INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

UN MODELO DEL SISTEMA INMUNE PARA PREVENIR Y ELIMINAR LOS VIRUS INFORMÁTICOS.

T E S I S

QUE PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:

Juan Carlos Sarmiento Tovilla

DIRECTOR DE TESIS

Dr. Bárbaro Jorge Ferro Castro

CODIRECTOR

Dr. Agustín Francisco Gutiérrez Tornés

MÉXICO D.F., 2003

Agradecimientos

A Dios, por darme amor.

A mi hija Karla, ya que tu has sido la inspiración que le dio luz a mi vida.

A mi esposa Karla, por haberme ayudado a que esto tuviera *sentido*.

A mi mamá, quien es un ejemplo de amor, paciencia y sabiduría.

A mi tía Lupita, quien con sus consejos y su apoyo me enseño a llevar a buen termino las cosas.

A mi tía Linita, gracias por ayudarme a ver más allá de las fronteras, donde solamente la imaginación toma forma.

A mi suegros, Irene y Cesar quien con su gran paciencia y buen ejemplo me han ayudado a que esto sea una etapa más en mi vida.

A mis hermanos Jorge, Marco, Migue y Maggy quienes con su dedicación y apoyo me han ayudado en todo momento.

Al Dr. Bárbaro Ferro, por su exhortación y paciencia.

Al Dr. Gutierréz Tonéz, por su apoyo incondicional.

A los miembros de la Comisión Revisora, por sus valiosas sugerencias.

A Leticia Tate Morales por la amistad y apoyo brindado.

Índice

Índice de figuras

Índice de tablas

Índice de programas

Resumen

En esta tesis, se presenta un modelo del sistema inmune para prevenir y eliminar los virus informáticos en ambientes locales y distribuidos. Lo anterior se realiza con base a los análisis de la velocidad de propagación en diferentes topologías. Para la elaboración de este modelo se tomó como base la analogía de la señal de peligro, que genera el sistema inmune de los vertebrados.

Para realizar este modelo, se llevó a cabo un estudio de los virus informáticos, analizando y definiendo su comportamiento bajo una Máquina de Turing Universal. En este estudio, se presentan varios problemas intrínsecos al momento de realizar una detección. Se propone entonces una solución que consiste en realizar una representación abstracta de los programas de cómputo. Estas representaciones serán utilizadas como firmas de identificación junto a una reconstrucción de estructuras principales. Todas estas firmas generadas y alteradas serán enviadas a una central llamada Servidor T, en la que se realiza un diagnóstico para obtener una identificación de propagación que permite predecir el riesgo o un brote de alguna infección. Posteriormente, en el momento en que el virus ha sido identificado será enviado a la central denominada Servidor B, en ella se diagnostica o crea la vacuna apropiada para erradicar el virus. Tanto la identificación como la vacuna se realiza en ambientes controlados y conforman el llamado sistema inmune.

El desarrollo del modelo se ha dividido en dos partes:

- La identificación y eliminación de un virus informático basada en una representación abstracta de las estructuras de control.
- Un simulador visual de granularidad fina que permite experimentar y predecir los riesgos de un virus en diferentes topologías.

Las herramientas antes mencionadas se basan en patrones de diseño y fueron representados en UML, por lo que permiten una mejor reusabilidad y fortalecen la experimentación con nuevas formas de virus informáticos.

Abstract

In this thesis is presented, an immune system to prevent and eliminate the computer viruses in local and distributed environments. This goal is achieved by analizing the propagation speed in different topologies. This model, is based upon the sign of danger, which is generated by the immune vertebrate system.

To elaborate the model, a study of computer viruses was realized. Its behavior was analyzed and defined under a Turing Universal Machine. In the study, several intrinsic problems appeared at the detection moment. A solution consisting in the realization of an abstract representantion of the computing program was then proposed. These representations will be used as identification signatures together with a reconstruction of main structures. All these generated and upset signatures will be sent to a head office called T Server. Here a diagnosis is executed to obtain a spread curve that allows predicting the risk or an outbreak of some infection. Later, at the moment in which the computer virus has been identified, it is sent to another server call B. Here the vaccine locates or creates the appropiated vaccine to eradicate the virus. Both the identification and the vaccine are done in controlled environments, and they shape the so called immune system.

The development of the model has been divided in two parts:

- The identification and elimination of a virus, is based upon an abstract representation of an the control structures.
- A visual simulator of fine granularity that allows to experience and to predict the risks of a virus in different topologies.

The above mentioned tools are based on design patterns and were represented in UML, in order to provide greater reusability and power experiments with new evolutionary forms of the computer viruses.

1 INTRODUCCIÓN

Resumen

Este capítulo incluye los objetivos del trabajo, tanto general como específicos. Además plantea la problemática que se está abordando y su justificación.

Objetivo del capítulo

- Plantear los objetivos de la tesis.
- Establecer la problemática que se aborda en el contenido y justificar su estudio.

1.1 Problemática

La creciente ola de virus informáticos y la rápida propagación que éstos tienen en la red, ha originado que los sistemas tradicionales para la detección se encuentren al límite de sus posibilidades para mantener a los sistemas libres de alguna infección.

Esto se debe a que el número de vacunas que existe por antivirus y por equipo sigue en aumento, ya que hay que actualizar de forma manual o automática las firmas de identificación así como las vacunas correspondientes. Este problema se ve acrecentado con la lentitud en que se crean las firmas de identificación y las vacunas para eliminar a los virus informáticos.

1.2 Justificación

En el trabajo, *"Un modelo del sistema inmune para prevenir y eliminar los virus informáticos"*, surge la necesidad de crear una firma que permita identificar y minimizar la curva de propagación que han generado los virus informáticos durante estas últimas décadas. Se presenta además un nuevo modelo de seguridad que permite coexistir con la evolución del software.

La principal dificultad para los creadores de antivirus es diagnosticar en qué momento surge un nuevo virus informático. Para ello han utilizado búsquedas heurísticas que permiten localizarlos basándose en pequeños patrones de comportamiento. Posteriormente, el virus es analizado para obtener un antídoto y así erradicarlo de los equipos de cómputo. Actualmente este proceso consume varios días antes de que las compañías de antivirus proporcionen una solución real al usuario, durante los cuales los virus se propagan rápidamente lo que causa importantes pérdidas económicas. Es por ello, que en esta tesis se propone un modelo basado en una firma llamada *señal de peligro,* que ayude a identificar una nueva infección y facilite la reconstrucción de los programas dañados, para así ayudar a minimizar la curva de propagación que estos presentan.

Para analistas en seguridad informática es necesario tener una herramienta que permita modelar y calcular los riesgos de una epidemia, así como la estrategia que se deberá implementar para salvaguardar la información de una forma eficaz. Esta estrategia se encuentra basada en el modelo del sistema inmune propuesto.

1.3 Objetivos

A continuación se describe el objetivo general y los objetivos particulares del presente trabajo.

Objetivo general

Presentar un modelo del sistema inmune que minimice la propagación de los virus informáticos. Una herramienta a nivel local llamada *"sistema inmune local"* que elimine los virus conocidos y futuros así como una herramienta visual que permita modelar y simular a dicho sistema en ambientes de propagación más complejos.

Objetivos específicos

Los objetivos particulares de esta tesis son los siguientes:

- Analizar y estudiar las diferentes formas evolutivas que han presentado los virus informáticos para diagnosticar futuras infecciones.
- Analizar las diferentes formas en que los virus contaminan a su huésped con el fin de diseñar una forma de eliminar a los virus de forma automática.
- Analizar y diseñar un antivirus con técnicas actuales, para encontrar los problemas básicos de los antivirus comerciales.
- Definir y establecer una analogía del sistema inmune de los vertebrados para encontrar una mejor barrera que permita prevenir y eliminar a los virus informáticos.
- Crear una firma llamada señal de peligro que identifique a los virus informáticos, permita reconstruir el archivo huésped y sea fácil de sintetizar.
- Realizar el análisis, diseño e implantación de una herramienta, que permita eliminar los virus conocidos y futuros dentro de un ámbito local.
- Realizar el análisis, diseño e implantación de una herramienta visual, que permita experimentar con la señal de peligro y las diferentes formas de propagación que tienen los virus informáticos.

1.4 Alcance de la tesis

En la presente tesis se muestran algunas evoluciones de los virus informáticos hasta finales del año 2002, dejando claro que no es posible conocer en su totalidad las nuevas formas evolutivas que pueden presentar los virus y que éstos están sujetos a los avances tecnológicos.

También se diseñaron y desarrollaron algunas técnicas utilizadas por los antivirus, las cuales únicamente operan bajo el sistema operativo Microsoft Windows y reconocen a un grupo de virus que infecta a los archivos con la extensión .EXE, .COM y el sector de arranque. Cabe recordar que muchas de las técnicas mencionadas en el Capítulo 3 no se encuentran documentadas debido a diversas patentes que manejan las empresas privadas de antivirus. Por lo tanto, este estudio solamente es para conocer los problemas esenciales de los antivirus.

En la analogía del sistema inmune de los vertebrados muchos de los aspectos principales fueron cubiertos como son:

- La firma llamada *señal de peligro* que se creó en el Capítulo 4, únicamente analiza programas en el lenguaje ensamblador del microprocesador Intel 8086/80386 y reconstruye los programas que han sido infectados con nuevos virus; siempre y cuando los virus utilicen las estructuras de contaminación estudiadas en el Capítulo 2.

- Funcionalidad del *Servidor T:* En él se presentan los aspectos básicos de su funcionalidad y está limitado a prevenir futuras infecciones. Éste solamente opera bajo un grupo de equipos de cómputo conectados entre sí, además de estar restringido a los aspectos que presenta la señal de peligro.
- La funcionalidad del *Servidor B*, solamente presenta los aspectos necesarios para construir una vacuna específica, limitando su funcionalidad a las restricciones que presenta la *señal de peligro*. Además de que su funcionalidad está dada por uno o más grupos de *Servidores T*.

En la analogía del sistema inmune de los vertebrados se analizaron las funciones básicas para mantener a los sistema de cómputo lejos de una infección. Aunque solamente se tomaron tres de los componentes básicos, este podría ampliarse con otras analogías basándose en otros tipos de células que ayuden a minimizar la propagación de los virus informáticos.

Se diseñó y desarrolló una herramienta llamada *"sistema inmune local"*que permite eliminar los virus en una máquina local. Esta herramienta se encuentra basada en el sistema operativo de Microsoft Windows y está limitada en la eliminación de virus por las estructuras analizadas en el Capítulo 2.

Por otra parte, se diseñó y desarrolló un simulador de granularidad fina en el lenguaje Java y con estudio de la señal de peligro, además de utilizar los modelos de propagación realizados por IBM [34].

1.5 Descripción del documento

Hablar de los virus informáticos, es tener que analizar los procesos involucrados para lograr su diagnóstico, así como una vacuna que permita erradicarlos de los sistemas de cómputo. En este trabajo, se presentan todas las partes que componen el problema de los virus informáticos desde el punto de vista evolutivo, en donde se analizan los modelos de seguridad y las técnicas utilizadas por los antivirus. También se analizan los diferentes modelos de seguridad basados en el sistema inmune, por lo que se propone un modelo que permita coexistir con la evolución del software.

1.5.1 Virus informático, antivirus y vacunas

Existen varias formas en que los virus informáticos han evolucionado, así como los programas de cómputo que existen hoy en día. Los virus contienen técnicas que les permiten ocultarse y cambiar la apariencia dentro de un archivo, con lo que impide que las herramientas de detección realicen un diagnóstico correcto y eficiente. Por otra parte, los diseñadores de los antivirus han creado clasificaciones de los virus informáticos, además de analizar la forma en que contaminan, que va desde que los virus se agregan a su huésped, hasta insertarse dentro de un programa en donde se modifica su conducta sintáctica.

Las políticas en seguridad informática en la actualidad son más específicas y, por lo mismo, se necesita incorporar detalladamente los lineamientos y posibles efectos alternos que ayuden a detectar a los virus informáticos. Estos factores han originado que los antivirus sean la

primera barrera contra los virus informáticos, pero hoy en día han sido rebasados por los avances tecnológicos.

Los antivirus han evolucionado lentamente. Esto se debe a que se encuentran en espera de las nuevas técnicas que implantan los virus informáticos, para después utilizar estas técnicas en busca de un diagnóstico que permita detectar a los virus en forma correcta.

1.5.2 Sistema Inmune

Un modelo del sistema inmune que elimine los virus informáticos, es la búsqueda de una mejor barrera que impida que los virus informáticos se propaguen rápidamente. Este modelo se basa en una analogía con el sistema inmune de los vertebrados, donde se analizan dos teorías. Una de ellas es la de realizar la diferencia entre lo propio y lo extraño, y la segunda es la teoría basada en la señal de peligro, para ello se presentan dos niveles de estudio:

- El nivel *micro,* que es donde se analizan las estructuras de control que tienen los programas para diagnosticar a un virus y eliminarlo de forma automática.
- El nivel *macro,* que es el estudio de la propagación de la señal de peligro y de los virus informáticos en forma distribuida.

1.5.3 Aplicaciones del sistema Inmune

La Epidemiología biológica proporciona la consolidación de los conceptos y los métodos que proporcionan un sustento a una verdadera disciplina científica. Los científicos han combinado las perspectivas micro y macroscópicas de enfermedades, lo que origina como resultado el estudio y analogía de enfermedades biológicas con los virus informáticos, que se encuentran separados en el mundo real, pero cerca al tratar aspectos científicos.

Se crea una herramienta llamada *"sistema inmune local"* que es la encargada de eliminar virus presentes y futuros bajo las reglas explicadas en el Capítulo 3. Esta forma de eliminar los virus siguiendo el modelo de propagación SIS (Susceptible Infecta a Susceptible)

La Epidemiología en los virus informáticos se ha utilizado para observar diferentes estadísticas, donde se analizan diversas topologías en busca de la tendencia, propagación e impacto que tienen los virus. Por esto, se propone utilizar a la Epidemiología para predecir el momento en que surge un nuevo virus informático; y así, de esta manera estimar la vacuna apropiada para que la propagación sea menor.

El propósito de diseñar un modelo del sistema inmune mediante un simulador, tiene la finalidad de comprender el comportamiento de la señal de peligro y el comportamiento de los virus informáticos dentro de un sistema de cómputo más complejo que el del modelo SIS. Además de evaluar nuevas estrategias para prevenir una propagación.

En el Capítulo 5, se muestran los aspectos del diseño de un simulador basado en UML y desarrollado en el lenguaje Java. Este simulador tiene las características de ser de granularidad fina y visual, lo que permite realizar experimentos con la señal de peligro y las

diferentes formas en que se propaga un virus; analizando a éstos en un equipo o en un modelo más complejo como son los sistemas distribuidos.

1 ESTADO DEL ARTE

Resumen

En este capítulo, se presentan los conceptos básicos necesarios para la comprensión de este trabajo, así como una descripción de los virus informáticos, poniendo énfasis en el contexto de las diferentes formas de contaminación que éstos presentan. Además de mostrar los esquemas más utilizados para realizar su detección.

Objetivo del capítulo

- Definir los conceptos básicos necesarios para la comprensión de los virus informáticos.
- Presentar un panorama general de la infección y propagación que tienen los virus informáticos.
- Mostrar los esquemas utilizados por los virus informáticos para propagarse y evolucionar.

1.1 Historia de los virus informáticos

John Louis Von Neumann, un matemático brillante que realizó importantes contribuciones a la física cuántica, la lógica y la teoría de la computación, publicó en 1949, en su artículo llamado *"Teoría y Organización de Autómatas Complicados (Theory and Organization of Complicated Autómata)"* la idea básica de un código que es capaz de reproducirse a sí mismo. Por esta razón se le conoce como uno de los primeros precursores en el campo de los virus informáticos [1].

- A finales de los años 50, en los laboratorios Bell, tres programadores, H. Douglas Mcllroy, Víctor Vysottsky y Robert Moris crearon un juego llamado "Core Wars" [2]. Éste consiste en la elaboración de programas para un simulador de computadoras, donde la idea principal era que los programas sobrevivieran con técnicas parecidas a las que utilizan los virus informáticos [3-5].

- John Shoch y Jon Hupp, investigadores de la empresa Palo Alto Research Center, que pertenece a Xerox, aseguran que en 1972 se elaboraron programas con ciertas técnicas que poseen los virus. Aunque estos programas podrían ser considerados "virus buenos", ya que controlanban continuamente la "salud" de las redes; a uno de ellos lo llamaron "el gusano vampiro" [6], debido a que se ocultaba en la red y sólo se activaba durante las noches para aprovechar que las computadoras no estaban en uso [7].

- En 1983 Ken Thompson recibía el premio Alan Turing de la Association of Computing Machinery. En su discurso basado en el juego "Core Wars" invita a experimentar con esas pequeñas "criaturas lógicas" [8].

- La adopción del nombre virus fue proporcionada en noviembre de 1983, en un seminario de seguridad en computadoras del Dr. Fred Cohen; quien realizó el experimento en una computadora VAX 11/750, en donde crea un programa que "puede modificar a otros, para incluir una copia quizá evolucionada de sí mismo"; con lo que establece un paralelismo entre los virus biológicos y los informáticos para justificar la adopción de dicha terminología [3,8]

- Dewdney publica un tercer artículo en el que se desliga de cualquier relación con los virus informáticos; posteriormente sigue involucrado en los juegos de la Guerra Nuclear y explica el funcionamiento de los virus. Se da el primer caso de contagio masivo a través del virus llamado "MacMag", también llamado virus "Peace" en computadoras Macintosh; este virus fue creado por Richard Brandow y Drew Davison y lo incluyeron en un disco de juegos que repartieron en una reunión de un club de usuarios. Uno de los asistentes, Marc Canter, consultor de Aldus Corporation, se llevó el disco a Chicago y contaminó el equipo de cómputo en el que se realizaban pruebas con el nuevo software [6].

- En el año 1987 se descubre la primera versión del virus "viernes 13", y el virus “Viena”; estos tienen mayor difusión en el mundo al momento de publicar una parte de sus códigos en un libro de virus, lo que provocó que muchos investigadores crearan variantes [9].

- En 1988 la informática realizó un trabajo que consistía en hacer más consciente a la industria; el objetivo principal era la necesidad de defender los sistemas informáticos de ataques realizados por los virus informáticos. Uno de los primeros antivirus llevaba el nombre de “Inyección para la gripe” (Full Shot); y fue creado por Ross Greenberg.

- En 1989, el virus llamado Dark Avanger se propaga por toda Europa y los Estados Unidos, haciéndose famoso por su estilo de programación a tal grado que se han escrito muchos artículos y hasta más de un libro acerca de este virus. Posteriormente inspiró a su propio país, para la producción masiva de sistemas generadores de virus automáticos [5].

