdir /usr/local/squid/logs

Foi necessário fazer a alteração das permissões dos diretórios criados acima, para o usuário *Squid*:

[root@Squid root]# chown squid.squid /usr/local/squid/cache

[root@Squid root]# chown squid.squid /usr/local/squid/logs

Sem a criação deste usuário, o *Squid* não poderia ser inicializado devido a restrições de segurança do *cache*. O próximo passo foi configurar o arquivo *squid.conf* para habilitar a comunicação ICAP com o servidor *Scan Engine* e também para adição do usuário criado.

De acordo com o proposto neste Projeto, foram utilizadas duas estruturas:

- Antivírus remoto, utilizando o ICAP; e
- Antivírus local, utilizando o *ClamAV*.

Para cada uma destas estruturas foi adicionada uma linha de configuração no arquivo *squid.conf*. O Anexo I contém o arquivo *squid.conf* configurado conforme ambos os Cenários (3.2).

4.3.3 Instalação do *Software SquidClamav*

Inicialmente foi necessário instalar o *SquidClamav*, *software* responsável pelo redirecionamento dos arquivos que sairão do *Squid* para o *ClamAV*.

Foi necessária a instalação de duas dependências:

- libcurl 7.12.1 no mínimo: e
- libidn-0.5.16.tar.gz (wget http://josefsson.org/libidn/releases/libidn-0.5.16.tar.gz)

Para a instalação dessa *lib* (biblioteca), foi executado o comando padrão

para o procedimento de instalação:

./configure && make && make install

Para o *download* do *SquidClamav* foi utilizado o comando:

http://www.samse.fr/GPL/squidclamav/

Foi necessário criar um diretório para o *SquidClamav* e após, foi criado um usuário e um grupo *squid*. Esta medida foi necessária, pois no momento da compilação, o mesmo acusou um erro ao executar o comando *make* ao informar a inexistência dos usuário e grupo, respectivamente *groupadd squid* e *useradd squid -g squid*:

root@squid:/tmp#mkdir antivirus

root@squid:/tmp#cd antivirus

root@squid:/tmp/antivirus#tar xvzf squidclamav-2.2.tar.gz

Foi necessário entrar no diretório descompactado :

root@squid:/tmp/antivirus#cd squidclamv

Posteriormente, foram seguidas as seguintes etapas:

root@squid:/tmp/antivirus/squidclamav#cd regex

root@squid:/tmp/antivirus/squidclamav/regex#./configure

root@squid:/tmp/antivirus/squidclamav/regex#make

root@squid:/tmp/antivirus/squidclamav/regex#cp regex.h regex.o ../

root@squid:/tmp/antivirus/squidclamav/regex#cd ..

root@squid:/tmp/antivirus/squidclamav#make

root@squid:/tmp/antivirus/squidclamav#make install

Foi realizado o processo de compilação e instalação do pacote. Para integração do *SquidClamav* com o *Squid* foi necessário alterar as seguintes

opções no arquivo *squid.conf* (Anexo 01) do *Proxy*.

Foram inseridas as três linhas de *redirect* abaixo da ultima ACL[30], senão poderiam ocorrer problemas durante a execução do *start* do *Squid*. As linhas são as seguintes:

redirect_program /usr/local/squidclamav/bin/squidclamav

redirect_children 15

redirector_access deny localhost

Se por acaso, houver muito acesso no *Squid* e pouca memória, será necessário reduzir o valor do *redirect_children*.

O *redirector_access* serve para previnir "*loops*[31]".

Agora foi necessário editar o arquivo: */usr/local/squidclamav/etc/ squidclamav.conf* e alterar as opções que melhor se enquadraram ao ambiente.

Após finalizada toda a configuração, foi necessário mandar o *Squid* reler o arquivo de configuração. Assim, finalmente, o *Proxy* está protegido com um antivírus.

root@squid:/#squid -k reconfigure

4.3.4 Instalação do *Clam* Antivírus

Após a instalação do *SquidClamav*, o *Squid* está apto a se comunicar com um *software* de antivírus local. Para isso basta existir um *software* antivírus local. De acordo com o mencionado, o próximo passo foi a instalação do *software ClamAV*:

#cd clamav-088

30 ACL (*Access Control List*) é uma lista de controle de acesso que define quem tem permissão de acesso a certos serviços.

31 *Loop* indica um grupo de instruções que serão repetidas.

#./configure --prefix=/usr/local/clamav

#groupadd clamav

#adduser -g clamav -s/bin/false -c"ClamAV" clamav

#make

#make install

4.4 INTALAÇÃO DO SERVIDOR INTERNET – *SMALL HTTP SERVER*

Para a instalação deste servidor, foi utilizada a ferramenta *Small HTTP Server,* versão 3.05.29, que é um pequeno servidor *Web*, que possui apenas as funções básicas de um bom servidor de páginas. Com características suficientes para os requisitos deste Projeto.

4.4.1 Instalação do *Small HTTP Server*

Como procedimentos para instalação do *Small HTTP Server* foi necessário fazer o *download* do arquivo *shttp3.exe* no link *http://smallsrv.da.ru/ shttp3.exe.* Quando executado o arquivo, é inicializada uma tela de configuração onde, além de aceito os termos da licença foi definido um *password* para o usuário de administração, selecionado a opção “NT Service” e “Add to Startup”, pois isso fará com que o serviço SHTTP suba com a inicialização do *Windows* e seja tratado como um serviço em sua console de gerenciamento. E finalmente, a instalação do *software* foi concluída.

Foram criados os seguintes arquivos para realização dos testes:

- Projeto100Mb-cv.zip;
- Projeto100Mb-sv.zip;
- Projeto50Mb-cv.zip;
- Projeto50Mb-sv.zip;

- Projeto10Mb-cv.zip; e
- Projeto10Mb-sv.zip.

Os arquivos foram criados em formato compactado ZIP, pois permitiam a sua manipulação de forma mais simples quanto ao tamanho. Para os arquivos infectados foi utilizado o arquivo de teste *Eicar*, que tem o objetivo de testar a eficiência dos mesmos. O *Eicar* é o Instituto Europeu para Pesquisa de Antivírus de Computador e foi fundado em 1990 para pesquisar melhorias para os antivírus.

A Figura 4.2 mostra a tela do *browser* acessando o servidor com os arquivos já criados.

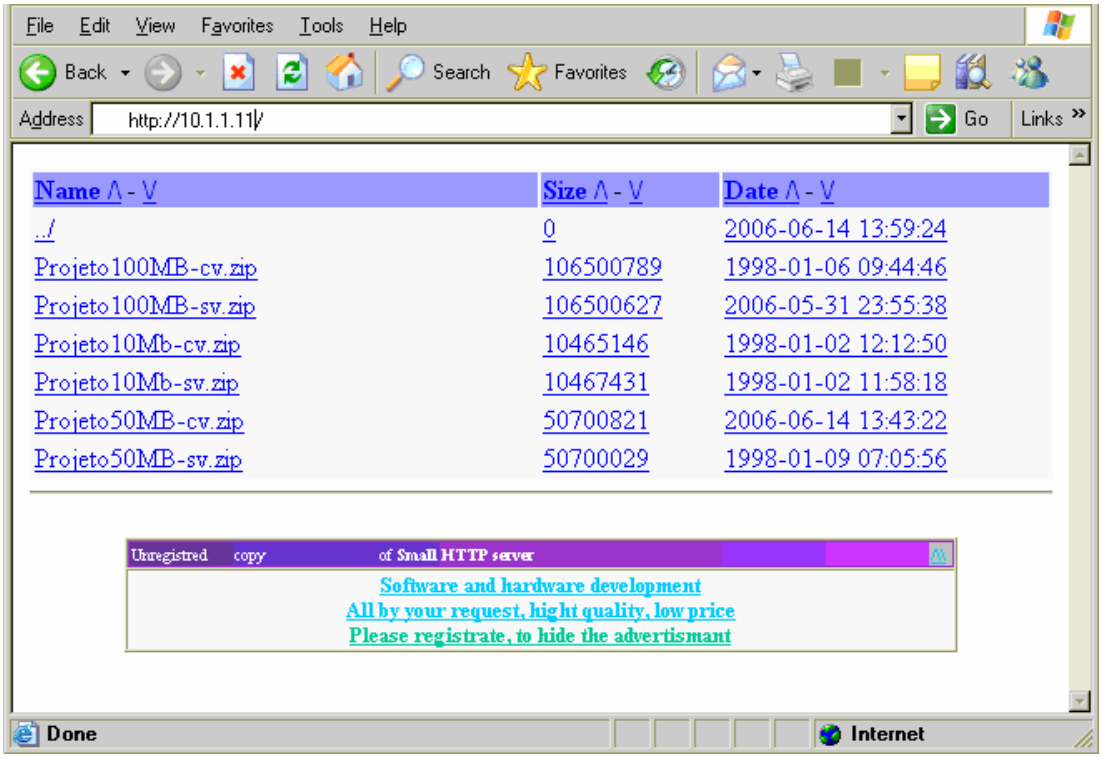

Figura 4.2: Página de *download* dos arquivos de teste.

CAPÍTULO 5. TESTES E ANÁLISE DE RESULTADOS

5.1 INTRODUÇÃO

Neste Capitulo serão mostrados os resultados que foram obtidos e analisados nos testes e simulações dos Cenários do Projeto.

Também serão citados os problemas que foram ocorrendo durante o desenvolvimento deste Projeto.

5.2 PROCEDIMENTO PARA *BENCHMARK*

Neste Projeto, os dois Cenários precisavam ser criados para efetuar a medição. Para evitar diferenciações causadas por *hardwares* distintos, a opção foi utilizar os mesmos equipamentos para ambos os cenários.

5.2.1 Procedimento padrão para ambas as medições

Para este Projeto foram analisados dois aspectos:

1) Tempo de *download* de arquivos de tamanhos variados no cliente sendo:

- Um arquivo de 10MB, outro de 50 MB e outro de 100 MB, todos sem vírus;
- Um arquivo de 10MB, outro de 50 MB e outro de 100 MB, todos com vírus.

Para cada arquivo foi feito *download* três vezes, limpando os *caches* e tirando uma média do tempo de *download*, para cada arquivo em cada situação. A tripla repetição do procedimento se faz necessária, pois os sistemas operacionais possuem rotinas e aplicativos que funcionam em *background*, efetuando tarefas paralelas com a utilização normal, pondendo causar variações no tempo do *download*. Uma média do tempo dará um resultado mais próximo do real.

2) Consumo de recursos no servidor *Proxy*, sendo:

- Uso de memória e uso de processador, no servidor *Proxy*.

5.2.2 Procedimento para Cenário I

Para o Cenário I, o *Proxy* precisava estar com quatro processos principais em execução: *Squid* (*Proxy+cache*), *ClamAV* (anti-vírus); *SquidClamav* (processo intermediário entre *Proxy* e antivírus) e TOP (medidor do consumo de recursos).

Para efetuar a medição foi necessário a elaboração de *shell script* (Anexo 04) que iniciasse o processo de captura, quando a solicitação do cliente fosse recebida pelo *Proxy* e finalizasse o processo de captura, quando o *download* fosse concluído pelo cliente.

5.2.3 Procedimento para Cenário II

Já para o Cenário II, o *Proxy* precisava estar com dois processos em execução: *Squid* (*Proxy+cache*), e TOP (medidor do consumo de recursos).

Para efetuar a medição neste Cenário, o mesmo *shell script* (Anexo 04) utilizado no Cenário I foi utilizado.

5.3 RESULTADOS OBTIDOS

De acordo com os testes descritos no subitem 5.2 (*Benchmark*) deste Capítulo foram obtidos os seguintes resultados:

5.3.1 Tempo de D*ownload*

5.3.1.1 Cenário I

Para o resultado do tempo de *download* no Cenário I são apresentadas duas tabelas: a primeira, com medições com tempo de *download* para os arquivos sem vírus (Tabela 5-1) e a segunda, para os arquivos com vírus (Tabela5-2).

Tabela 5-1: Tempo de Download Cenário I - Sem vírus.

Arquivo		**Tempo 01 (mm:ss.0)**		**Tempo 02 (mm:ss.0)**		**Tempo 03 (mm:ss.0)**	
Nome	Tamanho (Kb)	Verificação	Download	Verificação	Download	Verificação	Download
Projeto10Mb-sv	10,223	00:17.2	00:11.4	00:16.2	00:11.4	00:16.3	00:11.8
Projeto50Mb-sv	49,512	01:35.6	00:55.2	01:35.7	00:57.1	01:35.4	00:55.3
Projeto100Mb-sv	104,005	02:15.1	01:57.5	02:13.8	01:58.9	02:17.3	01:58.1

Tabela 5-2: Tempo de Download Cenário I - Com vírus.

Arquivo		**Tempo 01 (mm:ss.0)**		**Tempo 02 (mm:ss.0)**		**Tempo 03 (mm:ss.0)**	
Nome	Tamanho (Kb)	Verificação	Download	Verificação	Download	Verificação	Download
Projeto10Mb-cv	10,220	00:10.1	x	00:10.7	x	00:10.1	x
Projeto50Mb-cv	49,513	00:15.1	x	00:15.4	x	00:14.9	x
Projeto100Mb-cv	104,005	01:15.1	x	01:13.0	x	01:09.8	x

5.3.1.2 Cenário II

Para o resultado do tempo de *download* no Cenário II são apresentadas duas tabelas: a primeira, com medições com tempo de *download* para os arquivos sem vírus (Tabela 5-3) e a segunda, para os arquivos com vírus (Tabela 5-4).

Tabela 5-3: Tempo de Download Cenário II - Sem vírus.

Arquivo		Tempo 01 (mm:ss.0)		Tempo 02 (mm:ss.0)		Tempo 03 (mm:ss.0)	
Nome	Tamanho (Kb)	Verificação	Download	Verificação	Download	Verificação	Download
Projeto10Mb-sv	10,223	00:31.6	00:11.5	00:33.1	00:11.6	00:29.2	00:11.4
Projeto50Mb-sv	49,512	02:39.6	00:54.8	02:26.7	00:55.2	02:31.2	00:54.3
Projeto100Mb-sv	104,005	05:32.1	2:11.5	05:24.3	02:09.4	05:32.5	02:10.3

Tabela 5-4: Tempo de Download Cenário II - Com vírus.

Arquivo		Tempo 01 (mm:ss.0)		Tempo 02 (mm:ss.0)		Tempo 03 (mm:ss.0)	
Nome	Tamanho (Kb)	Verificação	Download	Verificação	Download	Verificação	Download
Projeto10Mb-cv	10,220	00:26.7	x	00:22.9	x	00:27.3	x
Projeto50Mb-cv	49,513	01:50.6	x	01:46.9	x	01:42.7	x
Projeto100Mb-cv	104,005	03:55.1	x	03:53.8	x	03:54.8	x

5.3.2 Consumo de Recursos

Para melhor visualização dos resultados obtidos, a apresentação dos gráficos foi dividida em duas sessões: a primeira envolve os arquivos sem vírus e a segunda, com vírus. Cada uma destas sessões foi subdivida em relação ao tamanho dos arquivos: 10Mb, 50Mb e 100Mb. Para cada uma destas subssesões existem dois gráficos: o de consumo de CPU e o de consumo de memória. E finalmente, em todos os gráficos, há a análise de consumo do Cenário I e do Cenário II.

5.3.2.1 Sem Vírus

- Arquivo 10Mb:

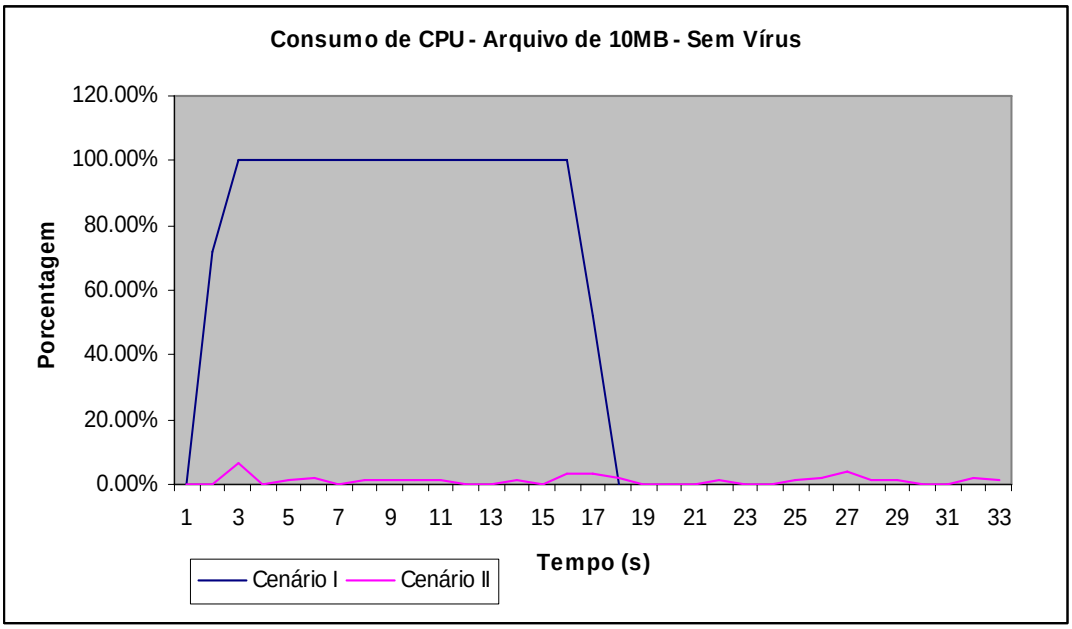

Figura 5.1: Consumo de CPU em arquivos de 10Mb, sem vírus.