- En junio de 1991, el Dr. Vesselin Bontchev, que trabajaba como director del Laboratorio de Virología de la Academia de Ciencias de Bulgaria, escribe un artículo, en el cual se reconoce a su país como el líder mundial en la producción de virus; por lo que dio a conocer la primera especie de virus búlgara, creada en 1988, que fue el resultado de una mutación del virus Viena. Estos virus fueron desensamblados y modificados por estudiantes de la Universidad de Sofía [10].

- El virus “Michelangelo”, no realiza una destrucción significativa pero la prensa lo vendió como una grave amenaza mundial en la primavera de 1992 [3].

- En los noventa, se producen enormes cambios en el mundo de la informática en donde se dio un aumento en el número de los virus en circulación, hasta límites insospechados. La razón principal de este desmesurado crecimiento se debe al auge de Internet, que en 1994 ya comenzaba a popularizarse en Estados Unidos, y un par de años más tarde empezaría a generalizarse en el resto del mundo.

- En 1995 se reportan, en diferentes partes del mundo, la presencia de una nueva familia de virus que solamente infectaban documentos sin ser archivos ejecutables directos del sistema operativo. Estos podían auto reproducirse infectando a otros documentos. Los llamados macro virus, sólo infectaban a los archivos de MS-Word [11], pero apareció una especie que atacaba al Ami Pro, ambos procesadores de textos. En 1997 se disemina a través de Internet en donde surge el primer macro virus que infecta hojas de cálculo de MS-Excel, denominado Laroux, y en 1998 surge otra especie de esta misma familia de virus que ataca a los archivos de bases de datos de MS-Access [12].

- El virus Strange Brew es el primer virus conocido en el lenguaje Java, este virus es capaz de copiarse en otros archivos siempre y cuando tenga los permisos de acceso, es decir, que sólo funcionará con ciertas condiciones como aplicación Java [13].

- A principios de 1999 se empezaron a propagar los virus adjuntos a mensajes de correo, ese mismo año fueron difundidos a través de Internet varios de estos virus.

1.2 Definición de un virus informático

El constructor universal de John Louis Von Neumann es una extensión del concepto lógico de las máquinas de cálculo universal. Neumann propone realizar un constructor para probar que su red celular colocada en una máquina de Turing es capaz de producir otra red celular. Esto únicamente sucede en el momento en que un programa se *"incrusta"* en otro, él llamó a esta última máquina *"Constructor Universal"*, y mostró que un programa que contenga dicha descripción es capaz de reproducirse a sí mismo.

Una definición más formal de un virus informático se encuentra en libro de Cohen [14], en la que menciona que *"cualquier secuencia de símbolos en la memoria de una máquina de Turing que tiene la capacidad de realizar un auto copiado en otro lugar de éste, se le llama virus."*

En otras palabras, un virus informático "*es un programa*" que puede "*infectar*" a otros programas modificándolos para incluirse como una copia de sí mismo, la copia puede desarrollarse o evolucionar [14].

El término *"virus"* fue dado por el Dr. Cohen en 1983; él establece una relación significativa entre los conceptos virus informáticos y virus biológicos. Es importante analizar estos dos conceptos de una manera más amplia, para comprender las similitudes y diferencias que existen entre ambos.

1.3 Analogía de los virus biológicos con los virus informáticos

La virología ha sido la ciencia microbiológica que ha surgido como resultado del hallazgo de enfermedades infecciosas, donde la implicación de microorganismos se demostraba con los medios habituales disponibles a finales del siglo XIX.

En 1898 Martinus Beijerink, realiza experimentos parecidos a los de Dimitri Iwanovski. Ambos en 1892 se enfrentaron a los conceptos de la época, y proponen que los agentes filtrables "*contagium vivum fluidu*" debían de incorporarse al protoplasma vivo del huésped para lograr su reproducción. Estos agentes infecciosos que cruzan los filtros de porcelana, fueron llamados *Virus Filtrables,* que después tomarían el nombre de "Virus". Hoy en día la definición aceptada es [15]:

> *"Los virus biológicos son fragmentos de ADN cubiertos de una capa de proteínas; se reproducen únicamente en el interior de las células vivas, tomando el control de su enzima y maquinaria metabólica. Sin esta maquinaria permanecen inertes como cualquier macromolécula".*

Esta definición describe a los virus biológicos, pero no ayuda a encontrar una semejanza más severa con los virus informáticos, por esta razón, se presentan las siguientes relaciones [16].

1) Los virus biológicos están compuestos por ácidos nucleicos. Estos ácidos poseen la información suficiente y necesaria para realizar cierta actividad, si esto se compara con los virus informáticos, éstos poseen un código que realiza un proceso similar.

2) Los virus biológicos *no tienen metabolismo propio;* por lo tanto, en el momento en que están fuera del huésped no manifiestan actividad de ningún tipo. Un comportamiento idéntico adquiere un virus informático en el momento en que no se encuentra en la máquina que lo interprete.

Aunque existen varias semejanzas entre los dos puntos analizados, también existen algunas diferencias que son importantes mencionar:

Los virus informáticos no tienen vida, capacidad que sí poseen los virus biológicos. El definir el término vida no es parte de ningún dogma, lo que hoy en día es cierto para la ciencia puede que el día de mañana no lo sea. Algunos científicos toman a los virus como partículas carentes de vida, y otro grupo lo ubica en una especie de "limbo" entre lo vivo y no vivo. La tercera opinión mantiene la postura que son formas vivientes y que al momento de evolucionar perdieron la característica de ser metabólicos; este proceso tiene el nombre de simplificación evolutiva, y surge en el momento en que los virus biológicos son definidos como seres vivos, lo que corresponde a una discusión filosófica.

1.3.1 Característica y similitudes

Después de analizar algunas similitudes entre los dos términos *"virus informáticos"* y *"virus biológicos"*, existen otras características que contribuyen a esta comparativa.

La primera característica es el tamaño que presentan los virus informáticos con relación a los programas que infectan, ya que estos deben ocultarse ante los usuarios, y estar latentes durante algún tiempo. Los virus biológicos presentan la misma característica, son demasiado pequeños comparados con sus huéspedes.

Tipo de acción: Los virus informáticos y los virus biológicos modifican y/o dañan a sus huéspedes. Aunado a esta característica, existe un proceso de acoplamiento con el organismo sano por así llamarlo, ya que en el momento de realizarse éste, existe una distinción de sí mismo para no autodestruirse.

Modo de acción: Ambos tipos de virus inician su actividad de forma oculta, sin conocimiento, ni consentimiento del sistema, después de un evento significativo el virus puede activar el mecanismo y hacerse visible.

Codificación de la información: Ambos tipos de virus por definición contienen la información necesaria para su auto reproducción. Los virus informáticos lo realizan por medio de cláusula transitiva [17] y los virus biológicos poseen un código cuaternario (cuatro nucleótidos combinados en tripletes).

Propagación: Los dos tipos de virus utilizan medios para su propagación; los virus biológicos se propagan a través del aire, el agua o por el contacto directo entre un huésped sano y uno infectado. Los virus informáticos se propagan por todo aquel medio que implique la transitividad de información.

Latencia: Ambos virus tienen el periodo de *latencia,* que es el momento en que la infección se realiza pero no se presentan los síntomas.

Mutar: La *susceptibilidad* de mutar algo, es la que posibilita el proceso evolutivo de las formas vivientes para sobrevivir, y se encuentra en los virus biológicos bajo determinadas condiciones; esto se debe por influencias de distintos factores o por manipulación genética. Los virus informáticos, pueden cambiar para dar origen a nuevas variedades de virus, algunos de estos cambian por sí solos, esto se realiza al cifrar el programa y otros por medio de programación directa.

Proceso reproductivo: Ambos virus carecen por sí mismos de la capacidad de reproducción y para realizarlo es necesario tener conocimiento de la estructura del huésped.

Sinergia: El término alude a un mecanismo de exaltación o potenciación del poder patógeno que ejerce un virus en otro diferente en el momento en que están juntos, dicho de otra forma el poder de ambos, es mayor que la suma de los poderes individuales. El fenómeno de sinergia es compartido por ambos tipos de virus.

Tipos de infecciones: Los virus biológicos pueden actuar por medio de dos tipos de infecciones:

> **Las líticas:** Fijación de los virus en la superficie de la célula huésped.
>
> **Las liso génicas:** El virus integra su ADN con el ADN del huésped y en el momento en que éste duplica su ADN también se duplica el ADN del virus. El huésped aparentemente no se ve afectado por tal proceso; de cualquier manera, la infección liso génica por influencia y por determinados factores puede transformarse en una infección lítica.

Los virus informáticos, poseen estos dos tipos de infecciones, aunque se explica con mayor detalle en el punto en el Capítulo 4.

Epidemiología: Ambas entidades cumplen con las mismas bases epidemiológicas y por lo tanto son válidos los términos utilizados en Epidemiología biológica.

1.4 Contaminación por medio de un virus informático

Una forma general de estudiar a los virus informáticos, es por medio de su código que muestra como se realiza la contaminación, y donde el usuario deberá analizar el programa para determinar si se encuentra infectado. Como ejemplo se utilizará un programa que permite mostrar el funcionamiento básico de dicho problema. Para ello definimos algunos símbolos como son:

```
= Símbolo que se utiliza para representar una definición.
: Símbolo que se utiliza para representar una etiqueta.
; Símbolo que se utiliza para separar sentencias.
:= Es usado para la asignación.
== Es usado para la comparación.
{ } Agrupan una secuencia de sentencias.
 . . . Es utilizado para indicar una parte irrelevante.
```

```
Programa virus = {
1234567;
     subrutina infectar ejecutables ={
          loop:
               Archivo = obtener un archivo;
               Sí (en la primera línea del Archivo == 1234567)
               entonces
                    loop;
                    Se inserta al principio el virus en el
                    archivo;
                    Copiar las instrucciones de daño y la pila
                    de disparo;
                    Copia la rutina para infectar a los demás
                    ejecutables;
                    Modificar el proceso main para insertarse y
                    ser ejecutado en primer plano;
     }

     subrutina proceso de daño = {
               inicia el proceso de realizar algún daño al
               sistema;
     }

     subrutina pila de disparo =  {
               detecta algún evento para activar el proceso de
               daño;
     }

     main =    {
               infectar a los archivos ejecutables;
               Sí(pila de disparo) entonces proceso de daño;
     }
}
```

En el ejemplo anterior, se observa que el virus realiza una búsqueda de los archivos del sistema, en donde verifica la existencia de una firma, de no existir ésta, el virus se inserta para contaminar al archivo huésped. El mecanismo de daño del virus, únicamente se disparará si existen las condiciones necesarias dentro del sistema, por lo que la propiedad principal del virus es de tener la “habilidad” de infectar a otros programas [14].

1.4.1 Modelo de propagación

En los modelos de propagación; algunos de los más rápidos son: el de los virus *LoveLetter*, *SirCim* y *Code Red*.

Según los datos recolectados por la Asociación Internacional de Seguridad Informática ICSA [18], *LoveLetter* es el virus que se ha propagado con mayor rapidez. En 1995 un virus tomaba aproximadamente un mes en propagarse; nada comparado con el virus informático llamado

LoveLetter, que en sólo cuatro horas se propagó por la red para contaminar a varias computadoras.

Este incidente, se discute desde el punto de vista epidemiológico, en donde se analiza la forma de propagación en una población determinada. Para realizar este estudio, se utilizan modelos más simples, basándose en una propagación homogénea, que no es más que una computadora, pueda infectar a cualquier otra computadora.

La suposición anterior no es verdadera para los virus tradicionales por así llamarlos, donde la propagación sigue un modelo gráfico de tipo jerárquico, en el cual la infección tiende a la propagación de las computadoras más cercanas. Sin embargo, no es el caso del LoveLetter, que infecta a las computadoras más lejanas, donde la correspondencia electrónica tiende a ser de una forma global y la propagación puede considerarse homogénea.

El ciclo vital de un "gusano" se divide en dos etapas. La primera etapa es desconocida para los analizadores de los virus informáticos, y es precisamente en este momento en que los virus sé propagan; la segunda etapa ocurre después, en el momento en que la firma del virus ha sido encontrado por alguna empresa.

En la primera etapa, la Ecuación 2-1, describe el número de sistemas infectados[19]:

$$N(t) = e^{(g-a)\frac{t}{\tau}} \qquad (2\text{-}1)$$

donde:

t = Tiempo transcurrido,

g = Promedio de infecciones,

a = Mortalidad y

τ = El tiempo en que se realiza la propagación.

En la Ecuación 2-1 no es exacta, ya que después que una empresa encuentra la firma del virus, el ciclo no es el mismo, esto es conocido como t_0, ya que los usuarios actualizan sus antivirus.

En el momento en que las actualizaciones de los antivirus se encuentran distribuidas, se puede pensar que están instalados en un porcentaje de las computadoras infectadas, y que la mortalidad resulta ser la probabilidad de ***1-p***. También existen algunos usuarios que son administradores y que han aprendido de experiencias anteriores por lo que actualizan puntualmente sus antivirus.

Por lo tanto la propagación inicia en una segunda etapa con una fracción de computadoras infectadas ***(1-p).***

(2-2)

$$N(t) = (1-p)\cdot e^{(g-a)\cdot\frac{t0}{\tau}} \cdot e^{(1-p)(g-a)\cdot\frac{t-t0}{\tau}}$$

La primera pregunta que surge es: ¿Cómo se puede reducir la propagación de los virus que tienen una forma exponencial creciente?

Para lograr una disminución en el exponente de la ecuación (2-1), se realiza mediante el factor de multiplicación del virus (*g-a*) que es mayor de 1, por lo tanto, el virus estará en aumento y, si es menor a 1 tenderá a la "muerte".

Una solución trivial para disminuir la población del virus consiste en aumentar el factor ***p;*** es decir adquirir más exploradores de virus e instalarlos en otras computadoras. Si se analiza la ecuación (2-2) se confirma la disminución del exponente, además se asegura que el virus morirá más rápidamente en la segunda etapa. El único problema es conocer el momento de adquirir más antivirus que disminuya el índice de infección en la primera etapa de la vida del virus.

1.5 Los antivirus y las vacunas

Comúnmente se confunden los términos *antivirus*[1] y *vacuna*[2] debido a que estos se encuentran fuertemente unidos, es difícil distinguir donde empieza uno y termina el otro

Se puede mencionar que el inicio de los antivirus surgió con John Shoch y Jon Hupp, investigadores de Palo Alto Research Center que habían elaborado programas con técnicas parecidas a las de un virus, aunque estos programas podrían ser considerados virus buenos, ya que controlaban continuamente la salud de las redes, sin embargo, éstos consumían demasiados recursos [7].

No se debe olvidar que la mayoría de los antivirus en el mercado surgieron por necesidad, debido a que la propagación de los virus informáticos se encontraban en expansión, por lo que se trató de encontrar un *"remedio"* por así llamarlo. Se observó que estas soluciones no resultaban factibles debido a que se encontraban un paso atrás en el desarrollo de los virus informáticos. Esto es, primero surge un virus y luego se debe encontrar el antídoto para que se actualicen a los antivirus que serán los encargados de realizar la vacunación.

Por otra parte las vacunas automáticas son una meta por cumplirse, ya que en el momento en que un virus informático reciente es encontrado, los expertos llevan un tiempo en buscar una firma o cadena que sea única para su detección. Posteriormente, esta firma es enviada a todos los antivirus para que detenga la propagación. Por otro lado, un grupo de expertos en virus

[1] Es un programa que se encarga de la detección y eliminación de los virus informáticos
[2] Es el antídoto o programa encargado de erradicar al virus del lugar donde se hospeda.

informáticos, realiza las vacunas correspondientes, por lo que se agrega más tiempo al proceso mientras la propagación del virus continua.

1.6 Algunas técnicas perfectas para la defensa de los virus informáticos

Para Cohen, existen tres maneras en que se obtiene una perfecta prevención contra la propagación de los virus informáticos, ya sean estos dentro del mismo equipo o en un ambiente distribuido [20-22] y estos son:

1. Compartir con limitaciones
2. Limitar la transitividad
3. Limitar el funcionamiento

1.6.1 Compartir con limitaciones

Esta técnica se basa en que un usuario no debe leer de un nivel más alto de seguridad, y no intentará escribir en una área de menor clasificación, esto tiene la finalidad de que no exista una fuga o infiltración en la información, como se observa en la Figura 2-1

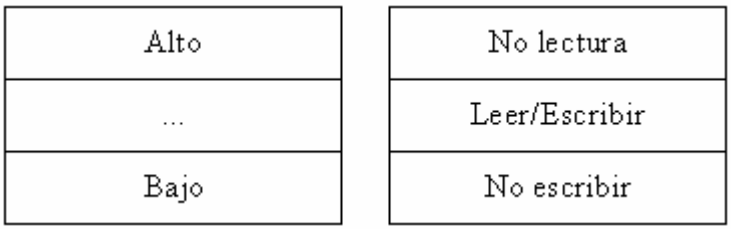


Figura 1-1 Modelo de seguridad de Bell LaPadula

De la misma manera, opera el modelo de integridad de Biba, donde no se lee la información de un bajo nivel de integridad, para no corromper la información y no se escribe en un nivel alto de integridad con el mismo fin.

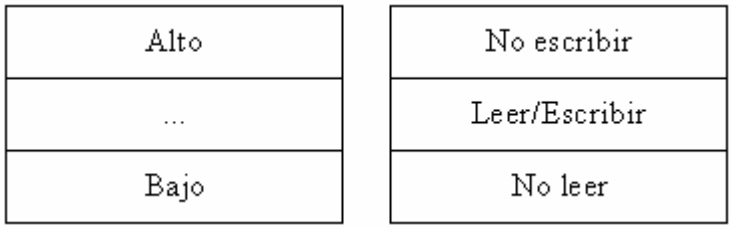


Figura 1-2 Modelo de integridad Biba

Ahora sí se requiere que el sistema mantenga la integridad [23] y la seguridad, se deben unir los dos modelos, lo que dará como resultado un sistema en el que no se permita Leer ni Escribir fuera del nivel de uso, como se observa en la Figura 2-3.

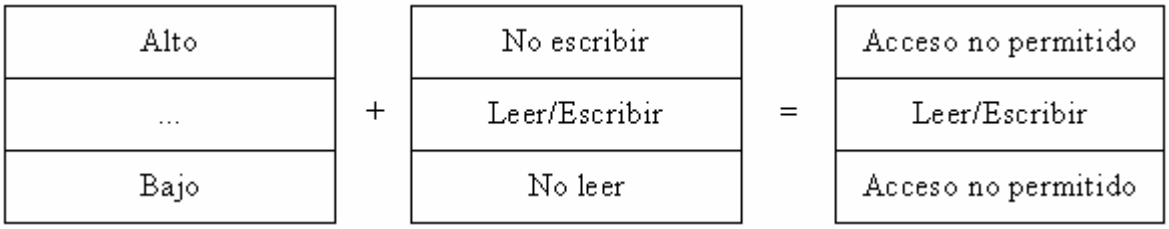


Figura 1-3 La combinación de seguridad más integridad

Esto es únicamente un caso específico, ya que al momento de compartir con limitaciones, se deberá implementar una política basada en una estructura llamada poset[3], que es un conjunto parcialmente ordenado.