Pode-se perceber que na Figura 5.1, o consumo de CPU do servidor *Proxy,* sobe no Cenário I, chegando a 100% neste período, quando ocorre a verificação de vírus no arquivo. Já no Cenário II, essa verificação é efetuada pelo Servidor ICAP, uma máquina diferente do *Proxy*. Pode ser observado que o consumo de CPU no servidor *Proxy* no Cenário II é muito baixo, em média 2%. O período de verificação do arquivo varia, entre 15 e 17 segundos para o Cenário I e 31 a 33 segundos para o Cenário II.

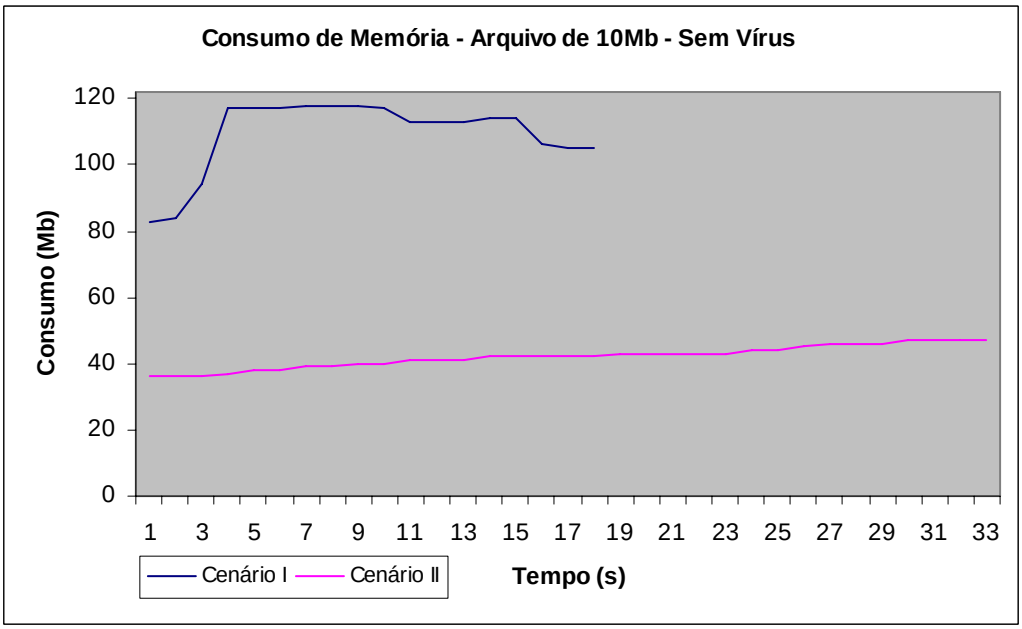

Figura 5.2: Consumo de memória em arquivo de 10Mb, sem vírus.

De acordo com a Figura 5.2, no Cenário I a média do consumo de memória no servidor *Proxy* é de 118Mb durante a verificação do arquivo. O máximo consumo de memória chega a 122Mb. No Cenário II este consumo é tem crescimento linear, pois o Cliente ICAP (Servidor *Proxy*) consome pouca memória local, enquanto aguarda a verificação do arquivo pelo Servidor ICAP.

- Arquivo 50Mb:

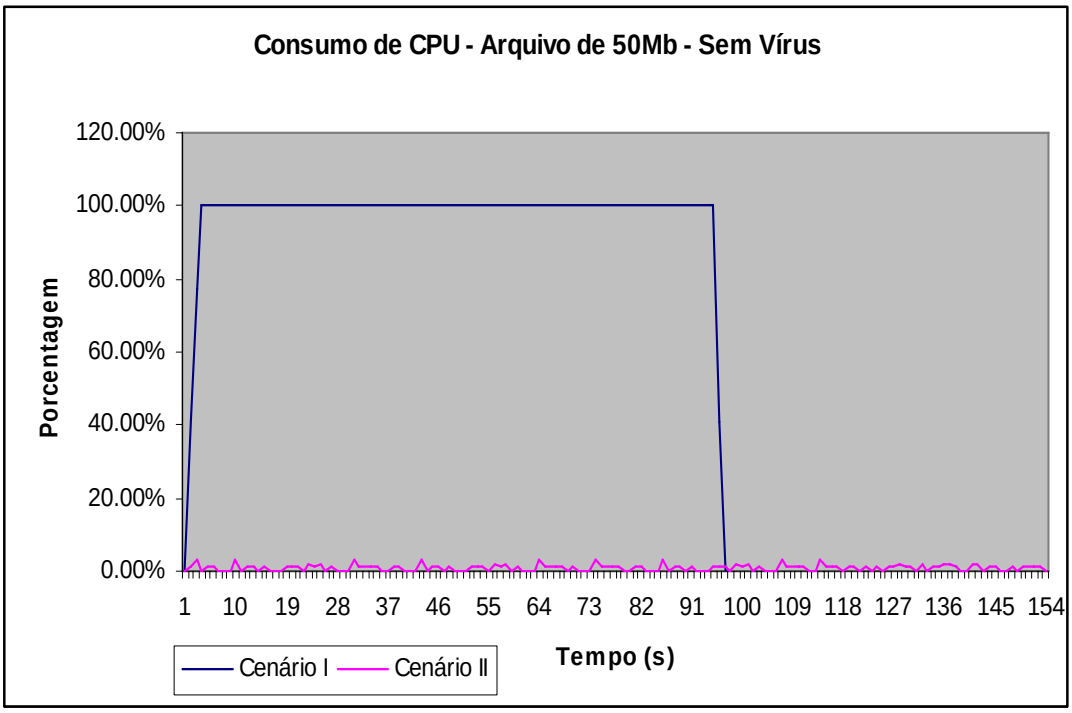

Figura 5.3: Consumo de CPU em arquivos de 50Mb, sem vírus.

Observa-se na Figura 5.3, que o *Proxy* trabalhando no Cenário I, a taxa de utilização de CPU chega a 100%, durante o período de verificação do arquivo. Já no Cenário II, de forma similar ao que ocorre no *download* do arquivo de 10Mb, a verificação é efetuada pelo Servidor ICAP, uma máquina diferente do *Proxy*. Analisando o gráfico acima é possível perceber que o consumo de CPU no Cenário II é muito baixo. O período de verificação do arquivo varia entre 153 e 155 segundos.

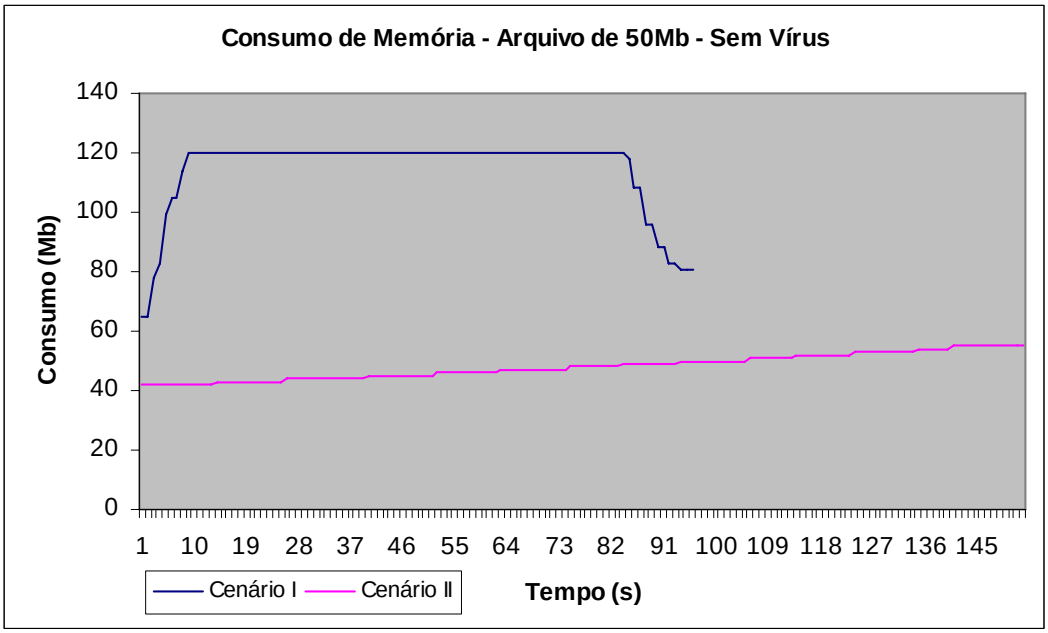

Figura 5.4: Consumo de memória em arquivos de 50Mb, sem vírus.

Observando-se a Figura 5.4, no Cenário I, o consumo de memória do servidor *Proxy* é de 120Mb durante a verificação do arquivo. O consumo máximo de memória chega a 122Mb. No Cenário II, o consumo é linear, porque o Cliente ICAP (Servidor Squid) não requer muito o uso de muita memória volátil para efetuar suas tarefas.