$\forall$ dominio A, B y C $\in$ poset
$A \leq B$ y $B \leq C \rightarrow A \leq C$
$A \leq B$ y $B \leq A \rightarrow A = B$
$A \leq A$

En un poset, la regla es que la información fluya hacia arriba, como se observa en la siguiente Figura 2-4:

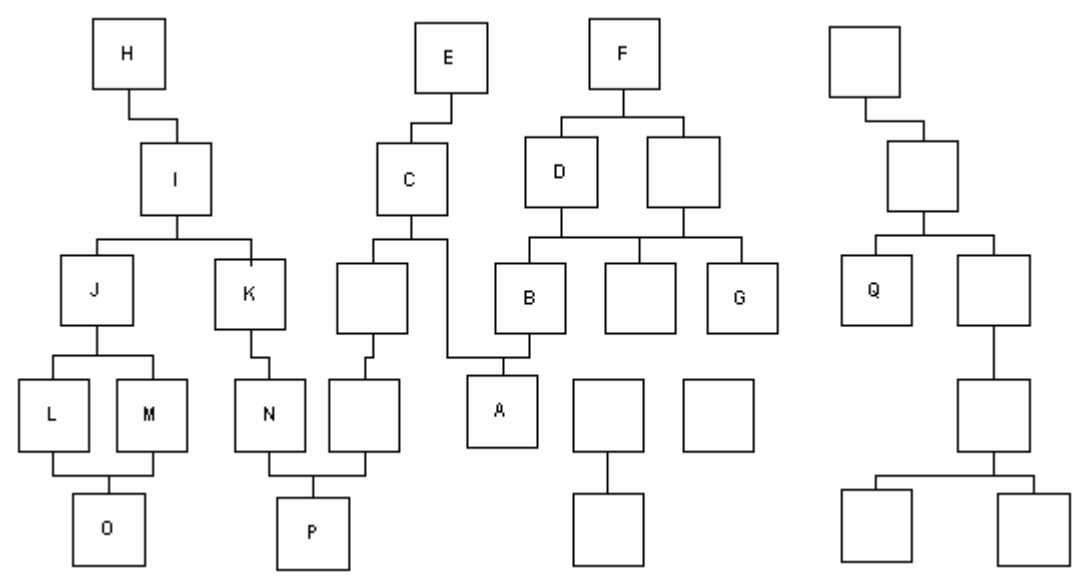


Figura 1-4 Un poset

[3] Es un conjunto de cuadros conectados por medio de líneas.

Se escribe un virus informático, en el cuadro marcado con la letra A. Como se observa en la Figura 2-4, el virus se propagará hacia los cuadros B, C, D, E y F, pero no en G, esto se debe a que no existe, una ruta que lleva al virus de A a G, la ventaja es que la propagación se presenta en un conjunto limitado y no en todo el sistema.

1.6.2 Limitar la transitividad

La segunda posibilidad es limitar la transitividad como se observa en la Figura 2-5.

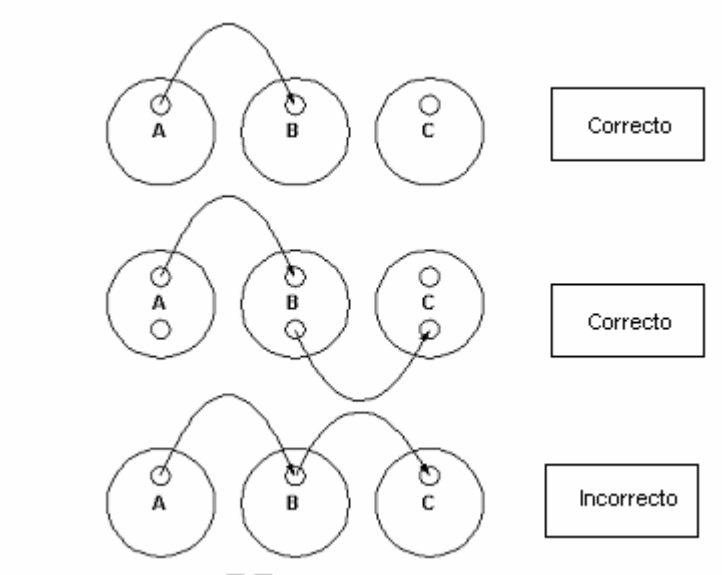


Figura 1-5 Limitar la transitividad

Se podría realizar una implantación, con limitar la transitividad para prevenir la propagación de los virus informáticos a partir de una cierta distancia del origen. El problema es que no es posible realizarlo en el mundo real [17].

1.6.3 Limitar la funcionalidad

La tercera posibilidad se basa en la prevención de los virus informáticos dentro de un sistema local o en un área distribuida. Esto se logra evitando instrucciones que puedan dar lugar a la creación de virus.

1.7 Detectar todos los posibles virus

Las tres técnicas descritas anteriormente permiten prevenir la propagación de un virus informático. Otro caso de estudio es, en el momento en que un equipo se encuentra contaminado por un virus informático y lo que se desea es eliminarlo. Cabe aclarar que todas las técnicas que se explican a continuación son imperfectas.

1.7.1 Características de los programas que detectan todos los virus informáticos

Una pregunta que surge es, ¿Se pueden detectar todos los virus? Únicamente sí existen las condiciones siguientes:

1. Funcionamiento de los programas por siempre sin cambio alguno o
2. Tener un número infinito de falsos positivos[4] o
3. Tener un número infinito de falsos negativos[5] o
4. Tener la combinación de los tres puntos anteriores

1.8 Las técnicas para la detección de los virus informáticos

El hablar de técnicas es hablar de procedimientos comprobados para la localización de los virus informáticos, desde sus inicios, el primer intento por mantener al margen este tipo de programas era el de crear un programa similar al de un virus informático. El investigador Bob Thomas liberó un programa llamado *"Rastrero"*; mientras que otro programador escribió un virus llamado *"Segador"*, el cual se reproducía en la red para matar *"Rastreros"*. Obviamente esto no sería una solución factible por ciertas razones que saltan a la vista, como es el costo de cómputo desperdiciado para mantener vivos a los *"Segadores"*; aunque en un principio su costo era sostenible, sería la solución más apropiada, hasta crear una especie de plataforma que permitiera controlar los costos que estos generaban. El primer problema fue resuelto, la vacuna sola debería de ser activada en el momento en que la presencia del virus fuera encontrada; esta plataforma hoy es conocida como antivirus.

La búsqueda de las firmas dentro de un virus es más que una técnica, es la parte esencial de los antivirus que ha sido estudiada, y es la encargada de identificar de forma correcta la parte del código de un virus, así como evitar los llamados falsos positivos y falsos negativos. Es por ello que se debe realizar un estudio sobre los diferentes métodos de búsquedas que han empleado los antivirus.

¿Qué es una firma? Una secuencia de bytes que identifica a un solo virus. Una buena firma es aquella que se encuentra dentro de cada objeto infectado por un virus; pero es menos probable de ser encontrado si el virus no está presente. Es decir que la probabilidad de falsos negativos y falsos positivos debe ser reducida al mínimo; normalmente un experto humano elige una firma para el nuevo virus al detectar una zona segura. Además deberá ser una rutina menos común y esto se logra con la extracción de la zona de auto verificación que tienen los virus informáticos.

Para los Macro Virus, el problema de la detección es aún más sencillo porque se obtiene una suma de validación del código y se compara con una base de datos. Ya que en éstos por lo regular no presenta el fenómeno de cambiar sus instrucciones al momento de acoplarse.

[4] Falso positivo: es cuando un antivirus anuncia la presencia de un virus que no lo es.
[5] Falso negativo: es cuando un antivirus anuncia que no existe la presencia de un virus y en realidad existe.

El experto en virus informáticos puede llevarse muchos días en encontrar una buena firma, una intención es el de crear la herramienta que pueda identificar a un programa en forma única. IBM ha desarrollado un método estadístico para extraer automáticamente firmas de un virus informático [24]. La idea básica es recopilar una gran cantidad de programas, y después utilizar esta información para estimar las probabilidades de falsos positivos en las firmas creadas. En la práctica, las firmas extraídas con este método son menos probables de generar falsos positivos.

1.8.1 Búsqueda basada en una firma simple.

Esta técnica consiste en localizar una cadena de 16 o 32 bytes dentro del archivo que tiene virus, el cual tiene una probabilidad alta de ser una identificación única para el virus. Para localizar dicha cadena, se utilizan algunos algoritmos que permitan realizar la localización en un tiempo menor y de una manera más compleja.

El primer intento se realizó con una búsqueda basada en la fuerza bruta:

Características:

- No posee ninguna fase del proceso previo;
- Su análisis será comparar uno a uno con un desplazamiento hacia la derecha.
-

Debido a la creciente ola de virus informáticos que se dio a finales de los años 80's, la cantidad de éstos aumentaron considerablemente. Esto motivó a que se mejoraran las técnicas de búsqueda, ya que las firmas de identificación eran más complicadas de extraer. Este cambio en los antivirus era provocado por algunos virus que tenían un parecido mayor.

Para solucionar el problema anterior, se toma la idea de un algoritmo que permita buscar una cadena en donde se hace uso de algunos caracteres como comodines, con la idea de ser más flexible, el algoritmo se conoce como búsqueda con un autómata.

Para realizar la búsqueda de una cadena que pudiera contener un patrón más complejo de virus informáticos, una herramienta llamada TBSCAN[6], empleó una manera de extender los caracteres comodines en la que utiliza la siguiente definición:

?	= Representa cualquier símbolo de entrada,
%n	= Saltar de 0 a N símbolos de entrada,
*n	= Saltar exactamente N símbolos y
**	= Saltar un numero arbitrario (incluye la posición 0).

Con estos comodines mencionados, se obtiene una firma más compleja, pero la forma de implementarla dentro de un sistema tiene ciertas características que deberán ser consideradas y que se explican en el Capítulo 3. Otra forma de mejorar la eficiencia en la búsqueda de

[6] Parte integral de la herramienta ThunderByte Antivirus.

patrones es el de no tener que realizar la búsqueda en todo el archivo ni en todos, por lo que se proponen algunos puntos que deberán tomarse en cuenta.

1.8.2 Algunos puntos para una mejor eficiencia en la implantación de un buscador.

- Las firmas se deben clasificar por el tipo de archivo en el que se encuentran.
- Las firmas se clasifican según la posición de inicio del código, esta es otra de las maneras de minimizar el árbol de búsquedas. Si un virus se incrusta al final del archivo tendrá que modificar las direcciones que indiquen el inicio del programa para realizar la unión de los segmentos. Esto reduce el tamaño de búsqueda, dado que por lo regular los virus son más pequeños con respecto a su huésped.
- Debe desarrollarse un algoritmo que sea lo más eficiente al momento de localizar el patrón deseado.
- Se debe realizar un diseño para implantar los diferentes algoritmos de búsqueda, debido a que cada empresa, desarrolla sus propios métodos, y se debe pensar en sustituir este módulo de una forma que sea lo más transparente posible para el desarrollador.

1.9 Epidemiología biológica

La primera definición de la Epidemiología basada en el objeto de estudio fue la propuesta por D. Kleinbaum y colaboradores en su libro de texto publicado en 1982, para quienes la Epidemiología es sencillamente el estudio de la salud y la enfermedad en las poblaciones humanas [25]. Enseguida los mismos autores aclaran que la definición no es redundante, ya que la salud debe entenderse como “estados de bienestar” y la enfermedad como “procesos patológicos”.

Con lo explicado anteriormente se tienen los dos siguientes puntos. El primero es al definir a la Epidemiología con base en sus métodos o en sus objetivos, queda claro que se trata de una legítima disciplina de la ciencia; el segundo punto es que más allá de la simple enumeración de hechos, la Epidemiología tiene el propósito de explicar los fenómenos que constituyen un objeto de estudio.

La Epidemiología podría perderse en el extremo opuesto, la mala e insuficiente especificación de los conceptos y los métodos, que es la falta de coherencia de los hallazgos epidemiológicos. Esto último fue claro por el doctor Petr Skrabanek, en su trabajo “La miseria de la Epidemiología” publicada en 1992, que censura a los estudios epidemiológicos por la falta de consistencia de sus resultados y por incurrir constantemente en el problema epistemológico de confundir asociación con causalidad [26]. Por una parte, entender y aplicar correctamente los conceptos y técnicas y por la otra, evitar la utilización desmedida y la interpretación de la estadística, como escribió el mismo Skrabanek [26], pensando en los que se dedican al estudio de la Epidemiología.

Dentro de la virología informática se encuentra Frederick Cohen, que definió a los virus informáticos [17] y junto con Murray [27] lograron precisar la conexión entre la propagación de los virus informáticos y la Epidemiología básica. Aunque existía mucha similitud entre la Epidemiología Biológica y la Epidemiología Informática, en donde no se crearon las bases para analizar las diferentes interpretaciones que éstos podían tener entre sí, lo que deja un hueco importante para el estudio de esté fenómeno. Para Gleissner [28], que examinó un modelo de virus informático separado de un sistema *"real"*, presentó el mismo resultado que había mostrado Cohen, que los virus tenían una propagación exponencial. Aunque, los resultados mostrados por los investigadores antes mencionados, no mencionan la capacidad que tienen los usuarios para eliminar a los virus informáticos, esto es un factor importante al momento de atacar esté problema. Solomon [29] presenta un estudio de este modelo en forma determinista [30] con el fin de analizar la propagación de virus informáticos basada en la Epidemiología Matemática.

La primera aplicación de modelar matemáticamente a la propagación de la enfermedad infecciosa fue realizada por Daniel Bernoulli en 1760 [31], él muestra la identificación del agente responsable de la transmisión de la viruela por un siglo, en donde formula una ecuación diferencial que describe la dinámica que tiene la propagación [32].

En las últimas décadas los virus informáticos han provocado que el funcionamiento de los equipos de cómputo tenga cambios importantes en la seguridad, debido a conductas provocadas por los brotes masivos y algunas otras formas de contagio extrañas, lo que los hace cada vez más parecidos a las enfermedades biológicas.

En el área de los virus informáticos, surgen varias preguntas del futuro que éstos producirán en el mundo de la computación y de la informática.

1. ¿Qué tan malo es este problema hoy en día?
2. ¿Qué tan malo podría ser este problema?
3. ¿Cómo se predicen futuros desastres?

Para darle un razonamiento correcto a las interrogantes antes mencionadas, se deben analizar los dos enfoques que tienen los virus informáticos. El primer análisis se establece en el nivel micro, que es el estudio que han realizado las empresas de seguridad como ICSA[7] [18] y el segundo, se proporciona en el nivel macro, el cual ha tenido poca importancia y donde la mayoría de modelos matemáticos explica la propagación de los virus informáticos en forma específica. Estos acercamientos epidemiológicos, han originado que las empresas, tomen las medidas necesarias para asegurar la integridad de sus sistemas.

Las estadísticas de ICSA, muestran cuáles son las tendencias actuales de los virus informáticos, aunque en ningún momento, indican cómo se llega a realizar la contaminación. Por otro lado, un grupo de investigadores de IBM, publicó un artículo en el que se analizan

[7] International Computer Security Asociation.

los virus informáticos a un nivel macroscópico, y de esta manera determina cómo se realiza la propagación y el impacto que éstos tienen en las diferentes topologías[8] epidemiológicas [33].

Otro trabajo importante, es el de Jeffrey O. Kephart y Steve R. Withe también de IBM [34], en donde presentan información de los modelos epidemiológicos. En este trabajo explican el modelo heterogéneo de la comunicación entre sistemas informáticos individuales.

Mostrando los siguientes modelos:

- Un Modelo SIS (Susceptible Infecta a Susceptible) en un grafo al azar.
- Un Modelo Jerárquico.
- Un Modelo Espacial.

[8] Un mundo es compuesto de células que pueden ser habitadas por *"criaturas"* que a su vez pueden reproducirse en otras células vecinas en condiciones específicas. La topología de un mundo es una descripción de la vecindad entre las células.

1 VIRUS Y ANTIVIRUS

Resumen

En este capítulo se crea una clasificación de los virus informáticos por la forma en que se acoplan a su huésped, así como la descripción de las herramientas más importantes que se han utilizado para detectar y eliminar a los virus informáticos. Asimismo se explican algunos problemas abiertos para los antivirus.

Objetivo del capítulo

- Crear una clasificación de los virus informáticos para crear una firma que puede identificarlos de una manera automática.
- Describir las técnicas utilizadas por los antivirus que permitan detectar a los virus informáticos.
- Crear algunas técnicas que permiten eliminar a los virus informáticos.
- Mostrar algunos problemas abiertos para los antivirus.

1.1 Clasificación de los virus informáticos

Las computadoras se diseñaron para ejecutar instrucciones útiles para alguien en particular, sin embargo, las instrucciones ejecutadas pueden ser perjudiciales o destructivas por naturaleza, estas codificaciones malévolas han recibido nombres diferentes *Malware* y *Vandalware* [35]. En años recientes, las ocurrencias del *Malware* han sido descritas casi uniformemente por los virus informáticos; aunque los virus son extensos, pero no son responsables de muchos de los problemas atribuidos a ellos [5].

En los orígenes de los virus informáticos, éstos se propagaban dentro de los sistemas operativos más difundidos, Microsoft y Linux.

1.1.1 Los virus que infectan a los archivos del sistema operativo Microsoft

Se ha realizado un estudio de las diferentes formas en que el Sistema Operativo de Microsoft llamado Windows y su antecesor MS DOS han sufrido el ataque de los virus; el estudio tuvo su base principalmente en explicar la forma en que se acoplan los virus en los programas existentes, este estudio se realizó con más de 10,000 virus principales.

Los virus que infectan a los archivos con la extensión .BAT, tienen una secuencia de instrucciones del Sistema Operativo Microsoft, en el cual se describe un pequeño lenguaje que permite manejar algunas propiedades del entorno, como se observa en el Código 3-1, esta es la manera de escribir un virus con estas instrucciones.

```
@echo virus de prueba
pause >nul
cls
ctty nul
for %%f in(*.bat) do copy %% + virus.bat
ctty con
cls
```

Código 1-1 Nuevo virus BAT

En el año de 1986 el virus llamado *Brain* se propagó por América; este virus fue creado por los hermanos Basit y Alvi Amja. El virus originalmente infecta los discos de 5,25 pulgadas, lo que da origen a la tecnología de antivirus (Fighting Computer Viruses). Estos virus se alojaban en la parte del sector de arranque de los discos.