- Arquivo 100Mb:

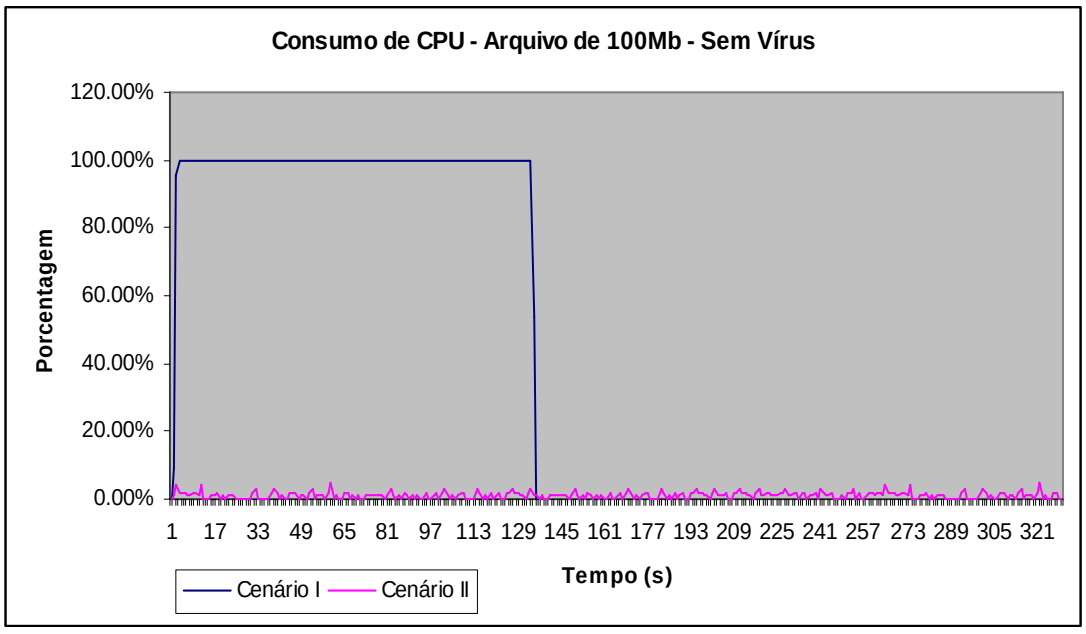

Figura 5.5: Consumo de CPU em arquivos de 100Mb, sem vírus.

Analisando-se a Figura 5.5, de forma muito similar ao comportamento com arquivos menores, o consumo de CPU chega a 100%, no período que o arquivo está sendo varrido em busca de código malicioso, que possui em média o tempo de duração de 135 segundos. Com arquivos maiores, o cenário utilizando ICAP (Cenário II), o consumo de CPU do servidor *Proxy* fica menor, por volta de 1%. O tempo de verificação do arquivo durou em média 329 segundos.

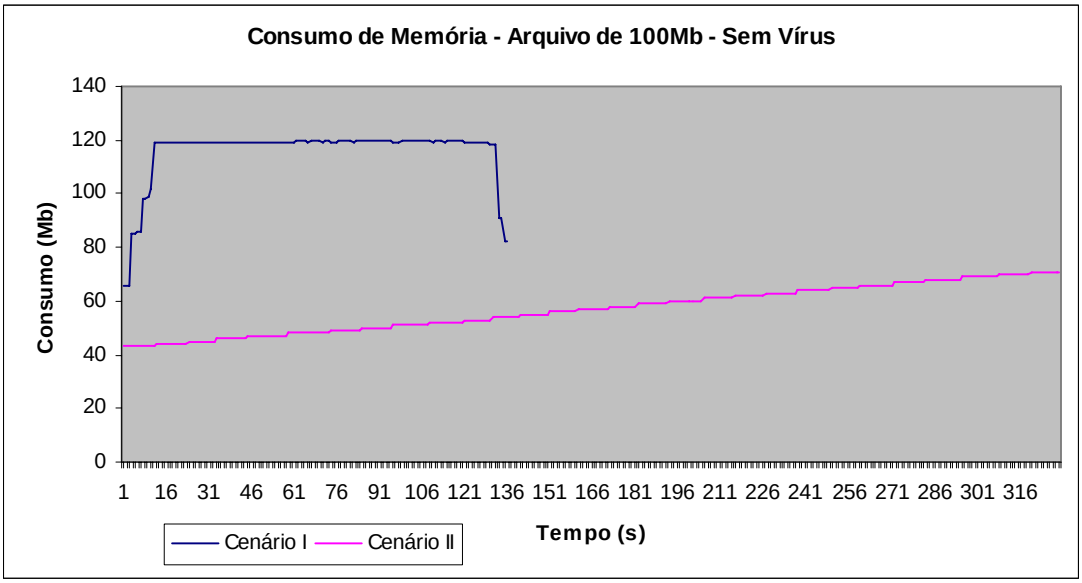

Figura 5.6: Consumo de memória em arquivo de 100Mb, sem vírus.

Similarmente às análises anteriores, na Figura 5.6, o comportamento do *Proxy* com o arquivo de 100Mb em relação ao consumo de memória foi o mesmo. No Cenário I, o consumo máximo de memória foi de 120Mb durante a verificação do arquivo. A máxima taxa de utilização chegou a 122Mb. No Cenário II, houve um crescimento linear no consumo de memória, porque o Cliente ICAP (Servidor *Proxy*) consome pouca memória local, enquanto aguarda a verificação do arquivo pelo Servidor ICAP.

5.3.2.2 Com Vírus

- Arquivo 10Mb:

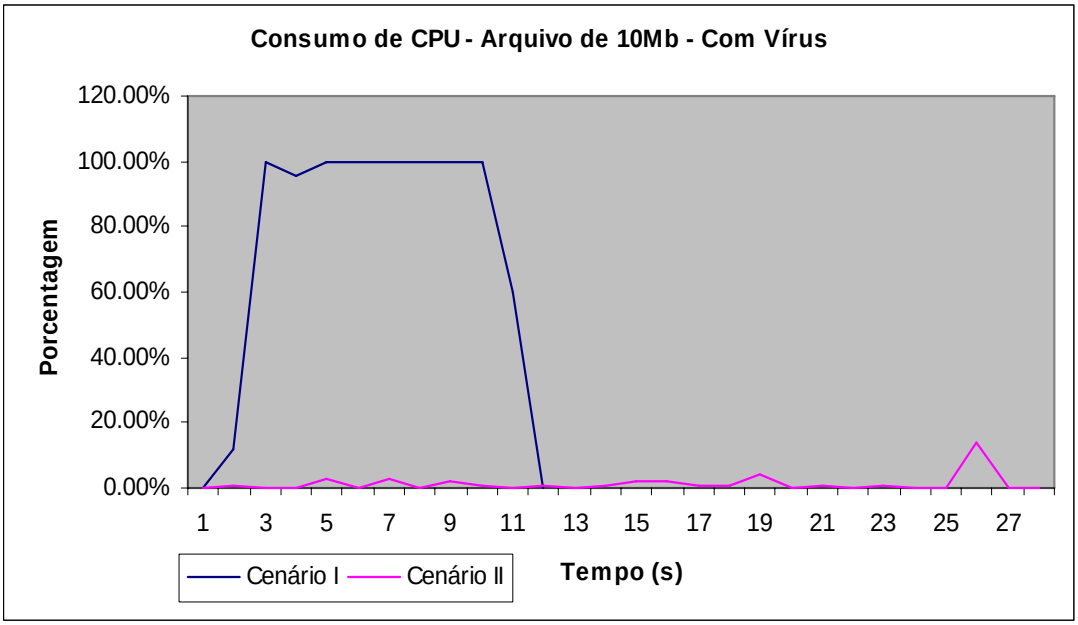

Figura 5.7: Consumo de CPU em arquivo de 10Mb, com vírus.

De acordo com a Figura 5.7, o consumo de CPU do servidor *Proxy* no Cenário I se apresentou alto e longo, chegando a 100%. Quando comparado, o consumo de CPU do arquivo com vírus e do arquivo de mesmo tamanho sem vírus, o tempo de verificação do vírus foi quase o dobro. Já no Cenário II, onde a verificação é efetuada pelo Servidor ICAP, o consumo de CPU se manteve em patamares muito baixos, em média 2%. Percebe-se, ainda, no Cenário II, um pico de consumo entre os segundos 25 e 27. Isto ocorre, pois foi neste momento que o servidor ICAP enviou uma resposta para o servidor *Proxy*, negando o acesso ao arquivo, por conter um código malicioso. O período de verificação do arquivo no Cenário II é superior ao do Cenário I, porque enquanto o Servidor ICAP faz a verificação do vírus, o *Proxy* fica em um estado de espera, aguardando a conclusão da verificação. Esse processo de envio do arquivo e espera da verificação, que ocorre no Cenário II, eleva o tempo total de *download*.