El funcionamiento de este virus se aprecia más claramente en la Figura 3-1.

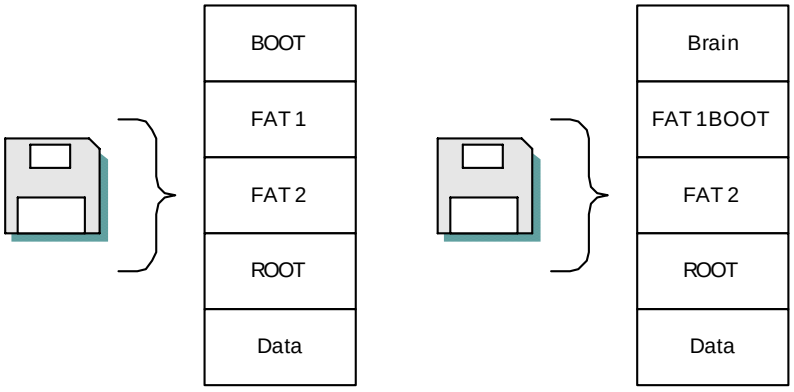


Figura 1-1 Contaminación del sector inicial

En la Figura 3-1, se observa que el sector inicial del disco contaminado, es desplazado a otro lugar en el mismo disco, y el virus es colocado en el sector inicial. De esta manera, en el momento en que el equipo de cómputo se inicia por medio de este disco, el sector inicial se ejecuta y toma el control de accesos de todos los discos (que en el microprocesador Intel es la interrupción 13h), posteriormente el disco que se introduzca en el mismo equipo llegará a ser contaminado.

Para mostrar cómo un virus se acopla a un programa, se tienen los diferentes esquemas en que se realiza este proceso, para ello se creará un ejemplo que permite mostrar cada uno de éstos. Inicialmente se tiene un archivo de una longitud determinada.

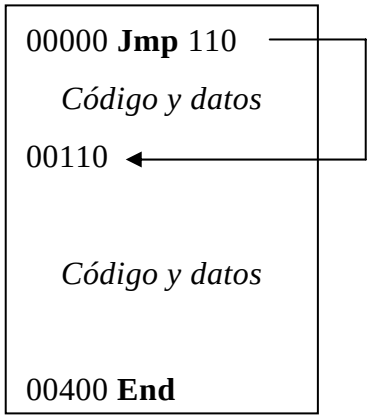


Figura 1-2 Archivo original y limpio

En la Figura 3-2 se observa un programa con una longitud de 400 bytes, y una instrucción (Jmp) que indica un salto hacia la etiqueta marcada con 00110, que en este caso es el lugar donde inicia el programa y termina en el momento en que la etiqueta 00400 es alcanzada.

El primer caso de estudio es la contaminación al inicio de un programa. Este posee dos formas básicas, colocar el virus al principio del archivo o al final del mismo, para ello se toma el ejemplo de la Figura 3-2 donde se muestra cómo se realiza esta primera forma de contaminación.

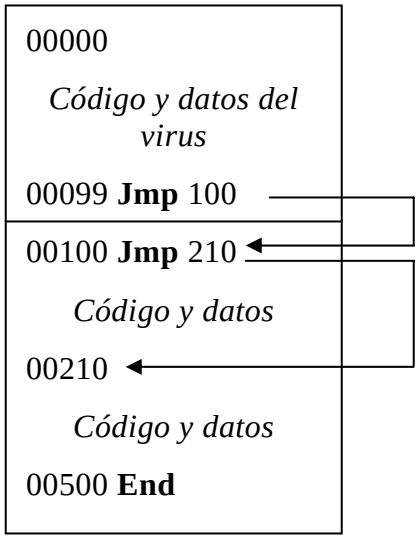


Figura 1-3 Contaminación al principio del archivo

En la Figura 3-3 se observa cómo el virus desplaza el código original a una posición igual o mayor a la del virus, con el objetivo de insertarse al principio. Esta forma de contaminación es usual en archivo con extensión *.COM*. La eliminación de este virus es rápida, ya que únicamente hay que identificar el segmento del código del virus y corregir el desplazamiento provocado por la contaminación.

El siguiente esquema de contaminación se da en el momento en que el virus es colocado al final del código, para modificar las primeras instrucciones con el fin de que sea el código del virus el que se ejecute primero; este virus informático se agrega a los archivos con extensión EXE[1] (*MZ* o *NE*) de Microsoft, la eliminación de éste virus requiere el conocimiento en la estructura del archivo ejecutable.

[1] Microsoft idea esta estructura para ejecutar programas mayores de 64Kbytes.

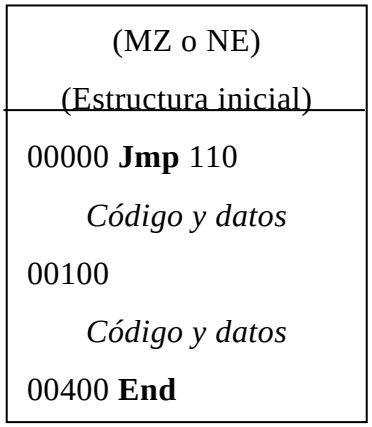


Figura 1-4 Archivo Microsoft del tipo EXE libre de virus

En la Figura 3-4 se muestra un archivo con una cabecera .EXE. Para que el virus se pueda insertar dentro de un archivo EXE, se tiene que modificar la estructura inicial y colocar al *IP* (iniciador de instrucción) con la dirección del virus como se ve en la Figura 3-5.

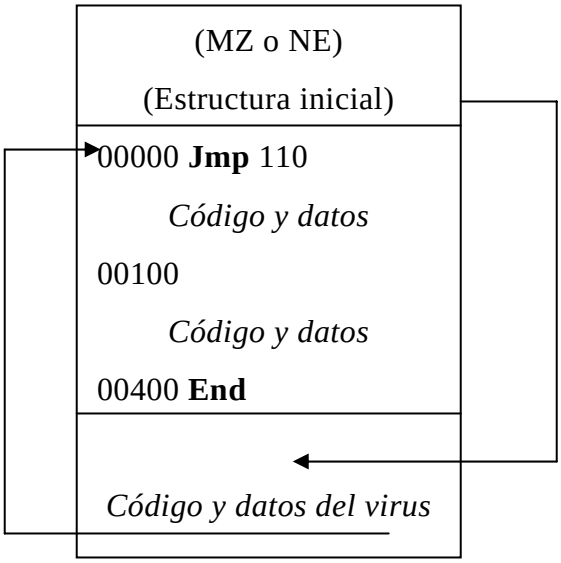


Figura 1-5 Un archivo Microsoft del tipo EXE contaminado por un virus

En la Figura 3-5 se observa que el virus modifica la estructura inicial del archivo huésped para que el virus se ejecute primero.

Otra forma en que se realiza una contaminación es de la siguiente manera.

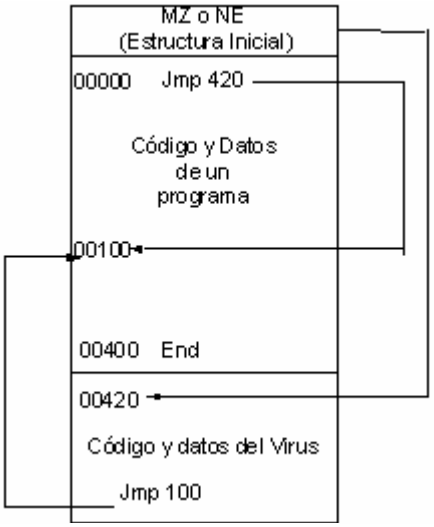


Figura 1-6 Un archivo Microsoft del tipo NE contaminado por un virus

En la Figura 3-6 se observa que la antigua instrucción llamada *"Jmp 100"* ha sido modificada para que el primer salto del programa se realice hacia el código del virus, y después este transfiera el control al programa original.

Una forma de eliminar el virus, es reemplazar la instrucción *"Jmp 420"* por el *"Jmp 100"* y eliminar el código agregado, la instrucción Jmp original (*Jmp 100*) se encuentra dentro del virus, ya que es el encargado de unir los dos nodos o segmentos, el del virus y el del programa.

1.1.2 Los virus que infectan a los archivos del sistema operativo Linux

El uso del Sistema Operativo Linux ha ido incrementándose estos últimos años, esto ha provocado que se instale con mayor frecuencia en los hogares, por lo que se da un aumento a la probabilidad de existencia de virus para esta plataforma. Dentro de los fabricantes más conocidos y los más populares son *"RedHat"* y *"Suse"*.

Se puede pensar que Linux es un sistema operativo seguro, aunque el término seguridad resulta ser utópico. Cohen dice que si existe la posibilidad de compartir información también puede existir la posibilidad de que existan los virus informáticos [14].

Las técnicas y estructuras que se presentaron en el punto 3.1.1, también pueden ser aplicadas dentro del sistema operativo Linux, y de igual manera, los virus informáticos modifican la estructura inicial del huésped para incluirse.

Los archivos de Linux tienen la estructura que les permite ejecutar el programa en la memoria, el formato más común es llamado "ELF" (Executable and Linkable Format), este soporta objetos de 32 y 64 bits.

Se muestra en forma esquemática en la Figura 3-7, como se realizan las infecciones en el sistema sistema operativo Linux.

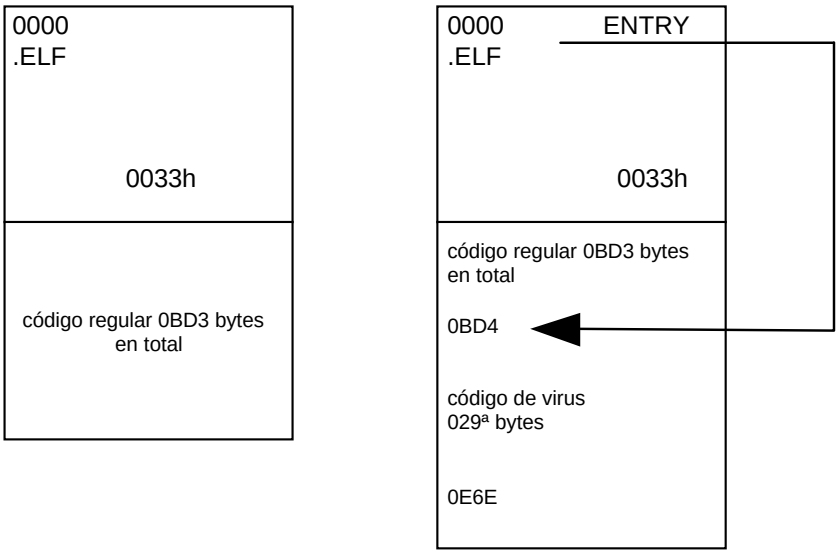


Figura 1-7 Un esquema de la infección del virus Lin Glaurung

En el esquema de la Figura 3-7, se puede observar cómo el virus *Lin Glaurung* modifica cierta información de la cabecera del archivo huésped para que él sea el primero en ejecutarse.

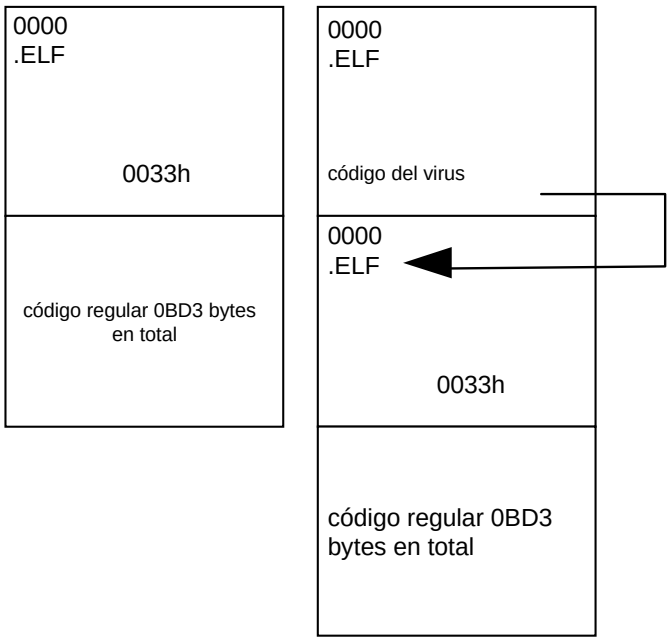


Figura 1-8 Esquema de infección del virus Obsiian E

En la Figura 3-8, se observa cómo el virus *"Obsiian E"* toma la posición inicial del programa y en el momento en que éste termina de ejecutarse, cede el control al programa original con la intención de ocultarse de cualquier sospecha que muestre su existencia, algo que se debe notar es que el virus no modifica al código en su parte inicial.

1.2 Los antivirus

Existen diferentes vacunas que van desde las más sencillas hasta las más elaboradas técnicamente hablando. Una de las más sencillas es obtener la firma de auto identificación del virus, colocándolo en los demás archivos *"sanos"*. Este experimento, se realizó con el virus *Jerusalén A*, que tiene la firma *"MsDOS"* y que se encuentra al final del archivo. En el momento en que el virus Jerusalén intenta contaminar a un programa, primero verifica si existe su firma, de ser positiva la identificación no se realiza la contaminación. Por el contrario se inserta el virus dentro del archivo huésped.

La técnica explicada con anterioridad es bastante efectiva, pero en la vida real no es posible colocar las más de 60,000 firmas dentro de un archivo para que este no pueda ser contaminado. Sin tomar en cuenta casos especiales como el virus *Jerusalén B* donde éste no verifica al archivo huésped, lo que provoca que se contaminen repetidas veces en el futuro.

Para ver esto de una manera más clara se analizarán y diseñaran las técnicas más utilizadas por los antivirus para mantener al margen a los virus informáticos.

Un algoritmo de fuerza bruta que busque firmas tiene las siguientes inconvenientes para identificar a los virus que cambian parte de sus datos, esto se puede observar con el siguiente ejemplo:

```
Instrucciones
Jmp    IP+ Datos arbitrarios1
Datos arbitrarios1
Instrucciones
Jmp    IP+Datos arbitrarios2
Datos arbitrarios2
Instrucciones
```

Código 1-2 Algoritmo básico en la construcción de un virus.

En el ejemplo del Código 3-2 se observa lo siguiente. Si los virus llegan a modificar el área de *Datos arbitrarios 1* o *2* en cada ejecución, la firma del virus cambiaría en una porción bastante considerada, lo que provoca que se den más falsos positivos o negativos. Para esto, una solución sería la de colocar comodines "?" que permitan ignorar el área o segmento que se encuentra en constante cambio. A esto se le llamó búsqueda basada en un autómata o búsqueda heurística.

1.2.1 Identificación de virus informáticos

Los expertos en virus informáticos pueden llevarse muchos días en encontrar una buena firma, con la intención de crear la herramienta que pueda identificar a un programa en forma única. IBM ha desarrollado un método estadístico para extraer automáticamente firmas de un virus informático [24]. La idea básica consiste en recopilar una gran cantidad de programas, y después utilizar esta información para estimar las probabilidades de falsos positivos en las firmas creadas. En la práctica, las firmas extraídas con este método son menos probables de generar falsos positivos.

Otra forma, en la que se puede extraer una firma que identifique a un tipo de virus, es buscar en un archivo una secuencia de bytes que pueda compararse con otro virus. Así de esta forma estimar una firma que produzca menos falsos negativos y falsos positivos. Para esto se muestra un ejemplo de cómo extraer una firma de un virus. Se tienen dos archivos de diferentes tamaños; *Archivo 1* y *Archivo 2,* donde cada uno se encuentra contaminado con el mismo tipo de virus.

Figura 1-9 Inicio en la extracción de una firma.

Se analizan dos archivos infectados con el mismo virus, como se muestra en la Figura 3-9. De esta manera, se extraen los dos códigos y se procede a una comparación de 16 bytes para tener una probabilidad aceptable que no permita producir falsos positivos.

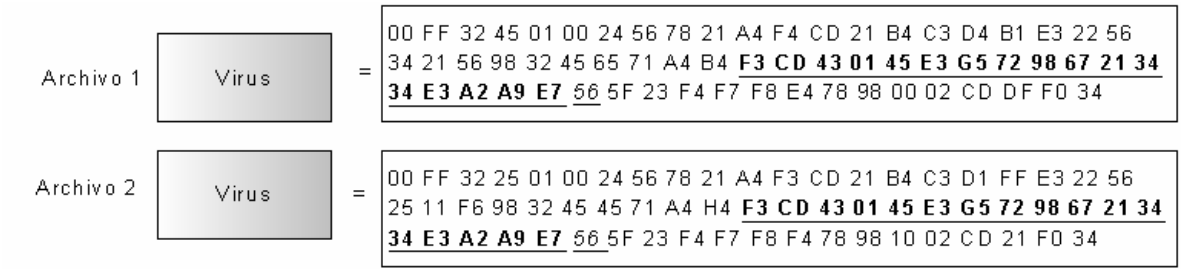


Figura 1-10 Búsqueda de una firma aproximada.

Como se observa en la Figura 3-10 se localizan los primeros 16 bytes, esta firma, verifica a los programas o virus ya registrados para analizar si no existe un falso positivo. Esto es que la firma se encuentre en algún otro segmento del programa que pueda causar una confusión al momento de realizar la búsqueda con los antivirus. Si la firma se encuentra en algún otro programa o virus informático se busca otra sucesión de bytes.

¿Qué pasa en el instante en que no existe alguna sucesión de bytes con la longitud especificada? Se procede a buscar el patrón con la utilización de los comodines.

Se supone que la firma localizada en la base de datos se encuentra en un programa que no tiene virus; a este hecho, se realiza la búsqueda con el siguiente candidato ver la Figura 3-11.

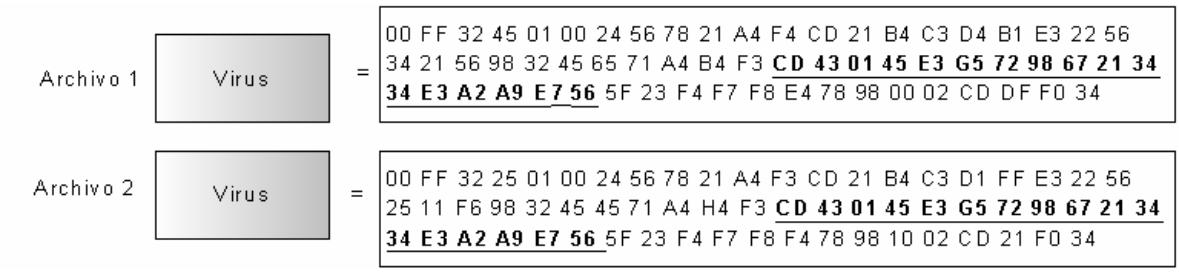


Figura 1-11 Búsqueda de una nueva firma.