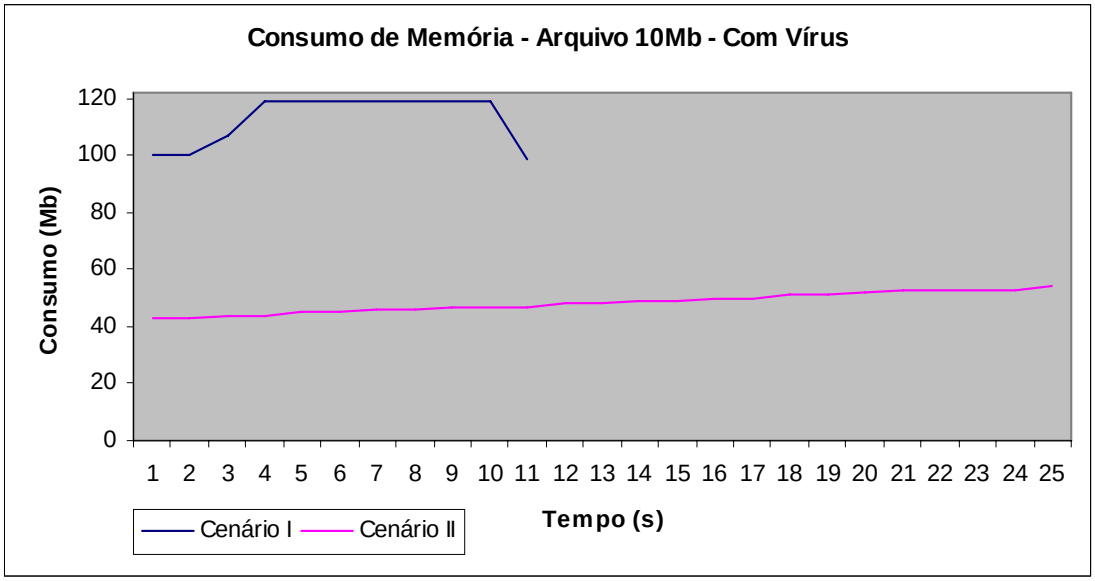

Figura 5.8: Consumo de memória em arquivo de 10Mb, com vírus.

Percebe-se que na Figura 5.8, no Cenário I, o consumo de memória no servidor *Proxy* foi 120Mb durante a verificação do arquivo, que se encerrou no segundo 11. O consumo máximo chegou a 122Mb. No Cenário II, este consumo foi inferior, porém crescente entre 43Mb e 58Mb e durou em média 25 segundos. Isto ocorreu porque enquanto o Cliente ICAP (Servidor Proxy) aguarda a verificação do vírus no Servidor ICAP, o consumo de memória apresentou crescimento linear.

- Arquivo 50Mb:

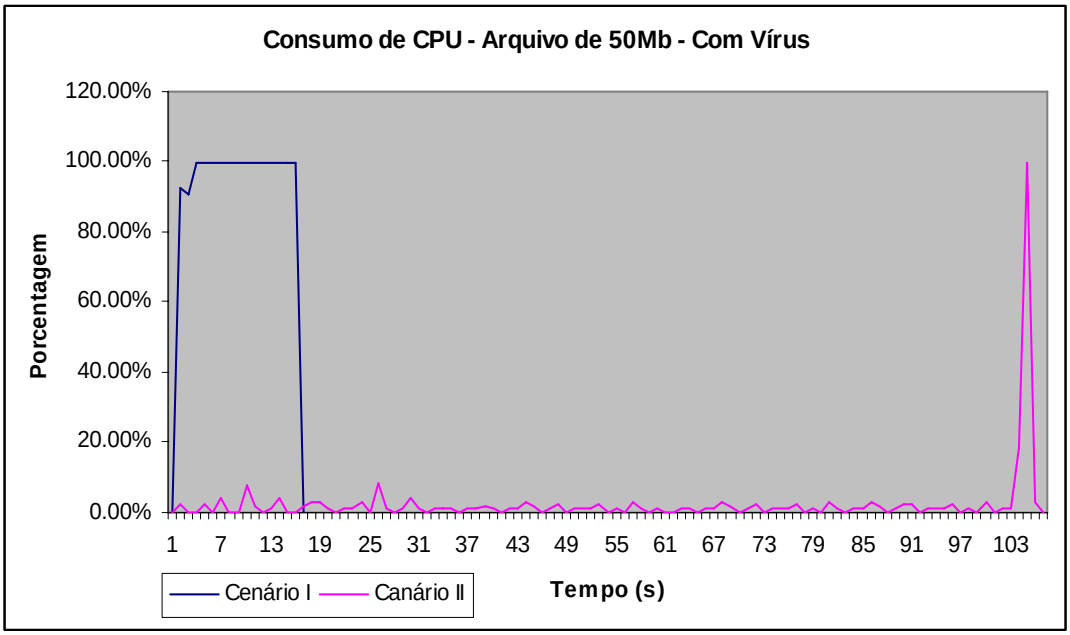

Figura 5.9: Consumo de CPU em arquivo de 50Mb, com vírus.

Na Figura 5.9, o consumo de CPU do servidor *Proxy,* no Cenário I, chega a 100%, similarmente ao arquivo de 10Mb. Já no Cenário II, onde a verificação é efetuada pelo Servidor ICAP, o consumo de CPU se manteve em patamares muito baixos, em média 2%. Percebe-se, ainda, que no Cenário II houve uma elevação no consumo, entre os segundos 103 a 106. Foi neste momento, que o servidor ICAP enviou a resposta para o servidor *Proxy*, negando o acesso ao arquivo, por conter um código malicioso. O período de verificação do arquivo foi superior ao do Cenário I, porque enquanto o Servidor ICAP efetuou a verificacao do vírus, o *Proxy* ficou em um estado de espera, aguardando a conclusão desta verificação.

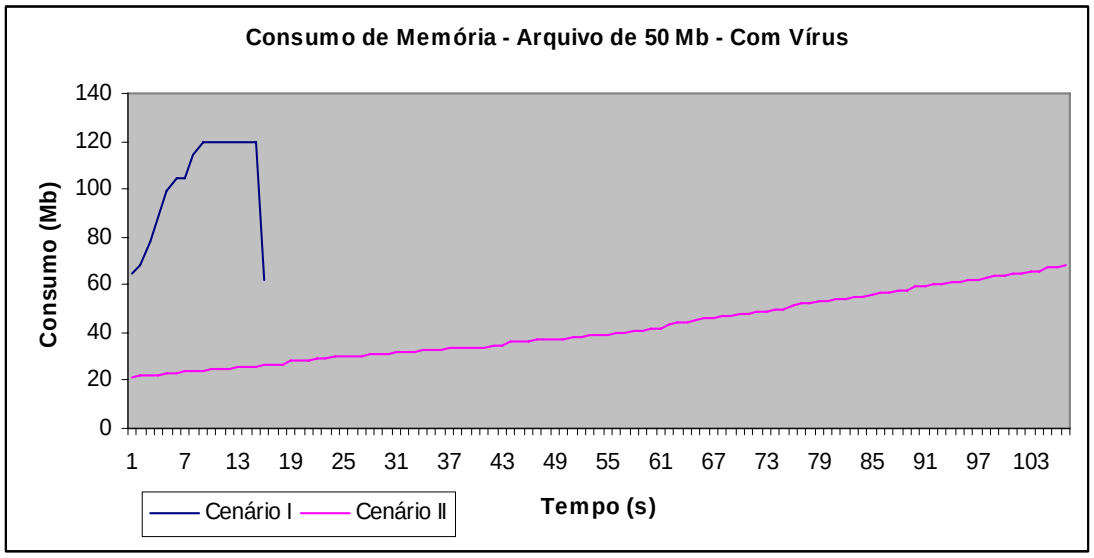

Figura 5.10: Consumo de memória em arquivo de 50Mb, com vírus.

Analisando-se a Figura 5.10, no Cenário I o consumo de memória do *Proxy* foi de 120Mb. Esse consumo elevado durou o tempo de verificação do arquivo que se encerrou no segundo 16. No Cenário II, este consumo foi inferior, porém crescente, variando entre 22Mb e 68Mb.