Si la nueva firma tuviera una probabilidad mínima de ser encontrada en algún otro programa que no tuviera este tipo de virus, el algoritmo terminaría. En el caso de que no exista la sucesión de 16 bytes, se utilizan comodines como se observa en el Apéndice C.

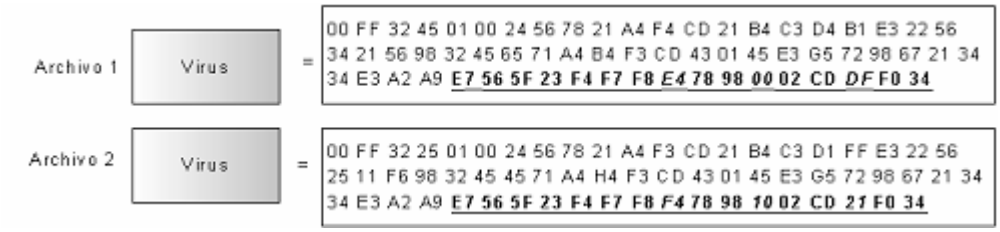


Figura 1-12 Búsqueda de una firma basado en comodines.

Como se observa en la Figura 3-12, se ha tomado una sucesión de bytes de cada uno de los archivos.

Archivo 1 tiene una sucesión 1 = E7 56 5F 23 F4 F7 F8 E4 78 98 00 02 CD DF F0 34

Archivo 2 tiene una sucesión 2 = E7 56 5F 23 F4 F7 F8 F4 78 98 10 02 CD 21 F0 34

Se tienen varios bytes que coinciden entre la sucesión 1 y la sucesión 2, pero existen diferencias, éstas son mostrados dentro de un recuadro. Para obtener una firma significativa los bytes que son mostrados en recuadros son remplazados por los comodines, por lo que la firma quedará de la siguiente manera.

Firma 1 = E7 56 5F 23 F4 F7 F8 ?? 78 98 ?? 02 CD ?? F0 34

El problema de estas firmas es que se debe mantener un porcentaje mayor de bytes encontrados que de comodines. El propósito es el de no producir falsos positivos o falsos negativos.

Sí el algoritmo no encuentra una firma significativa, entonces se habla de la existencia de dos firmas para el mismo tipo de virus. Se observa los resultados siguientes de las firmas VCL (Virus Create Laboratory) que identifican al mismo tipo de virus.

VCL (encrypted) virus	COM	01B9????8134????4646E2F8C3
VCL (encrypted) virus	COM	01B9????8135????4747E2F8C3
VCL virus	COM	ACB90080F2AEB90400ACAE75??E2FA89
Yankee Doodle Related	COM EXE	53BB??%2F8B8%2CD215B

1.3 Herramientas para la corrección de los programas infectados

La idea básica es que un programa tenga la capacidad de corregir cierta área de datos en el momento en que ésta se haya alterado. Este experimento se llevó a cabo en 1994 con un proyecto denominado Multi-Protect, donde se analizaban partes de programas huésped para luego colocarlos en un archivo de recuperación o simplemente al final del archivo [36]. De esta manera si el archivo se encuentra infectado, se podría reconstruir en un índice del 80% de éxito.

- La primera fase T_1, será crear un archivo al cual se le llamará “Programa ejecutable”.
- La segunda fase T_2 obtendrá parte de las estructuras iniciales de dicho programa y tal formación se alojará al final del archivo mismo.
- La tercera fase T_3, se contamina con un virus informático al “Programa ejecutable” Por lo que ya se ha visto en el punto 3.1, los virus tienden a modificar ciertas instrucciones del programa huésped.
- La cuarta fase T_4, se elimina el virus del archivo y se reconstruye el “Programa ejecutable”, dejando al archivo en la forma de T_1.

Este proceso se puede observar en la Figura 3-13.

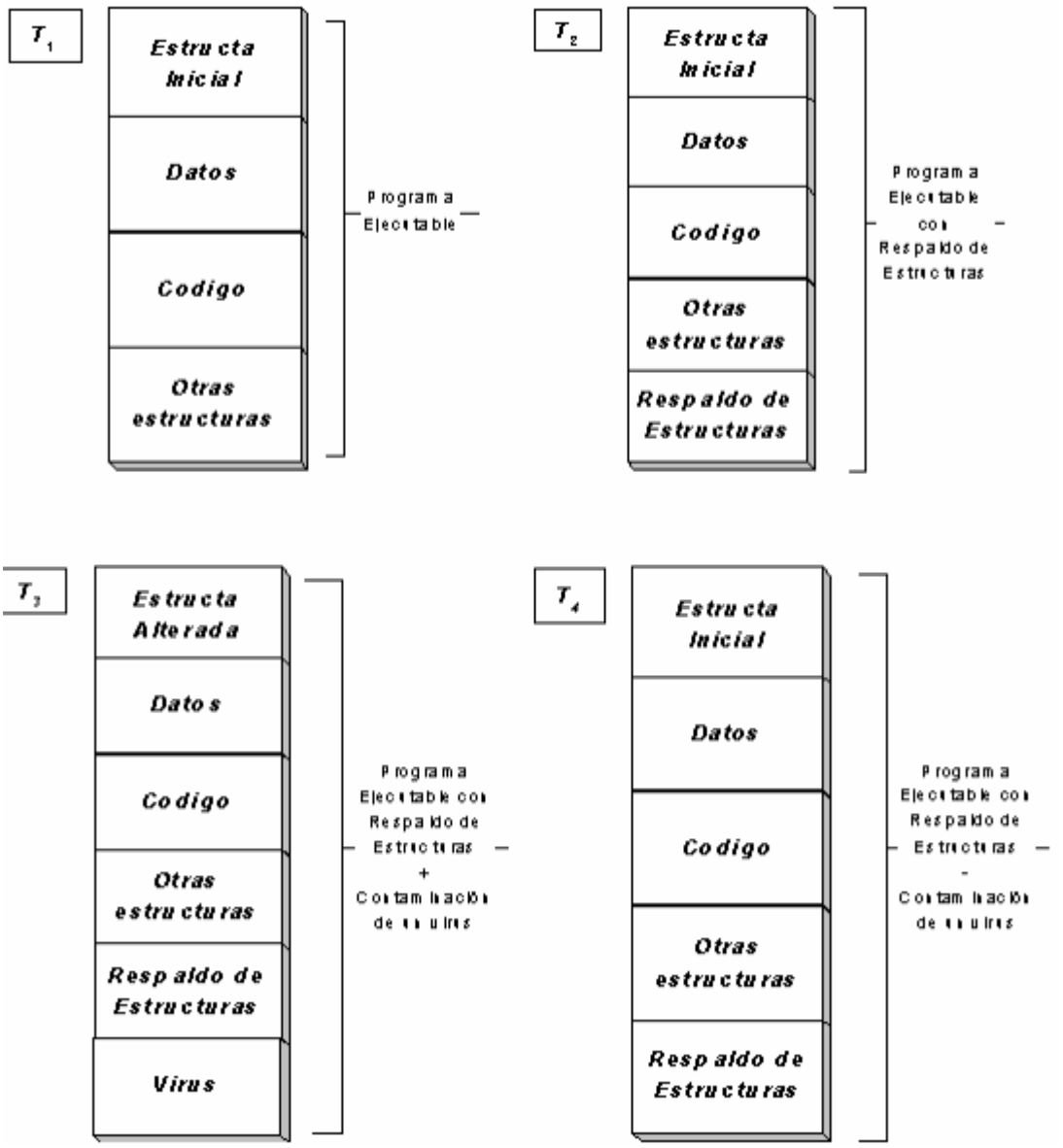


Figura 1-13 Reconstrucción de un archivo infectado por la técnica Multi-Protect

Lo que se observa en la Figura 3-13 parece ser una medida eficaz, pero se presentaron algunos inconvenientes al momento de implantarlo. Esto es porque algunos archivos utilizan la misma técnica con el fin de verificar algún registro de derechos de autor.

Otro inconveniente que presenta la técnica anterior, es en el momento en que un grupo de desarrollo se encuentra en la realización de pruebas en los programas en desarrollo. En ese instante es complicado obtener una firma apropiada, ya que no habrá manera de corregir ni detectar la existencia de un virus. Otro problema es el momento en que un archivo está contaminado, lo que origina que el virus se proteja con esta técnica, además que se crearon problemas en los antivirus comerciales, ya que se dificultaba la detección del virus, cuando éste también se encontraba protegido.

1.3.1 Eliminar el virus llamado ping pong

Este tipo de virus, se coloca en los discos en su sector inicial. Como se observó en el Capítulo 2 la parte original es movida a otra posición del disco. Para eliminar este virus, primero se debe conocer la firma que lo identifique de forma positiva y mover el código desplazado a su posición original, de tal manera que se reconstruya el daño causado por el virus.

```
1.  #include <dos.h>
2.  void main(){
3.     char         unidad, Buffer[512];
4.     unsigned     Boot_Original;
5.      printf("Este programa detecta y elimina el virus Ping-Pong \n");
6.      printf("Introduzca la letra de la unidad a verificar");
7.      scanf("%d",&unidad);
8.      absread(unidad,1,0,Buffer);
9.      if ( Buffer[507]==0 && Buffer[508]==0x57 && Buffer[509]==0x13 )
10.     {
11.         printf("El disco se encuentra infectado por el virus Ping-Pong \n");
12.         Boot_Original = Buffer[506]*0x100+Buffer[505]+1;
13.         printf("El sector donde está el Boot original: %d \n",Boot_Original);
14.         absread(unidad,1,Boot_Original,Buffer);
15.         abswrite(unidad,1,0,Buffer);
16.         printf("Virus eliminado \n");
17.      }
18.      else
19.           printf("El disco no contiene el virus Ping-Pong\n");
20. return;
21. }
```

Código 1-3 Eliminación del virus Ping-Pong.

En el Código 3-1 en la línea marcada con el número 10 se obtiene el sector inicial, éste es el lugar donde se aloja el virus al momento de infectar. En la línea 11 se compara con una firma o una sucesión de bytes que es “00 57 13”. La probabilidad de encontrar el virus es alta, debido a que en este sector únicamente tiene 420 bytes. El resto del sector está dado para la estructura del disco.

En la línea 14 del Código 3-3, se calcula el lugar en donde se encuentra el sector *“original”* o sector anterior a la contaminación del virus, posteriormente procede a leerlo y copiarlo a su posición original y así eliminar el virus.

1.3.2 Eliminar el virus Viernes 13

Este virus se agrega a los archivos con la extensión EXE y COM. En el ejemplo siguiente, únicamente se realiza la vacuna para eliminar el virus de los archivos que tengan la extensión .COM, ya que estos tienen una firma al final “MsDos”. Para detectar si un archivo tiene virus se verifica al final de éste si existe dicha firma. Por razones de simplicidad, el siguiente ejemplo sólo funciona con archivos menores a 60K.

En la línea 13 hasta la 17 se obtienen los últimos 5 bytes para verificar si son iguales a la firma “MsDos”, si es localizada e identificada se elimina. Lo que se hace es colocar en una memoria intermedia el archivo; para eliminar los primeros 1808 y los últimos 5 bytes, qué sumados dan la cantidad de 1813. En la línea 26 hasta la 31 se extrae el código original, el cual es colocado nuevamente en el archivo, pero en la posición cero del archivo, de esta manera se elimina el virus informático.

```
1 #include <stdio.h>
2 #include <dir.h>

3 void main(){
4       unsigned char firma[5],buffer[61441];
5       unsigned tamanio;
6       FILE *Fp;
7       struct ffblk Archivo;
8
9       printf("Programa para detectar el virus Viernes 13\n");
10      if ( !findfirst( "*.COM",&Archivo,0x20) )
11              do{
12                      printf("%-12s",Archivo.ff_name);
13                      Fp = fopen(Archivo.ff_name,"rb");
14                      fseek(Fp,-5L,SEEK_END);
15                      firma[5]=´\0';
16                      fread(firma,5,1,FP);
17                      if( !strcmp(firma("MsDos")){
18                              printf("Archivo infectado\n");
19                              tamanio = ftell(FP);
20                              if (tamanio >61440 )
21                              {
22                                      printf("No es posible desinfectarlo \n");
23                                      break;
24                              }
25
26                              fseek(FP,1808L,SEEK_SET);
27                              fread(buffer,Tamanio-1813,1,FP);
28                              fclose(Fp);
29                              Fp = fopen(Archivo.ff_name,"wb");
30                              fwrite(Archivo,Tamanio-1813,1,Fp);
31                              printf("Virus Eliminado");
32                      }
33                      else printf(" =>Ok \n");
34                      fclose(Fp);
35              }while(!findnext(&Archivo));
36 return;
37 }
```

Código 1-4 Eliminación del virus viernes 13

En el Código 3-4 se observa que con una simple cadena se detecta a un virus, también se producen falsos positivos. Estos se dan, en el momento en que se coloca la misma firma al final de un archivo sano, lo que provoca una falla al realizar la detección.

1.4 Problemas abiertos en la investigación de los virus informáticos

Hoy en día, aunque para muchos parece que el problema de los virus se encuentra bajo control debido a que los antivirus se actualizan constantemente; en el área de la investigación falta por resolver algunos problemas que los virus han generado. IBM propone varios problemas abiertos de investigación [37].

- Cada día se desarrollan nuevos lenguajes, con lo que se dan nuevas posibilidades de crear nuevos virus informáticos; esto lleva como consecuencia el mejorar sustancialmente las búsquedas heurísticas para predecir cualquier tipo de infección.
- Se posee una compresión razonable y cualitativa de la epidemiología al momento de tratar con virus informáticos, así como analizar las características de su propagación en términos de natalidad, mortalidad y los modelos de transferencia de programas entre los sistemas de cómputo. Aunque su aplicación únicamente se ha utilizado para estudios estadísticos, la Epidemiología puede ayudar a prevenir nuevas infecciones.
- La creación de un sistema inmune que permita eliminar y diagnosticar los futuros virus, tomando como base la evolución del software en términos de seguridad.
- Los gusanos, han causado problemas serios al momento de detectarlos, ya que no se alojan en el sistema de archivos de la computadora, y por lo contrario utilizan la red para moverse de lugar en lugar. Se requiere de una nueva forma de tratar a este tipo especial de virus [6].

1 EL SISTEMA INMUNE

Capítulo 4

Resumen

En este capítulo se presenta el funcionamiento básico del sistema inmune de los vertebrados, así como las analogías que se han realizado para eliminar a los virus informáticos. Además de introducir y formalizar el nuevo modelo del sistema inmune basado en el sentido innato del peligro, aunado al desarrollo de una firma llamada señal de peligro que permite distinguir en que momento un programa se encuentra alterado y/o modificado. Posteriormente se presenta la aplicación correspondiente que permite experimentar con las diferentes especies de virus y el modelo del sistema inmune propuesto.

Objetivos del Capítulo

- Análisis del sistema inmune de los vertebrados.
- Estudio del sentido innato del peligro en los vertebrados.
- Definir un modelo llamado señal de peligro basado en el modelo del sentido innato del peligro.
- Se define el funcionamiento del modelo del sistema inmune para los virus informáticos a niveles micro y macro.
- Describir la implantación e integración del sistema inmune para prevenir a los virus informáticos.
- Presentar resultados de la señal de peligro que activa al sistema inmune para prevenir y eliminar los virus informáticos.

1.1 Sistema inmune natural

Se presenta un modelo de un sistema inmune para prevenir, evitar la propagación y eliminar a los virus informáticos, en él se lleva a cabo un análisis en diferentes topologías, basándose en la teoría del sentido innato del peligro. Para realizar este estudio, es necesario definir su comportamiento dentro de una Máquina de Turing Universal, en donde se presentan varios problemas intrínsecos al momento de realizar la detección de un virus. Así que, se propone, una solución donde se crea una firma basada en una representación abstracta de un programa, llamándola señal de peligro. Esta firma permite realizar un diagnóstico para encontrar cualquier problema relacionado con los virus informáticos. Las firmas de señal de peligro serán enviadas a una central de diagnóstico, en donde se realiza y se obtiene un resultado basado en la Epidemiología. Con este resultado, se emitirá otra firma, llamada respuesta inmune, que impedirá que el virus se siga propagando; posteriormente, se utilizarán los códigos que han sido alterados por el virus para estimar la vacuna apropiada.

La teoría clásica que más se han manejado en la inmunología de los vertebrados es:

> *"Es un grupo de células que discrimina entre lo propio y lo extraño* [38]*; ataca a los antígenos no nativos y son tolerantes de uno mismo (excepto en estados de la enfermedad). Se define a lo propio como los antígenos que se presentan en la vida temprana* [39]*."*

La teoría de Cheers Polly Matzinger se basa en el sentido innato del peligro, la cual dice:

> "Es un grupo de células que responde a señales endógenas que se originan en las células dañadas [40]."

1.2 Funcionamiento básico del sistema inmune natural

Antes de comenzar con la analogía entre el sistema inmune biológico y la seguridad informática, se analizará al sistema inmune de los vertebrados para comprender su funcionamiento esencial.

El sistema inmune de los vertebrados se compone por una variedad de células morfológicas y funcionalmente diferentes, que se diferencian a partir de células primordiales pluripotenciales[1]. Todos estos tipos celulares ejercen funciones diferentes, que interaccionan constantemente; estas interacciones, pueden estar mediadas por contacto físico o a través de factores solubles que ejercen su función en células con receptores específicos.

Las células que forman el sistema inmune se organizan a su vez en tejidos y órganos; éstas son estructuras que reciben el nombre genérico de sistema linfoide.

[1] Multiplicación de las células germinales del tejido hematopoyético, también cocidos como células germinales.

Todas las células del sistema inmune se originan a partir de células primordiales pluripotentes, las cuales siguen dos líneas fundamentales de diferenciación: El linaje linfoide y el linaje mieloide. Las células del linaje linfoide, son los linfocitos, que se diferencian en los tejidos linfoide primarios. Existen dos tipos fundamentales de linfocitos: las células T y las células B. Ambos tipos celulares poseen en su membrana receptores capaces de reconocer al antígeno de una forma específica.

La principal característica de los linfocitos B, es su capacidad de producir anticuerpos o inmunoglobulinas "ig". Estas moléculas constituyen el receptor específico para el antígeno de las células. Las células T tienen un receptor de membrana de estructura similar a las inmunoglobulinas, conocido como receptor de la célula T "TCR". Mediante este receptor, los linfocitos T, son capaces de identificar el antígeno de forma específica; a diferencia de los linfocitos B, las células T necesitan que el antígeno, sufra una serie de modificaciones antes de que se pueda reconocer.