- Arquivo 100Mb:

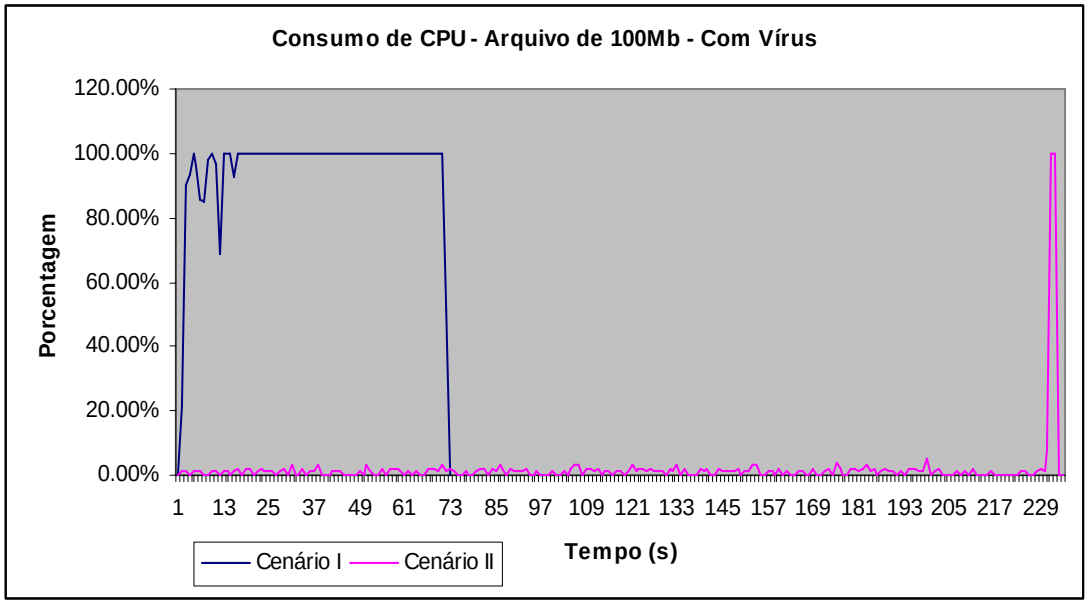

Figura 5.11: Consumo de CPU em arquivo de 100Mb, com vírus.

Na Figura 5.11, o consumo de CPU no Cenário I é extremamente alto, chegando a 100% e durou em média 72 segundos. Neste período é quando ocorre a verificação do arquivo. Já no Cenário II, onde a verificação é efetuada pelo Servidor ICAP, o consumo de CPU se manteve em patamares muito baixos, em média 02%. Percebe-se, ainda, no Cenário II, uma elevação no pico de consumo entre o período de 231 a 234 segundos, por que neste momento, o servidor ICAP enviava uma resposta para o servidor *Proxy*, negando o acesso ao arquivo, por conter um código malicioso. O período de verificação do arquivo é superior ao do Cenário I, porque enquanto o Servidor ICAP faz a verificacao do vírus, o *Proxy* fica em um estado de espera, aguardando a conclusão desta verificação.

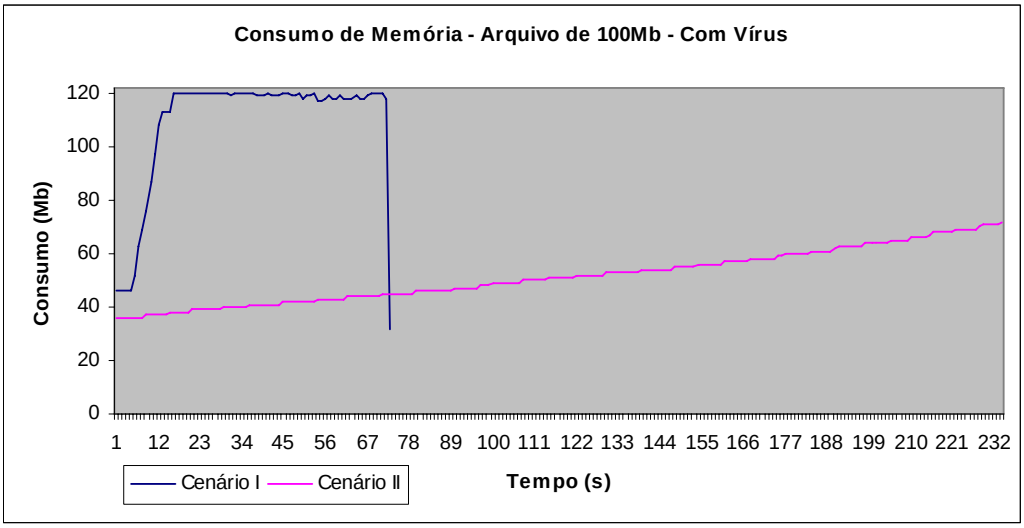

Figura 5.12: Consumo de memória em arquivo de 100Mb, com vírus.

Verifica-se na Figura 5.12, que no Cenário I o consumo de memória no servidor *Proxy* é de 120Mb, durante a verificação do arquivo que é encerrada no período de 73 segundos. O máximo consumo de memória é de 122Mb. No Cenário II este consumo é inferior, porém crescente, entre 36Mb e 72Mb. Isto ocorre porque enquanto o Cliente ICAP (Servidor Proxy) aguarda a verificação do vírus no Servidor ICAP, o consumo de sua memória cresce em intervalos de tempos iguais.

5.4 ANÁLISE DE RESULTADOS

5.4.1 Tempo de Download

Tendo como base a média dos três tempos da verificação de vírus para cada tamanho de arquivo pode-se formar a seguinte tabela:

Tabela 5-5: Média do tempo de verificação dos arquivos (mm:ss.0)

Média de Verificação	**Cenário I**		**Cenário II**	
Arquivo	**Sem Vírus**	**Com Vírus**	**Sem Vírus**	**Com Vírus**
10Mb	00:16.5	00:10.3	00:31.4	00:25.6
50Mb	01:35.6	00:14.5	02:32.8	01:46.7
100Mb	02:15.4	01:12.7	05:29.3	03:54.6

De acordo com a Tabela 5-5, percebe-se que o Cenário II possui um tempo maior de análise (verificação) dos arquivos, em relação ao Cenário I. Isto ocorre devido ao fato do Servidor *Proxy,* no Cenário II, ao invés de guardar em disco, envia o arquivo para o Servidor ICAP, fazendo com que ocorra um *download* e um *upload* do arquivo (*Proxy*-ICAP-*Proxy*), aumentando assim o tempo total de verificação. Já no Cenário I, a verificação é feita somente no Servidor *Proxy*.

Percebe-se, ainda, que o tempo de verificação dos arquivos infectados foi inferior ao dos arquivos sem vírus. Isto ocorre porque os sistemas de antivírus verificam primeiro o diretório raiz do arquivo .zip. Sendo assim, como os vírus estão contidos no diretório raiz, eles então são encontrados mais rapidamente, do que se o antivírus tivesse que percorrer ate o final de um diretório, como por exemplo, mp3. Com isso, o scaner ao encontrar o vírus, nega imediatamente o acesso ao arquivo. Assim, a verificação e a negação de acesso ao arquivo contaminado é mais rápida do que nos arquivos sem vírus.

5.4.2 Consumo de Recursos

5.4.2.1 Sem Vírus

Nesta seção, todos os arquivos estavam sem vírus. Fazendo a análise, de acordo com a divisão de cenários, tem-se como Cenário I aquele composto do servidor *Proxy* e o antivírus local, e tendo o Cenário II com o Cliente ICAP e o Servidor ICAP.

No Cenário I, o consumo de CPU no servidor *Proxy*, foi extremamente alto, com isso a CPU encontrou-se ocupada durante a maior parte do tempo de *download* do arquivo, apenas com o processamento da requisição do *Proxy* + antivírus local. Caso ocorresse a coexistência de algum outro serviço importante, o mesmo provavelmente estaria comprometido, pois seria incapaz de passar uma resposta em tempo real, ou ainda, esse consumo excessivo de CPU poderia comprometer a execução do mesmo.

Já no Cenário II, o Cliente ICAP, máquina responsável pelo *cache*, não foi sobrecarregada com as requisições do cliente, podendo operar de maneira normal e estável. Pode-se observar, também, que a memória utilizada tanto pelos processos locais quanto pelas requisições iniciadas são altas, e em alguns momentos, chega a atingir o limite da memória física, o que compromete o desempenho da máquina, obrigando o sistema operacional a requisitar a utilização de SWAP. Neste caso, uma memória estendida, que é alocada temporariamente em disco.

Pode-se notar a grande diferença no consumo de recursos entre o Cenário I e II, no qual o consumo de recursos para a verificação de códigos maliciosos, de forma remota é extremamente baixa, com a utilização do protocolo ICAP. O protocolo ICAP permite, caso seja necessário, utilizar mais de um equipamento com outras funções, como por exemplo, a função de verificar arquivos contra vírus. Essa característica permite agregar funcionalidades e distribuir a carga de processamento na execução da tarefa, tornando a solução como um todo, escalonável em qualquer ambiente.

5.4.2.2 Com Vírus

A segunda coleta de dados foi efetuada utilizando o mesmo ambiente anterior (Cenário I e II). Entretanto, desta vez, os arquivos continham amostras de código malicioso. Nesta medição pretendeu-se avaliar o comportamento do servidor *Proxy* em relação a arquivos contaminados.

No Cenário I, o consumo de processamento para a máquina com antivírus local era superior em tempo de verificação contra vírus. O consumo de CPU extremamente alto é desaconselhável, caso coexista na mesma máquina outros serviços.