Existen dos tipos fundamentales de linfocitos T: los linfocitos T citolíticos (Tc), que son portadores de la molécula CD8 en su membrana plasmática, y los linfocitos T cooperadores (Th, Helper), que expresan la molécula CD4. Los linfocitos Tc detectan a los péptidos, que son presentados por moléculas MHC de clase I, su función es eliminar (lisar) las células que presentan a los péptidos que son extrañas al organismo, por ejemplo péptidos de virus. Los linfocitos Th, reconocen a los péptidos en el momento en que están unidos a moléculas MHC de clase II, su función, es la de ayudar a los linfocitos Tc, así como a las células B y los fagocitos.

1.2.1 Células presentadoras de antígeno profesionales

Todas las células nucleadas (cuestionadas) del organismo, expresan moléculas MHC de clase I en su membrana, y son, por lo tanto, susceptibles de presentar los péptidos a los linfocitos Tc. Sin embargo, sólo algunas células con funciones inmunológicas son capaces de expresar moléculas MHC de clase II en su superficie.

Las células NK, (natural killer) son linfocitos con actividad citotóxica o citolítica innata. El origen ontogénico de estas células es desconocido, aunque se sabe que no maduran en el timo. Todo parece indicar que se trata de un conjunto heterogéneo de células, que pueden tener varios receptores diferentes. La característica común sería su capacidad de lisar[2] determinadas células; por ejemplo células tumorales o infectadas con determinados virus, sin una previa sensibilización a diferencia de lo que ocurre en el caso de los linfocitos Tc. Las células NK también son capaces de lisar otras células que por alguna razón han perdido su capacidad de expresar moléculas MHC de clase I en su superficie, la ausencia de moléculas MHC de clase I en una célula provoca que ésta sea atacada por las células NK. Al contrario de lo que ocurre con los linfocitos Tc, que requiere la presencia de moléculas MHC de clase I para reconocer el antígeno presentado por la célula diana [41].

[2] Eliminar al patógeno de la célula

1.2.2 Los precursores de las células del sistema inmune de los vertebrados

Todas las células del sistema inmune se originan a partir de las células hematopoyéticas primordiales pluripotenciales de la médula ósea, a través de los linajes mieloide y linfoide. Los precursores de los distintos tipos celulares producidos en la médula ósea migran hacia otros órganos, los cuales sufren ulteriores procesos de diferenciación para convertirse en células plenamente funcionales.

Los tejidos linfoides periféricos, son lugares de interacción entre los linfocitos y los antígenos, para que los linfocitos B y T inmonucompetentes recién formados puedan interaccionar entre sí y con los antígenos, de esta manera, se podrá generar y diseminar una respuesta inmune adaptativa celular y/o humoral específica, donde cada órgano linfoide secundario es el encargado de controlar una determinada región del organismo.

Se dice que el sistema inmune está compuesto por una variedad de células con funciones diferentes, y tienen una interacción entre sí, y que las células responsables de desencadenar la respuesta inmune específica son los linfocitos T y B. Otras células como son los fagocitos monucleares, las células dendríticas y los linfocitos B, tienen la misión de capturar antígenos, que posteriormente procesarán para que puedan ser reconocidos de una forma adecuada por los linfocitos T. Por otra parte, los linfocitos NK son células con actividad citolítica innata y a diferencia de los linfocitos Tc, no requieren una reactivación para lisar sus células diana.

1.2.3 Los linfocitos B

La respuesta inmune adaptativa humoral está mediada por los linfocitos B, estas células son capaces de reconocer antígenos solubles y sintetizar anticuerpos específicos, para ello se valen de una estructura de membrana que constituye su receptor para antígenos que es análogo al TCR de los linfocitos T. Se pretende dar una visión global de los eventos moleculares asociados a la activación B, así como una descripción de estas células.

Función: Los linfocitos B reconocen antígenos nativos y presentan sus fragmentos a los linfocitos T.

Los linfocitos B tienen dos funciones esenciales en el sistema inmune: actúan como células productoras de anticuerpos y también como células presentadoras de antígeno altamente eficaces.

Una vez que se produce la interacción con el antígeno, la célula B puede activarse y se pone en marcha la respuesta inmune humoral, las células B pueden actuar para presentar los antígenos de las células T e inducir a la respuesta de tipo celular. La presentación del antígeno, supone la interacción inicial con él, en su forma nativa y su posterior procesamiento, para que al final, exponga sus fragmentos en la membrana asociada a las moléculas MHC.

1.3 El sentido innato del peligro

En el momento en que se realiza el estudio del funcionamiento básico del sistema inmune de los vertebrados, se encontró que una de las primeras personas que había realizado una analogía entre el sistema inmune natural y la seguridad informática era Stephanie Forrester, ella ve al "*Sistema Inmune de los vertebrados, como una rica fuente de inspiración para la seguridad en informática, en la cual se argumenta que tiene muchas características que se asemejan a los problemas que presenta la informática en cuestión de seguridad [42].*"

Al analizar el artículo de Forrester, "Los principios de un sistema inmune informático [42]", se encontró que estaba basada en la teoría clásica del sistema inmune de los vertebrados, visto en el punto 4.1. Al momento de estudiar su analogía, dice que el problema en la protección de los sistemas en computación puede verse como un problema general del estudio y la distinción entre lo propio y lo extraño [43,44].

Durante el estudio del sistema inmune de los vertebrados se encontró que la Doctora Polly Matzinger en su artículo "Un sentido innato del peligro" [45], explica la mayoría de las características de la inmunidad, además, posee algunos problemas esenciales, por dar un ejemplo, "si lo propio" es aprendido durante la vida temprana, vida fetal o natal, surge la pregunta ¿Qué sucede con la pubertad o durante la lactancia? Aunque el sentido "evolución" afecta en el cambio de los individuos, y los cambios que ocurren en uno mismo, no son el único problema con la teoría explicada en el punto 4.1.

Para Polly existen cuatro funciones que realiza el sistema inmune; son el peligro, la muerte, la destrucción y señal de socorro, por lo que abandona la definición de la discriminación entre lo propio y lo extraño.

1.3.1 Las diferencias entre el modelo de lo propio y extraño y el modelo del peligro

Hay particularidades en donde estos dos modelos son iguales; ambos modelos acentúan las funciones críticas de la inmunidad natural de los vertebrados y muchas de las reglas fundamentales son similares. Sin embargo, una diferencia crucial entre el modelo del peligro y el modelo de lo propio y extraño (SNS: Self o Noself), es la manera en que se realiza el lanzamiento de una inmunorespuesta. El modelo del SNS, admite que los cuerpos extraños accionan una respuesta natural, adaptándose al patógeno. Mientras que el modelo del peligro, admite que lo que realmente importa desde el punto de vista evolutivo, es si se estropea la entidad celular o no. Si la entidad celular no sufre cambios esenciales se presume que el ambiente es sano y si ésta muere por causas normales, ésta se elimina y por lo tanto no existe inmunorespuesta. Únicamente si la célula tiene una muerte provocada o se daña, se inicia la inmunorespuesta específica la cual identifica al patógeno y lo destruye respectivamente. Los dos modelos mencionados presentan las siguientes diferencias esenciales.

1) El origen de la señal.
2) La naturaleza de las células que controlan inmunidad.

1.3.2 La señal que inicia la inmunorespuesta

El modelo SNS, admite que el sistema inmune se encuentra dando vueltas por el exterior en busca de señales exógenas que representan a lo propio o extraño [46]. En cambio, el modelo basado en el peligro, sostiene que el sistema inmune está gobernado desde adentro, y que responde a las señales endógenas que se originan en las células dañadas [40].

Para generar una señal de peligro y una respuesta inmune se debe analizar el modelo desde dos perspectivas: la primera se proporciona a nivel Micro, donde una computadora tiene que autovacunarse en forma local, y la segunda llamada Macro, que es la encargada de resolver el problema en el ámbito distribuido. Además de ver las topologías en las que operan. En el nivel Micro se encuentra el análisis realizado por IBM en modelo de propagación *"Susceptible infecta a Susceptible"* en donde se presentan las siguientes gráficas.

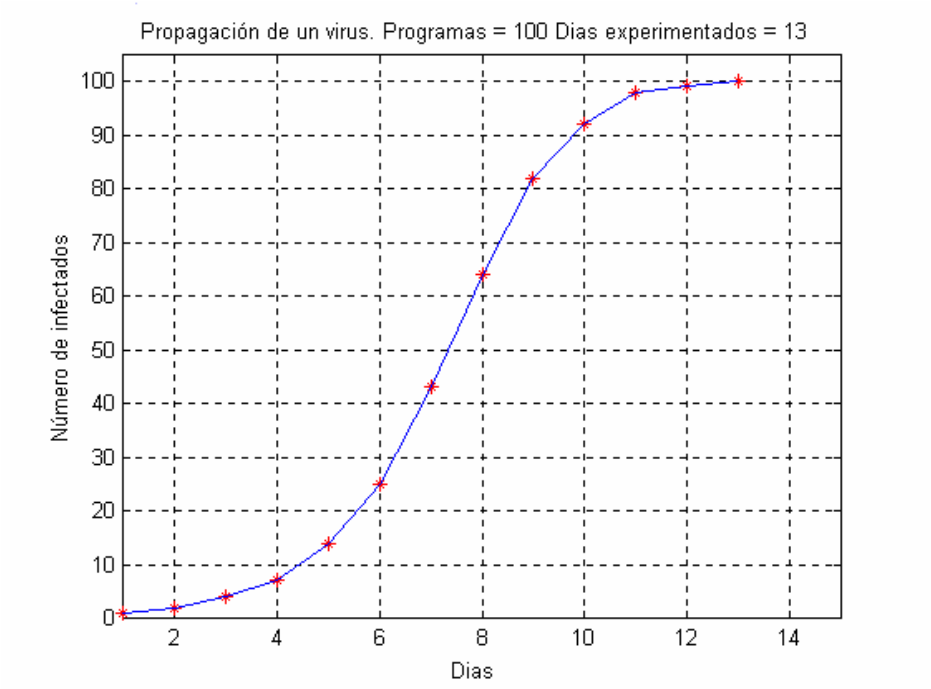


Figura 1-1 Propagación de un virus informático.

En la Figura 4-1 se observa cómo la propagación se da en forma exponencial. Para establecer una restricción mayor en el modelo de propagación *Susceptible infecta a Susceptible*, se planea un modelo del sistema inmune que funcione en cada equipo y de forma local.

1.4 Un modelo del sistema inmune a nivel micro

Para establecer un paralelismo más cercano con la teoría de Cheers Polly Matzinger, se definen algunos conceptos basados en su modelo del peligro, esto ayudará a estudiar el tema de una manera más cercana. Hablar de los virus informáticos es hablar de los patógenos, y en

el momento en que se habla de una infección, es el instante en que éste se introduce en un sistema, para ello existen dos tipos de infecciones: las líticas y las liso génicas.

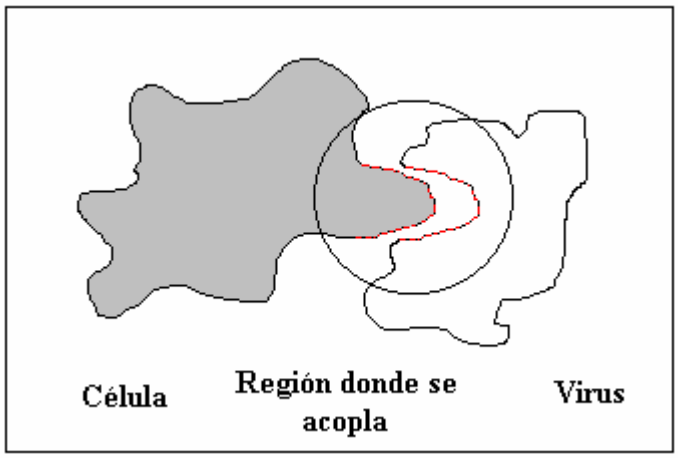


Figura 1-2 Infección lítica.

Como se observa en la Figura 4-2 se genera una infección del tipo lítica, esto es, que existe un lugar específico donde el virus y la célula se acoplan o se unen. Se deduce que el virus conoce la estructura esencial de la célula; este mismo efecto sucede con los virus informáticos, se investigaron las diferentes formas en las que se realiza esta unión dentro de los formatos más comunes como son COM, EXE, NE, PE y ELF. En la Figura 4-3, se ve claramente el porque se le llama "acoplarse".

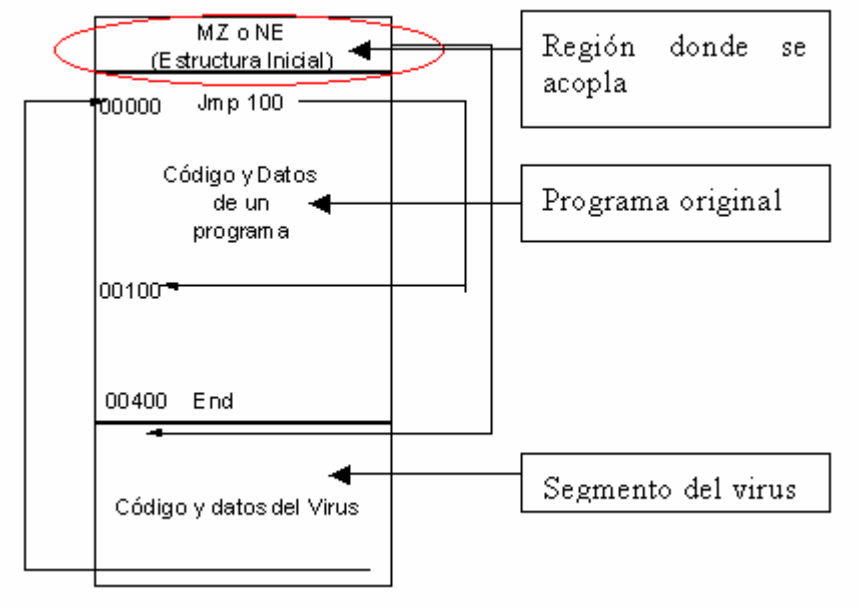


Figura 1-3 Infección lítica en un archivo ejecutable EXE.

En la Figura 4-3, se muestra una región circular que indica en donde se acoplan los dos segmentos de código. Esta región presenta una estructura que le indica al sistema operativo cómo se debe realizar la ejecución del programa, por eso es que los virus infectan a este tipo de archivos, ya que conocen perfectamente el área. Por lo contrario la expansión del virus estará limitada por su autodestrucción.

El otro tipo de infección es la liso génica que es donde el virus no se acopla, sino que se agrega al cuerpo del programa.

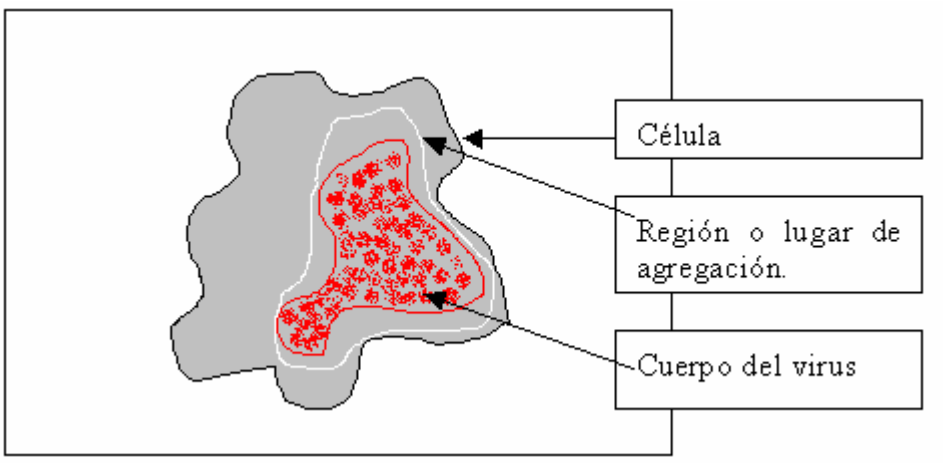


Figura 1-4 Infección liso génica.

En el Capítulo 2, se muestra cómo los Macros de Visual Basic y los Gusanos proponen el esquema de contaminación mostrado en la Figura 4-4.

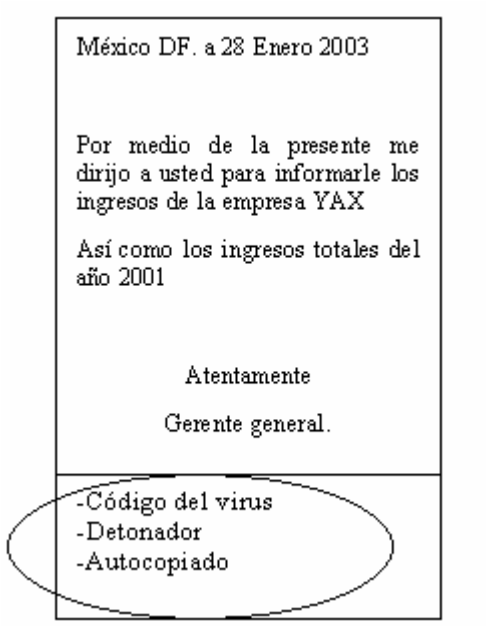


Figura 1-5 Infección liso génica a un documento.

En el área marcada con un círculo de la Figura 4-5, es el lugar donde se aloja este virus, en donde presenta una infección del tipo liso génica. Para eliminarlo, únicamente hay que conocer el área y quitarlo, esto se realiza de forma directa o por medio de alguna herramienta.

1.4.1 Cómo se acoplan los virus informáticos

En la formalización de los virus informáticos que presenta Cohen, muestra claramente que un elemento del conjunto de los virus puede autocopiarse en cualquier celda por todas las historias y todos los tiempos [14]. El análisis que realizó fue con las Máquinas de Turing, y la definición que presenta es abstracta, lo que origina muchas preguntas que plantearemos y responderemos a continuación, los estudios que se harán serán pensados en las Máquinas de Turing Universales, y el comportamiento dentro de los lenguajes estructurados.

Al momento de realizar el estudio de los virus informáticos bajo un lenguaje y una máquina de Turing determinista, surgen las siguientes preguntas: ¿Un virus informático *w* escrito bajo las reglas que presenta un lenguaje estructurado puede ser interpretado por otra máquina que acepte otro tipo de lenguaje? Solamente si se cumple la siguiente definición donde se dice que un lenguaje $L_1 \subseteq \sum_1^*$ puede transformarse en $L_2 \subseteq \sum_2^*$, si existe una función t: $\sum_1^* \rightarrow \sum_2^*$, siempre y cuando sea una transformación polinomial y la función t sea computable [47].

Esto dice, que un virus que es programado para cierta gramática deberá respetar la definición anterior, lo que infiere que existe una restricción en la propagación del virus. Por ello se realizará un estudio esencial del comportamiento que éstos tienen bajo un lenguaje estructurado por frases que es lenguaje que interpreta cualquier máquina de Turing.

¿Un virus informático debe conocer a su huésped? En el sentido estricto no es necesario que conozca al huésped, basta con copiarse en cualquier parte, pero, para hablar de propagación, es importante tomar en cuenta esto. Ya que para que sea una amenaza el virus deberá acoplarse a los otros programas y funcionar de forma transparente para el usuario.

Si un virus está basado en una gramática que es interpretada por una Máquina de Turing, y su objetivo es el de copiarse en otra celda y otro tiempo, entonces el virus debe buscar un lugar donde pueda copiarse. Para analizar más claro esto, se realiza un estudio bajo las Máquinas de Turing y los Lenguaje Estructurados.

Una gramática estructurada por frases está definida de la siguiente forma:

$$G = (VT, VN, S, P)$$

VT = { Conjunto finito de terminales}

VN = {Conjunto finito de no terminales}

S = Es el símbolo inicial donde $S \in VN$

P = {Conjunto de producciones o de reglas de derivación}

donde:

$V+=(VN \cup VT)+$ y

$V*=(VN \cup VT)*$

Y donde se tienen producciones del tipo $\alpha \rightarrow \beta \in P$. En la cual $\alpha \in (VN \cup VT)+$ y $\beta \in (VN \cup VT)*$ la única restricción es de la forma $\lambda \rightarrow \beta$ donde λ es la cadena vacía.

Sean $\alpha 1$ y αm cadenas pertenecientes a V^+. Se dice que están en relación de derivación en la gramática G si existe $\alpha_1\alpha_2\alpha_3...\alpha_m$ donde se menciona que α_m se deriva de α_1, o que α_1 produce α_m [48].

Se denomina a una instrucción, a cualquier cadena que sea el resultado último de una derivación a partir del símbolo inicial S y que está compuesta únicamente por símbolos terminales [49].

$$S \rightarrow \alpha m \qquad y \qquad \alpha m \in VT$$

Ahora que se tiene la definición formal de una gramatica, se pasará a analizar el comportamiento de los virus dentro de un lenguaje y cómo éstos se insertan dentro de otra sentencia o instrucción. Para esto se escribe una gramática sencilla, que permita mostrar las características que tienen los virus al momento de acoplarse a su huésped. Esta gramática tendrá el nombre de G [48].

G=({main, código, virus},{C}, S,{S→main(C);,C→ código; C, C → virus, C → λ; C}

$S \rightarrow$ main(C);
$C \rightarrow$ código; C

$C \rightarrow$ virus; C

$C \rightarrow \lambda$

La gramática anterior muestra un pequeño lenguaje, en el cual se escribirá un programa como el siguiente:

p_1={ main (código; código);}

Ahora bien si se intenta escribir un virus, se deben de tener en cuenta dos aspectos: El primero tiene que ver con la inserción del virus dentro de una relación del tipo sentencia, de esta forma el virus destruiría su única forma de propagarse:

p_2={main (có**virus**digo; código);}

El virus no se propaga debido a que $p_2 \notin L(G)$.

La segunda es que el virus pertenezca a la misma gramática:

p_3={main(código; código; Macro_virus);}

Al ejecutar el programa p_3 con la gramática G se produce una derivación que no existe (Macro_virus) y por lo tanto, el virus no es interpretado, debido a que $p_3 \notin L(G)$.

Para que el virus pueda ser interpretado por una Máquina de Turing, se deberá tener algunos de los programas siguientes:

p_4={main(código; código; virus);}

p_5={main(código; virus; código);}

p_6={main(virus; código; código);}

ya que $p_4, p_5, p_6 \in L(G)$.

Se observa algo que es importante y que también ayuda a minimizar la propagación de los virus: todos los virus escritos en lenguajes estructurados por frases, deben conocer un lugar donde insertarse. Además, deberá tener todo o parte de la gramática con la que se creó el huésped.

Lo anterior no es una regla, pero al estudiar el concepto propagación, estas condiciones son importantes y deberán tomarse en cuenta. Nada impide que los virus escriban en cualquier parte de un programa para copiarse, pero sí para insertarse, ya que deberá respetar el sentido sintáctico y semántico del huésped.

Esto parece correcto para los virus, pero falta analizar la parte que representa a la semántica que dice: "*El sentido que se le da a una frase es correcto en el momento en que todos sus elementos sean verdaderos [50]*". Ahora bien sí un virus requiere ser correctamente ejecutado [51] deberá insertarse en un lugar donde tenga sentido su propio código. Por citar un ejemplo, suponemos que un virus busca la cadena o patrón "aquí" en un código y se inserta después de éste. Se observa en la Figura 4-6, que el virus se coloca en una área donde la información no

será ejecutada ya que los programas, por lo regular, tienen código y datos. La diferencia entre estos dos conceptos, es que uno es interpretado por una máquina de Turing y el otro no.

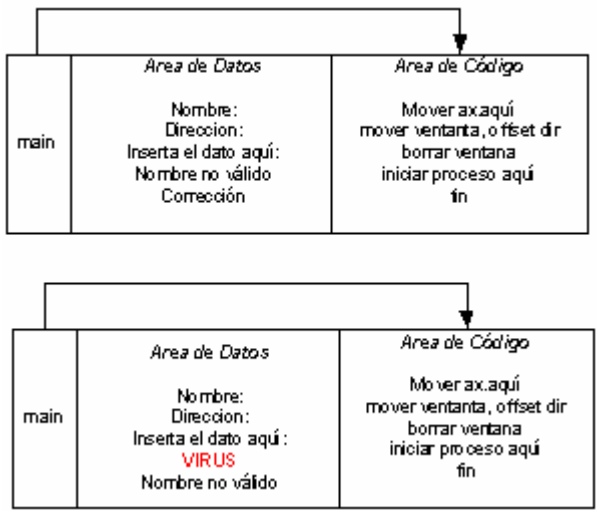


Figura 1-6 Infección de un virus en un área de datos.

Los virus deben tener un conocimiento sintáctico y semántico de su huésped, para ello deben basarse en la teoría que establece que una máquina de Turing universal puede simular cualquier Máquina de Turing Universal [52]. Ahora bien, sí los virus poseen una *Máquina de Turing* que sea capaz de simular a un programa *p*, tendría que realizar un recorrido completo, además de resolver de manera heurística los problemas de decisión dentro del programa.

1.4.2 Interpretación abstracta

La técnica de la interpretación abstracta permite realizar analizadores de programas definiendo un dominio de valores abstractos y una versión también abstracta [53] del programa [54]: la abstracción de los valores de datos es una descripción de propiedades que cumplen los datos de la versión abstracta del programa. El análisis obtiene información de las propiedades que cumple el programa a partir de simular la ejecución de la versión abstracta del programa con datos abstractos. Se trata de utilizar esta técnica para realizar un analizador automático (programas que analizan otros programas).

Una ventaja de los lenguajes de programación lógica (en general, de los lenguajes declarativos) respecto a otros lenguajes más estructurados, es la facilidad de definir semánticas formales sencillas, basadas en conceptos puramente lógicos. Esta característica facilita el análisis de los programas, ya que permite razonar sobre dicha semántica puramente lógica, en lugar del programa original.

Se propone el estudio de las posibilidades del análisis de programas lógicos basado en interpretación abstracta de forma práctica, en el cual se manejan analizadores implementados, extendiéndolos a nuevos dominios abstractos o a nuevos tipos de análisis. Esto no excluye la

posibilidad de realizar pequeños estudios teóricos relativos a las propiedades de los nuevos dominios o análisis.

1.4.3 Crear una señal de peligro

Como se vio en el epígrafe 4.3.2 existe una señal de peligro que alerta al sistema inmune natural el cual indica que inicie su respuesta para contraer el antígeno, y para establecer un paralelismo con el funcionamiento de los virus informáticos, se debe establecer una señal que indique en qué momento un programa ha sido contaminado por un virus informático. La historia ha proporcionado una firma comúnmente conocida como suma de validación (checksum), el cual proporciona una probabilidad [55][56-58]de 2^{32} [59]. Esta característica ayudará a conocer en qué instante un programa ha sido modificado, aunque existen varios problemas, los cuales han provocado que sea abandonado. Una de las principales es que existen empresas de desarrollo que realizan programas que estén en constantes cambios y evolución.

El examinar un programa completo para obtener una firma es un problema complejo, por lo que se utiliza una forma heurística de obtenerla. Esto es debido a que dentro de un programa, existen variables que son proporcionadas de forma externa o decisiones que únicamente permiten moverse por una sola relación convirtiéndose en un problema, no polinomial (NP).

Si un virus, intenta infectar un programa bajo simulación, deberá resolver los problemas antes mencionados, además de analizar la complejidad de la gramática, es por ello que muchos virus solamente infectan al principio del programa.

Este inconveniente lleva a analizar a los virus informáticos y estudiar su comportamiento al momento de ser ejecutados [5,60,61]. Para solucionar este problema, se plantea el obtener una firma basada en una interpretación abstracta que permita analizar a los virus en diferentes formas estructurales.

1.4.3.1 Buscar el lugar en donde se inserta un virus informático

El tomar varios tipos de virus informáticos que han existido, lleva a estudiar algo que ya ha sido resuelto de una u otra manera, por eso se decide tomar los llamados programas generadores de virus, como son el VCL (Virus Create Laboratory) "*Laboratorio creador de virus*", y Nuke, los virus informáticos son sencillos, pero permiten su modificación rápidamente, lo que da origen a nuevos tipos y formas de virus.

La primera forma gráfica que se obtuvo de un virus informático es la siguiente:

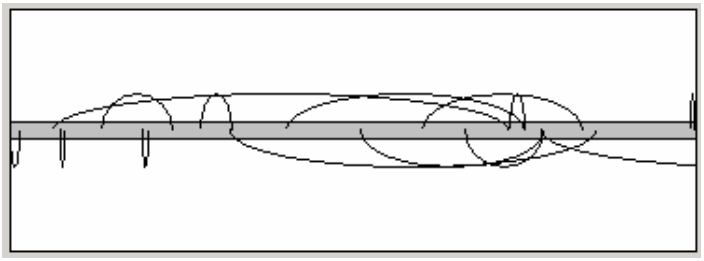


Figura 1-7 Vista gráfica de la interpretación abstracta de un virus.

La firma observada en la Figura 4-7, es una firma de forma heurística. La línea gris es el flujo de análisis del código, ya que por lo contrario se debe obtener una firma más compleja al tratar de solucionar los problemas de decisión.

Para la creación de estas firmas en donde se permitan observar todos los saltos y retrocesos también llamadas estructuras de control, así como su comportamiento básico que tiene al momento de la ejecución; se toman en cuenta gramaticalmente todas las sentencias en las que no se predice la ejecución, como son los saltos, ciclos y las funciones, ya que si un virus requiere tener una propagación rápida, deberá tener un análisis semántico de los lugares donde inicia o termina ese tipo de frases, que son aquellas donde es más probable que los virus se puedan acoplar al huésped.

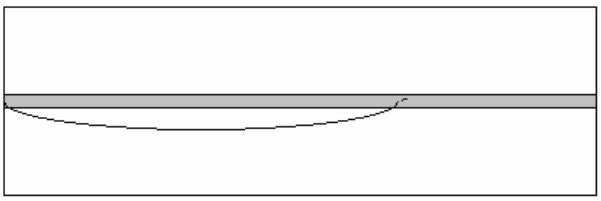


Figura 1-8 Vista gráfica de un archivo sin virus.

Para estudiar este fenómeno de una forma más clara, se toma un archivo sin virus (ver Figura 4-8), posteriormente se contamina con un virus informático creado con el generador de virus llamado NUKE, por último se obtiene una imagen como la que se observar en la Figura 4-9.

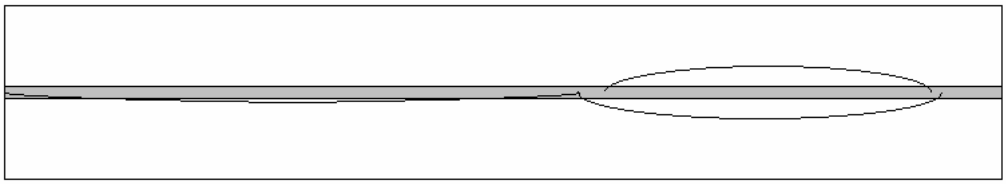


Figura 1-9 Vista gráfica de un archivo contaminado.

Se observa en la Figura 4-9, que existe una línea curva más prolongada que la mostrada en la Figura 4-8. Esta línea indica que el estado inicial del programa se ha movido hacía donde se encuentra el código del virus, y dentro del virus existen otros saltos, que en este momento no son de interés y que se explicarán con detalle más adelante (ver el epígrafe 4.4.3.2).

Se examinaron más de mil tipos de virus primarios en archivos con extensión (COM, EXE, NE, PE) y son parte de la plataforma Microsoft Windows. Algunos modifican el estado inicial de un programa, aunque existen excepciones como el virus Jerusalén, ya que este virus se coloca en la parte principal del archivo, en donde éste es el que ejecuta al programa original. Otros virus que se analizaron fueron Dir-II y AIDS. El primero, utiliza una inseguridad del Sistema Operativo y cambia las direcciones de la FAT (File Allocation Table) o tabla de localización de archivos. El segundo virus, sobrescribe al programa original, el cual destruye parte de la información existente en el huésped, para evitar con esto que no se tenga una propagación relevante según las estadísticas proporcionadas por la ICSA.

1.4.3.2 Crear una firma digital

Ahora se observará con detalle, cómo un virus se inserta en un programa, en donde asegura en un futuro su existencia y reproducción, para ello debe buscarse una manera de firmar el código y que, además, permita realizar operaciones básicas de conjuntos [62].

La probabilidad de 2^{-32} se obtiene de los checksum como son Adler o CRC 32 [63,64], que permite identificar, en el momento en que una secuencia de bytes ha sido modificada con una alta probabilidad.

Las firmas actuales son cadenas de 16 o 32 bytes que se utilizan para identificar un virus. Hoy en día, existen más de 70,000 virus, y cada virus, puede tener 20 firmas diferentes (esto depende de los antivirus utilizados), pero algo sí se tiene claro, el buscar la cantidad de firmas por cada programa que esté en una máquina, lleva más tiempo. Lo que dice la ICSA, en su encuesta del año 2001, es que para el año 2005 se llegarán a más de 100,000 virus aproximadamente, lo que lleva a un consumo importante en la memoria de la máquina para satisfacer sus requerimientos. Se sabe que se debe tomar otro camino el cual obligue a analizar con mayor precisión a los virus informáticos, ya que el modelo que se plantee deberá mantenerse por varios años.

Se pretende que las firmas no se obtengan de los virus en análisis, por lo contrario, se debe obtener del programa que se va a ejecutar. Esta firma debe buscarse en la base de datos de identificadores de los virus para emitir un resultado, lo que llevaría un consumo de tiempo menor, al que ya existe, claro está, en la segunda etapa [53].

Para explicar con mejor detalle los pasos que involucran este proceso se tiene el siguiente programa en forma de grafo.

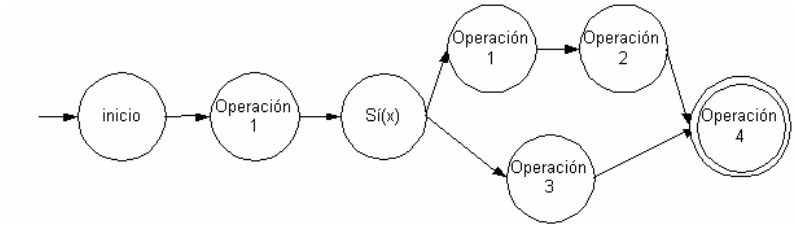


Figura 1-10 Representación de un programa.

En la Figura 4-10 se observa la estructura de un programa mediante [53] un grafo dirigido, donde indica un estado inicial "Inicio" y estado final "Operación 4", también se ve que existe una decisión "Si". Ahora bien bajo un análisis semántico se reduce el grafo, para lo cual se toman los nodos que no presentan bifurcación, lo que lleva a una representación mediante alguna firma (checksum) [58,59]. El objetivo es impedir que los virus se inserten en medio del grafo, lo que dará el siguiente resultado.

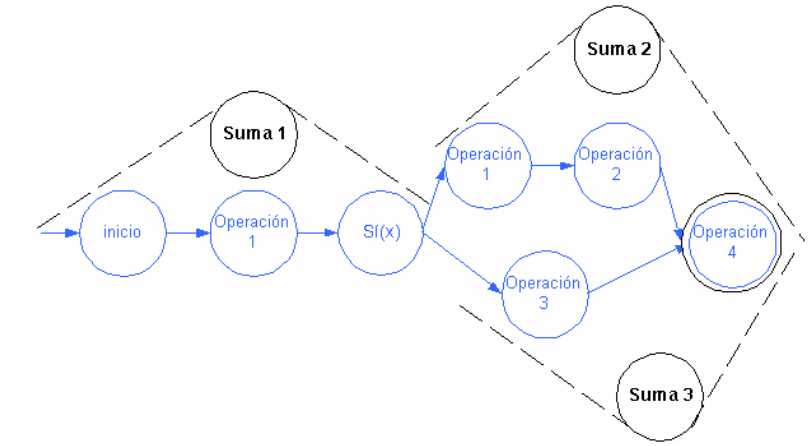


Figura 1-11 Extracción de la firma digital de un programa.

En la Figura 4-11, se observa que los tres valores "Suma 1, Suma 2 y Suma 3"son una representación del programa mostrado en la Figura 4-10. Si un virus se inserta en cualquier parte del programa se conoce el lugar aproximado de esta modificación, también ayuda a identificar a los virus como se ve el caso del programa infectado.

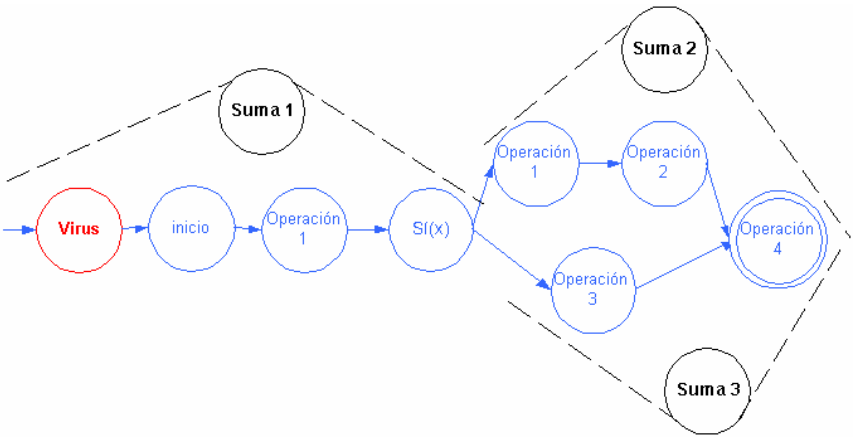


Figura 1-12 Esquema de un programa contaminado por un virus.