Para o Cenário II, o consumo da CPU do servidor *Proxy* (Cliente ICAP) se manteve sempre em patamares entre 2% e 4%. Em apenas alguns segundos foram atingidos picos. Os picos ocorreram porque o Servidor ICAP

enviava uma resposta para o servidor *Proxy,* negando o acesso ao arquivo, por conter um código malicioso, diferente do Cenário I em que na totalidade da execução do *download* é notado que no servidor *Proxy* o processo de verificação de vírus atinge constantemente 100% de consumo.

O Cenário II possui um tempo muito maior de consumo de memória no servidor Proxy, isso acontece, pois o *Proxy* faz o download do arquivo, e em vez de guardar em disco, envia o arquivo para o servidor ICAP. Esse procedimento consome mais recursos de memória, e escrita temporária em disco. Enquanto o Servidor ICAP faz a verificação do vírus, o *Proxy* fica em um estado de espera, aguardando a conclusão desta verificação. Uma vez verificado o arquivo, o *Proxy* responde ao cliente.

Quanto ao consumo de memória, como o tempo de espera do servidor *Proxy* no Cenário II, este apresenta consumo linear e libera a CPU para utilização em outros processos, isso tudo devido ao *delay* entre Servidor *Proxy* – Servidor ICAP.

O administrador de rede que optar por utilizar o ICAP deverá avaliar a questão do perfil do usuário final, pois os *downloads* irão ficar ligeiramente mais demorados.

5.5 PROBLEMAS

Durante a fase de instalação das soluções, encontrou-se dificuldade para efetuar a configuração nos servidores: Cliente e ICAP. Verificou-se, ainda, a falta de uma documentação específica, resultando assim, na necessidade de instalação de módulos específicos no *Squid* para a preparação do ambiente.

Para a montagem do ambiente, ocorreram dificuldades técnicas na instalação do *Squid,* conflito de *hardware* e a falta de biblioteca para o correto funcionamento da placa de rede com o sistema operacional (excesso de perda de pacotes).

Durante a execução dos testes, verificou-se a falta de aplicação específica para mensurar de forma precisa os dados capturados. Houve ainda, a necessidade de desenvolvimento de *script* para efetuar a captura de forma adequada para a medição mais precisa.

Por fim, foi necessária a instalação de duas versões de Squid. A primeira, uma versão de desenvolvimento para funcionar com o ICAP e a segunda, uma versão estável para funcionar com o *Clamav*. A versão desenvolvimento era uma versão incompleta e apresentava erro na biblioteca LIBCURL.

CAPÍTULO 6. CONCLUSÃO

Percebe-se que o estudo sobre *malware* é bastante complexo, por se tratar de uma área em constante desenvolvimento. De todos os problemas que são encontrados na área de TI, poucos são tão perturbadores e caros, como os ataques dos *malwares* e os custos associados ao tratamento dos mesmos. As compreensões de como eles funcionam, como evoluem sobre o tempo, e os tipos de ataque que exploram, ajudaram a tratar sua infecção.

Os malwares utilizam-se de muitas técnicas de criação, distribuição e exploração de sistemas computadorizados. Tornando difícil de se detectar se um sistema pode ser seguro o bastante para suportar tais ataques. Entretanto, uma vez que os riscos e suas vulnerabilities foram compreendidas, é possível controlar o ambiente de rede, de maneira que um ataque possa ser gerenciado e controlado.

A proposta inicial deste Projeto era analisar o Protocolo ICAP e verificar sua efetiva redução do nível de utilização do processamento e alocação de memória em um servidor *Proxy.*

O ICAP é um protocolo não-proprietário, tendo suas especificações para implementação, detalhadas em RFC. Por ser um protocolo novo, poucos desenvolvedores acrescentam compatibilidade ao ICAP, em suas ferramentas. Das ferramentas que possuem suporte ao ICAP, a grande maioria o utiliza como protocolo para verificação e troca de mensagens de vírus. Todos os grandes desenvolvedores de ferramenta de antivírus já adicionaram a funcionalidade de uso do ICAP. Por ser um protocolo de código aberto, os fabricantes que o estão utilizando, adicionaram a capacidade para balanceamento de carga, uma vez que esta capacidade não está definida na RFC.

Os fabricantes que implementaram o ICAP, o fizeram baseando-se em requisições HTTP. Todavia, se o ICAP for utilizado em um cenário não-HTTP, o cliente ICAP necessitará de adaptações para efetuar o tratamento

de vírus. Exemplo é o SMTP, onde cada fabricante utiliza o ICAP de forma incompatível entre as soluções.

O ICAP possui pouca documentação disponível, essa característica dificulta sua utilização, principalmente em ambientes com soluções em código livre. Para efetuar sua integração com outras ferramentas, o que tem sido adotado ao redor do mundo, é a utilização de *scritps*: *shell* e *perl*. De uma maneira geral, o processo inicial de instalação e configuração é de certa complexidade e requer conhecimento avançado do ambiente onde o ICAP será instalado. Esta característica, de uma maneira geral, encarece a solução.

Neste estudo a expectativa era que houvesse redução no uso de CPU e memória do servidor *Proxy*. Analisando os resultados, pode-se perceber que o ICAP efetivamente reduziu o consumo de CPU e de memória do servidor *Proxy*. Entretanto, pode-se observar que ocorreu um aumento significativo no tempo de *download* para o usuário final. Por exemplo, um *download* que sem a utilização do ICAP levava 98 segundos, com o ICAP passou a levar 155 segundos, um aumento de 63,22%.

Por fim, embora o ICAP efetivamente reduza o consumo de recursos do servidor *Proxy*, ele apresentou desvantagem em relação ao tempo de duração do *download*, e esse tipo de comportamento pode dificultar a utilização do ICAP, num ambiente onde velocidade de *download* seja requisito de negócio.

Como sugestão para o desenvolvimento de projetos futuros poderia ser efetuada a análise do ICAP com outros serviços, além de HTTP. Além disto, existe a possibilidade de implementação dos outros módulos do ICAP, como por exemplo:

- Mostrar o redirecionamento de uma requisição não autorizada ou restrita para uma outra página, realizando assim o filtro de conteúdo;
- Tradução de linguagens (PDA’s e celulares), permitindo a dispositivos

sem suporte a HTML, como os celulares, de falar com dispositivos HTML, como os PCs e vice-versa;

- Fazer o direcionamento de propagandas de acordo com o perfil do cliente;
- Tradução de conteúdo formatado em HTML, de uma linguagem para outra;
- A compactação de páginas HTML de um servidor de origem;
- Fazer uma análise estatística dos resultados obtidos; e
- Por fim, fazer a implementação de segurança em ambiente de borda de rede, com a utilização do protocolo ICAP em servidores de arquivo e *e-mail*.

REFERÊNCIAS BIBLIOGRÁFICAS

COHEN, Fred. **Computer Viruses - Theory and Experiments.** Capítulo 2 - Computer virus. 1984 *http://www.all.net/books/virus/part2.html* - Out. 2005

CERT.br – **Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil**. *http://www.cert.br/* - Jan. 2006

ELIAS, Vinícius Graciano. **Servidor de E-mail - Postfix/Amavisd-new/ SpamAssassin/ClamAV.** Brasil, 2005. *http://www.ginux.ufla.br/ documentacao/monografias/mono-ViniciusElias.pdf* - Mar.2006

ELSON, J. **RFC3507 - Internet Content Adaptation Protocol (ICAP).** Abril 2003. *http://www.ietf.org/rfc/rfc3507.txt* - Jun 2005

FIELDING, R. **RFC: 2616: Hypertext Transfer Protocol – HTTP.** Versão 1.1. Junho 1999*. ftp://ftp.rfc-editor.org/in-notes/rfc2616.txt* - Mar. 2006

HARRISON, Richards. **The Antivirus Defense-in-Depth Guide.** Editora Volt Information Sciences. 2004

HASSELL, Jonathan. **Learning Windows Server 2003.** Editora OREILLY & ASSOC. 2ª Edição, 2006 .

HORSTMANN, Ralf. **Squid ICAP Client Configuration.** *http://icap-server.sourceforge.net/icap-configuration.html.* - Fev. 2006

LUOTONEN, Ari. **Web Proxy Servers.** Editora Prentice Hall. 1ª Edição - 1998 - 416 pág.

MARCELO, Antonio. **SQUID - Configurando o Proxy para Linux.** Editora Brasport. 4ª Edição - 2005 - 96 pág.

NEGREIROS, Ana. **A Mudança na Natureza do Crime.** Pesquisa de Segurança B2B - IBM América Latina, 2006

NETWORK Appliance. **Internet Content Adaptation Protocol (ICAP).** Versão 1.01, 7/30/01. *http://www.i-cap.org/docs/* - Jun. 2005

ROHDE, Laura. **Windows Dominates on the Desktop.** IDG News Service, 2003.

STECHER, Martin. **ICAP Extensions. Abril, 2003.** *http://www.martin-stecher.de/draft-stecher-icap-subid-00.txt* - Jun. 2005

SYMANTEC Corporation. **Manual Symantec Antivirus Scan Engine. Administrator's and Developer's Guide.** Versão 4.0, 12/2002. Symantec Corporation, 2000-2002 - Jan. 2006

SZOR, Peter. **The Art of Computer Virus Research and Defense.** Terceira Edição. Symantec Press, 2005

TANENBAUM, Andrew**. Rede de Computadores.** 3ª Edição. Editora Campos Ltda, 1997.

WESSELS, Duane. **Squid: The Definitive Guide.** Primeira Edição, Janeiro 2004, Pgs 464. *http://squidbook.org/.* - Fev. 2006

ANEXO 1 – SQUID.CONF

O *squid.conf* é o arquivo responsável pelas configurações do *software Squid*.

Segue abaixo, o arquivo utilizado neste Projeto para o Cenário I, sem a utilização do protocolo ICAP, utilizando o software Clamav localmente.

```
http_port 3128
acl QUERY urlpath_regex cgi-bin \?
no_cache deny QUERY
cache_mem 50 MB
maximum_object_size 10240 KB
minimum_object_size 100 KB
cache_dir ufs /usr/local/squid/cache 100 16 256
cache_access_log /usr/local/squid/logs/access.log
cache_log /usr/local/squid/logs/cache.log
cache_store_log /usr/local/squid/logs/store.log
# ftp_user root@localhost.localdomain
emulate_httpd_log off
client_netmask 255.0.0.0
hosts_file /etc/hosts
visible_hostname projeto-squid
auth_param basic children 5
auth_param basic realm Squid proxy-caching web server
auth_param basic credentialsttl 2 hours
refresh_pattern ^ftp:         1440  20%   10080
refresh_pattern ^gopher:      1440  0%    1440
refresh_pattern .             0     20%   4320

# ACCESS CONTROLS
# -----------------------------------------------------------------
acl all src 0.0.0.0/0.0.0.0
acl localhost src 127.0.0.1/255.255.255.255
# acl allowed_hosts src 10.1.1.20/255.255.255.255
acl FTP proto FTP
acl ftpusers src 10.1.1.5/255.255.255.255
#always_direct allow FTP
http_access allow all
http_access allow ftp ftpusers
#http_access deny ftp
cache_effective_user squid
cache_effective_group squid

# ICAP OPTIONS
# -----------------------------------------------------------------
#icap_enable on
#icap_send_client_ip on
#icap_service service_2 respmod_precache 0
icap://10.1.1.11:1344/respmod
#icap_class class_1 service_2
#icap_access class_1 allow all
#query_icmp on

redirect_program /usr/local/squidclamav/bin/squidclamav
redirect_children 15
redirector_access deny localhost
```

```
#SNMP
#  ------------------------------------------------------------------
# snmp_port 3401
#Example:
# snmp_access allow snmppublic localhost
# snmp_access deny all
```

Segue abaixo, o arquivo utilizado neste Projeto para o Cenário II, com a utilização do protocolo ICAP comunicando com o Symantec Scan Engine.

```
http_port 3128
acl QUERY urlpath_regex cgi-bin \?
no_cache deny QUERY
cache_mem 50 MB
maximum_object_size 10240 KB
minimum_object_size 100 KB
cache_dir ufs /usr/local/squid/cache 100 16 256
cache_access_log /usr/local/squid/logs/access.log
cache_log /usr/local/squid/logs/cache.log
cache_store_log /usr/local/squid/logs/store.log
ftp_user root@localhost.localdomain
emulate_httpd_log off
client_netmask 255.0.0.0
hosts_file /etc/hosts
auth_param basic children 5
auth_param basic realm Squid proxy-caching web server
auth_param basic credentialsttl 2 hours
refresh_pattern ^ftp:         1440  20%   10080
refresh_pattern ^gopher:      1440  0%    1440
refresh_pattern .             0     20%   4320

# ACCESS CONTROLS
# ------------------------------------------------------------------
acl all src 0.0.0.0/0.0.0.0
acl localhost src 127.0.0.1/255.255.255.255
acl allowed_hosts src 10.1.1.20/255.255.255.255
acl FTP proto FTP
always_direct allow FTP
http_access allow all
cache_effective_user squid
cache_effective_group squid

# ICAP OPTIONS
# ------------------------------------------------------------------
icap_enable on
icap_send_client_ip on
icap_service service_2 respmod_precache 0
icap://10.1.1.13:1344/respmod
icap_class class_1 service_2
icap_access class_1 allow all
query_icmp on

#SNMP
#  ------------------------------------------------------------------
# snmp_port 3401
#Example:
```

```
# snmp_access allow snmppublic localhost
# snmp_access deny all
#Default:
# snmp_access deny all
# snmp_incoming_address 0.0.0.0
# snmp_outgoing_address 255.255.255.255
#coredump_dir /usr/local/squid//var/cache
```

ANEXO 2 – CLAMD.CONF

O *clamd.conf* é o arquivo responsável pelas configurações do *software Clamav.*

```
[root@projeto-squid etc]# vi clamd.conf
##
## Example config file for the Clam AV daemon
## Please read the clamd.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 daemon.
# A full path is required.
# Default: disabled
LogFile /tmp/clamd.log
# Also log clean files. Useful in debugging but drastically
increases the
# log size.
# Default: disabled
#LogClean

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

# Specify the type of syslog messages - please refer to 'man
syslog'
# for facility names.
# Default: LOG_LOCAL6
#LogFacility LOG_MAIL

# Enable verbose logging.
# Default: disabled
LogVerbose

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

# Optional path to the global temporary directory.
# Default: system specific (usually /tmp or /var/tmp).
TemporaryDirectory /var/tmp

# Path to the database directory.
# Default: hardcoded (depends on installation options)
#DatabaseDirectory /var/lib/clamav
#DatabaseDirectory /usr/local/clamav/lib

# The daemon works in a local OR a network mode. Due to security
```

```
reasons we
# recommend the local mode.

# Path to a local socket file the daemon will listen on.
# Default: disabled
#LocalSocket /tmp/clamd
```

```
# Remove stale socket after unclean shutdown.
# Default: disabled
#FixStaleSocket

# TCP port address.
# Default: disabled
TCPSocket 3310
"clamd.conf" 289L,7502C                                35,1 Top
```

ANEXO 3 – SQUIDCLAMAV.CONF

O *squidclamav.conf* é o arquivo responsável pelas configurações do *software SquidClamav.*

```
[root@projeto-squid etc]# vi squidclamav.conf
# squidclamav.patterns.dist
#
# The ordering of lines is this file is critical
#

# Lines have the form:
#
#       regex|regexi pattern
#
#       abort|aborti pattern
#
#       redirect cgi_redirect_url
#
#       logfile /path/to/log_file
#
#       proxy http://127.0.0.1:3128
#
#       debug 0|1
#
#       force 0|1
#
#       timeout secondes
#
#       clamd_ip 127.0.0.1
#
#       clamd_port 3310
#
#       clamd_local /tmp/clamd
#
# Examples of valid lines:
#
# proxy http://127.0.0.1:3128/
# logfile /usr/local/squidclamav/logs/squidclamav.log
# redirect http://proxy.domain.com/cgi-bin/clwarn.cgi
# debug 0
# force 1
# clamd_ip 192.168.1.5
# clamd_port 3310
# timeout 60
# abort ^.*\.gz$
# abort ^.*\.bz2$
# abort ^.*\.pdf$
# abort ^.*\.js$
# abort ^.*\.html$
# abort ^.*\.css$
# abort ^.*\.xml$
# abort ^.*\.xsl$
# abort ^.*\.js$
```

```
# abort ^.*\.ico$
# aborti ^.*\.gif$
# aborti ^.*\.png$
# aborti ^.*\.jpg$
# aborti ^.*\.swf$
# content ^.*application\/.*$

proxy http://10.1.1.12:3128
logfile /usr/local/squidclamav/logs/squidclamav.log
redirect http://www.yahoo.com.br
debug 1
clamd_ip 10.1.1.12
"squidclamav.conf" 83L, 1490C                         58,1          Top
```

ANEXO 4 – SHELL SCRIPT

O *script-cpu* é o arquivo responsável pelo processo de captura do consumo de CPU enquanto é feita a verificação e download do arquivo.

```
#!/bin/bash

if test -d resultado-cpu
then
echo "OK - Resultado CPU existe"
else
mkdir resultado-cpu
if
iostat -c 1 $1 -t > resultado/resultado-cpu$1

~
~
~
"script-cpu" 10L, 151C                                9,1           All
```

O *script-mem* é o arquivo responsável pelo processo de captura do consumo de memória enquanto é feita a verificação e download do arquivo.

```
#!/bin/bash

free -mo -s1 -c$1| cut -c1-40 > resultado/resultado-mem$1

~
~
~
"script-mem" 4L, 73C                                  3,1           All
```