En la Figura 4-12, se observa que se ha insertado un proceso al principio del estado inicial. De esta manera, el virus se asegura de ser el primero en ser ejecutado (la mayoría de los virus se insertan al principio), lo que provocaría un cambio en la "Suma 1". Para resolver el problema se debe realizar un cálculo que permita eliminar al virus informático.

Una forma de estudiar este problema en un lenguaje ensamblador es el siguiente:

Se tiene un programa en el cual se extrae una firma:

```
1.  ; Programa Prueba                    ;Firma del programa prueba
2.  .MODEL SMALL
3.  .CODE

4.  ORG 100H
5.  INICIO:JMP MAIN                      5 JMP MAIN Checksum=876;
6.  MSG DB 'Archivo BAIT ',13,10,'$'
7.  PROC MAIN                            7 MAIN Checksum= 7438 ;
8.  ; . . . . . Programa
9.  INT 20H
10. MAIN ENDP
11. END INICIO
```

Código 1-1 Firma de un programa sin virus.

Si al programa mostrado en el Código 4-1 se le inserta un virus, el resultado será el siguiente.

1 .MODEL SMALL 2 .CODE 3 4 ORG 100H 5 INICIO:JMP Virus 6 MSG DB 'Archivo BAIT ',13,10,'$' 7 PROC MAIN 8 INT 20H 9 MAIN ENDP 10 11 PROC VIRUS 12 ; Código del virus para restablecer el JMP MAIN (Línea 5) 13 Mov Ax,100h 14 Push Ax 15 Ret 16 VIRUS ENDP 17 18 END INICIO	Firma: 5 JMP VIRUS Checksum=876 7 MAIN Checksum= 7438 ; 11 VIRUS Checksum=2344 15 Ret 100

Código 1-2 Archivo contaminado.

El Código 4-2 muestra cómo el virus es colocado al final de los segmentos de código y datos, pero al principio del programa. El virus modifica lugares esenciales, como es la línea 5. La finalidad es restablecer la línea 5 antes de alcanzar la línea 15, así, pasa desapercibido, en otras palabras, cambia la transición del estado inicial para insertarse.

Este tipo de interpretación abstracta toma más tiempo en ser generada comparada con las búsquedas que realizan los antivirus, para realizar la verificación en menor tiempo se utiliza una suma de validación. La idea es suponer que se tiene un programa ***a*** y un virus ***v*** que es interpretado por una máquina ***m*** y tres tiempos t_1, t_2 y t_3, además, si en el tiempo t_1 al programa ***a*** se le obtiene una huella o firma digital ***ct(a)=cta,*** por lo que en el tiempo t_2 el programa ***v,*** infecta al programa ***a***, esto originaría un programa nuevo llamado ***av***, y si en un t_3 se obtiene la firma digital de ***ct(av)*** y se compara con ***cta,*** se mostrara claramente que son diferentes. Las diferencias entre las dos firmas no permiten que en ningún momento se pueda verificar qué parte del código ha sido cambiada, esta modificación puede tener varios factores como son las actualizaciones o los virus, por ello se recurre a la interpretación abstracta como una segunda fase o etapa quien realiza un diagnóstico más preciso del cambio sufrido por el archivo ***av*** en el t_3.

Los linfocitos T, son capaces de identificar el antígeno de forma específica. Esto se implantó por medio de una validación de datos y de la firma llamada señal de peligro, de tal forma que cada programa que se ejecute en un equipo de cómputo sea verificado por si éste presentaba alguna modificación. Aunque se presenta un problema si los programas cambian

constantemente y evolucionan los resultados podrían ser incorrectos. Es por ello, que este modelo deberá ser validado a nivel Macro, en donde intervienen otras formas de propagación, que son a nivel jerárquico y espacial, y para ello se realiza otra analogía con el sistema inmune de los vertebrados llevándolo a este ámbito.

1.5 Un modelo del sistema inmune a nivel Macro

Un problema que presentan los antivirus comerciales es la creación de una firma que identifique a un virus en forma rápida, además que la creación de la vacuna se simplifique en gran medida.

¿Quién deberá iniciar la respuesta inmune en forma global? Todo programa que sea interpretado por una Máquina de Turing y que sufra una alteración (que es la modificación de la firma) deberá generar una señal de peligro. Esto es, enviar la firma original y la señal de peligro o la firma alterada a una central que se llamara Servidor T y que se le proporciona el nombre por la analogía con las células T del sistema inmune de los vertebrados. Para ello en el momento en que se ingresa un nuevo archivo a nuestro sistema éste deberá ser firmado.

El Servidor T, llamado así por las células T, será el encargado de recibir todas las firmas alteradas para compararlas entre sí y encontrar alguna similitud entre estas. La primera comparación que se hace es buscar segmentos que sean iguales, la probabilidad estará dada por el $checksum^n$ donde $n>1$ significa que es el número de segmentos encontrados en el primer programa. La siguiente comparación es la lógica de ejecución del programa, que está dada por el orden semántico en que se ejecuta. Por último, la comparación de una simple firma que se encuentra basada en la suma de todos los símbolos con un módulo. Esta última se debe a que los virus pueden encontrarse con algún reordenamiento de instrucciones. En el momento en que el Servidor T termina de realizar el diagnóstico de firmas, envía un mensaje llamado respuesta inmune a todos los equipos o nodos que se encuentran conectados a éste, con el fin de evitar que los virus se propaguen con mayor rapidez.

Otra pregunta que surge es ¿Qué tan rápido responde el modelo del sistema inmune ante un virus? Para generar la respuesta inmune específica se utiliza el concepto de tolerancia. Ésta se utiliza para distinguir la cantidad de firmas que son necesarias para emitir una firma de identificación que permita detectar en forma positiva a un virus o la llamada respuesta inmune.

Se supone que dos señales de peligro son suficientes para generar la respuesta inmune, pero para obtener una firma que permita una identificación con menos falsos negativos o falsos positivos se debe requerir de más señales de peligro. Para realizar el cálculo se utiliza la ecuación (4-1) que permite conocer la velocidad con que se propaga un virus y de esta manera estimar el momento en que hay que generar la respuesta específica.

$$T_j = \sum_{n=0}^{n=i} \frac{|t_{n+1} - t_n|}{n+1} \qquad (4\text{-}1)$$

donde $j = 1,2,3....m$ y t es la frecuencia con que llegan las firmas de peligro al Servidor T.

La ecuación anterior detecta el momento en que una pendiente en el tiempo T_j es mayor que una pendiente en el momento T_{j-1}, que proporciona una señal de inmunidad con los datos recolectados. Esta ecuación impide que se propaguen los virus en forma exponencial, tratando de evitar con esto un daño mayor a los sistemas de cómputo.

1.5.1 Funcionamiento real del servidor T

Primero se obtiene una firma de todos los programas que existen en la computadora. En el momento en que un programa es contaminado, se tiene una representación gráfica, como la que se ve en la Figura 4-7. En este momento, se envía una señal de peligro al Servidor T el cual contiene las dos firmas, él realiza una separación directa para eliminar áreas repetidas.

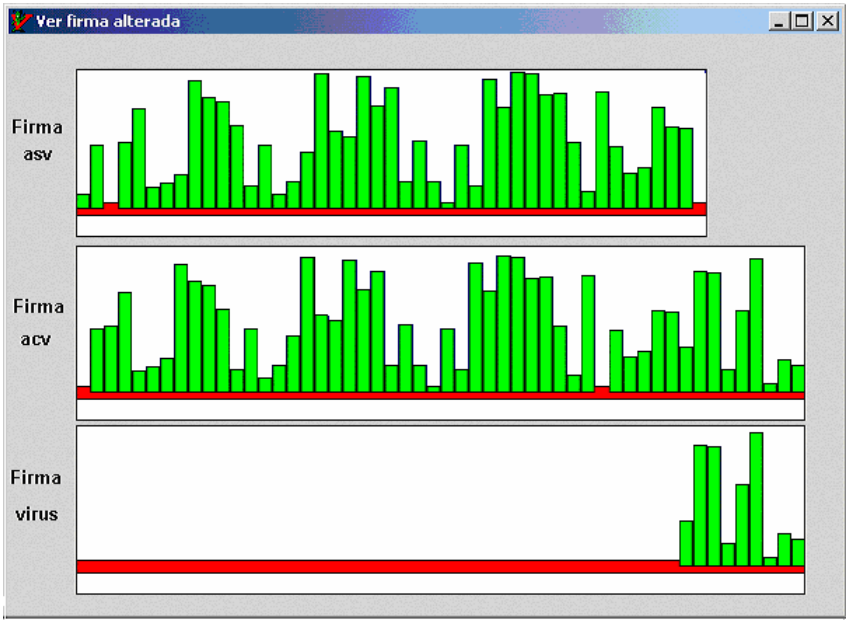


Figura 1-13 Localización de un virus informático.

Posteriormente se solicita ese segmento del virus, el Servidor B realizará un análisis más exhaustivo, que será el encargado de crear la vacuna específica.

Mientras el virus no sea un riesgo, se seguirán la recolección de las firmas que sean parecidas en un factor del 50%, lo cual garantiza que se trata del mismo virus. Así en el instante en que el virus empiece a generar una función exponencial creciente, se emitirá la llamada respuesta inmune que no es más que una firma que pueda identificar al virus. Esta

firma puede ser la suma de validación de un segmento, la estructura de ejecución o una cadena simple basada en patrones sencillos, como los estudiados en el Capítulo 2.

El Servidor B, es el encargado de recibir todos los segmentos de los códigos que el Servidor T diagnosticó como virus y así estimar una vacuna apropiada. Estas vacunas deberán ser enviadas a todos los nodos o equipos que tengan el virus y después enviarla al Servidor T, quien tendrá una respuesta inmediata para algún virus que sea repetido, no es recomendable enviar las vacunas a todos los equipos, ya que el costo de búsqueda irá en aumento. Éste es un problema que presentan los antivirus, ya que ellos guardan todas las firmas de identificación y las vacunas correspondientes.

En el momento en que un virus es identificado por el Servidor T, éste enviará los códigos de los virus al Servidor B, el cual se encargará de encontrar la vacuna correspondiente con los siguientes pasos:

a. Se emulan, los programas que tienen virus (ver Apéndice B) y se obtiene el área del código y el de datos. Posteriormente se buscan las partes que han cambiado en el área de datos dentro de las diferentes copias de los virus.

b. Se localizan las correspondencias entre los bytes del programa huésped y los del virus, esto se repite hasta que la suma de validación sea igual al original.

c. Se tienen las tablas de correspondencia entre los bytes que cambiaron en el archivo huésped, por lo que se genera la vacuna correspondiente.

d. De no existir la correspondencia antes mencionada, se procede a eliminar segmentos del virus, hasta restablecer el programa que corresponda con la firma original. Esto da la suposición de que se habla de un virus del tipo liso-génico y que su eliminación será la extracción de esa porción de segmento del archivo huésped, por lo que se buscan los patrones que marquen una limitante para realizar una identificación rápida.

Otro aspecto complementario en el modelo del sistema inmune basado en la señal de peligro, es la primera defensa que existe en el organismo (auto inmunidad). Esto indica que debe existir una ayuda para mantener aislado al virus mientras los Servidores T y B encuentran la identificación y la vacuna respectivamente. Una forma de realizar esto es utilizar algún método para la reconstrucción de estructuras, como el mencionado en el Capítulo 3.

Capítulo

5

1 APLICACIONES

Resumen

En este capítulo se presenta el diseño, funcionamiento e implantación de un sistema inmune en el ámbito local desarrollado en el lenguaje C++. Asimismo se presenta el diseño, funcionamiento e implantanción de un simulador de granularidad fina en el lenguaje Java y UML, en el cual se modelan las diferentes topologías de propagación y se analiza la señal de peligro con la respuesta inmune.

Objetivos del Capítulo

- Análisis, diseño e implantación de un sistema inmune en el ámbito local.
- Análisis, diseño e implantación de un simulador de granularidad fina.
- Definir los componentes básicos del modelo del sistema inmune en el ámbito micro y en el ámbito macro.
- Describir la implantación e integración del simulador basado en el sistema inmune y la señal de peligro.

1.1 Sistema inmune en el ámbito local

El programa llamado “sistema inmune local” debe tener la capacidad de localizar cualquier modificación dentro de los archivos ejecutables del sistema operativo Microsoft Windows. Asimismo el programa deberá estimar una cura inmediata para aquellos casos en que exista un virus, quedando restringida a los modelos de contaminación mencionados en el Capítulo 3.

El sistema inmune en el ámbito local deberá realizar un reconocimiento de los programas ejecutables del sistema operativo y extraer una firma, como la ya mencionada en el Capítulo 4, este análisis se realiza mediante una búsqueda recursiva de los directorios que presente el sistema operativo Microsoft.

Para el diseño del “sistema inmune local” se deben tener en cuenta algunos aspectos técnicos como son: La interpretación abstracta de un programa, la emulación, la reducción de instrucciones irrelevantes y la reconstrucción automática de programas de cómputo mediante las estructuras básicas de los archivos huéspedes, esto se puede ver en la Figura 5-1.

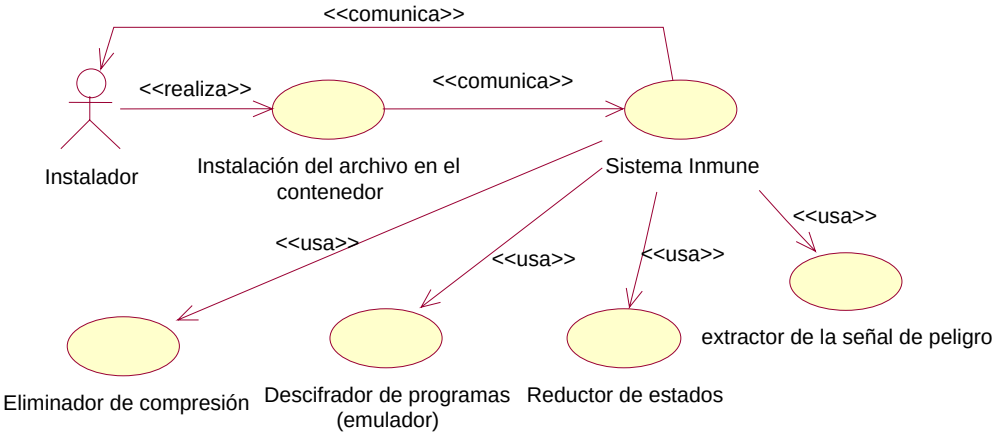


Figura 1-1 Casos de uso a nivel proceso del sistema inmune local

1.2 Interpretación abstracta de un programa de cómputo

Para crear un tipo de representación abstracta de un código y determinar si es un virus o no, primero se debe realizar una serie de pasos que garanticen que el código es lo más simple y representativo posible. Esto es debido a que los virus se cifran o se mutan para cambiar su apariencia y así generar falsos diagnósticos.

Como se ha observado en el Capítulo 3, existen diferentes virus. Las técnicas representan la evolución de un código, y el único objetivo de los creadores de virus es ocultar la apariencia de un programa, por lo que se deben utilizar varias técnicas para localizarlos. Una de las primeras técnicas utilizadas, fueron los llamados filtros, los cuales permiten representar el

código que se ejecuta en forma más simple; los antivirus actuales, utilizan a los emuladores, para lograr descifrar a los virus.

Este tipo de filtros conocido como emulador tiene el fin de descifrar cualquier virus. En el ejemplo del Apéndice B, se usa un emulador basado en el Microprocesador 8086/80386, actualmente existen varias empresas que venden estos filtros para cualquier tipo de máquina.

La creación del emulador se basa en el patrón de diseño *"Pequeño Lenguaje". (Little Language)* [65] de Grand Mark. Las modificaciones que se hicieron en el patrón, son para que las instrucciones tengan acceso al flujo de entrada, de esta manera se hace uso de la clase abstracta llamada *InstruccionAbstract, ésta* amplía la capacidad de instrucciones en el emulador.

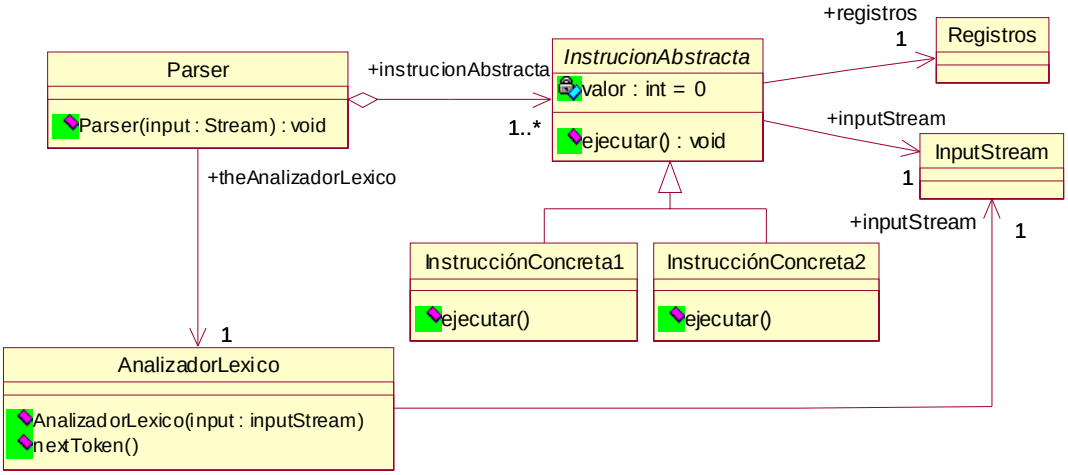


Figura 1-2 Patrón de diseño *Pequeño Lenguaje*

La clase flujo de entrada InputStram será el flujo donde quedará el código descifrado. En la clase Registro, se almacenan los registros de la máquina virtual (ax, bx, cx, etc); entre tanto si se utiliza otro lenguaje basado en otro tipo de máquina, se deberán analizar sus características estructurales y gramaticales.

Los virus utilizan un motor de mutabilidad, el cual presenta un tipo de evolución que impide que los antivirus los detecten. La solución a este problema consiste en crear un algoritmo que simplifique las *instrucciones irrelevantes* pudiéndose implementar con el patrón anterior. (Para analizar la solución a este tipo de virus véase el Apéndice B.)

La pregunta que surge aquí es: ¿Qué son instrucciones irrelevantes? Las instrucciones irrelevantes son:

- NOPS.
- Instrucciones que no cambian el control del flujo.
- Instrucciones que modifican los registros que no intervienen en el proceso.
- Secuencia de instrucciones que se optimiza bajo una reducción semántica.

Ejemplo:

```
Add ax,1
Sub ax,1
```

Una vez que todos los programas se encuentran firmados se espera a que existan dos condiciones: que el archivo se encuentre contaminado con un virus o que simplemente se trate de una actualización. Para encontrar la diferencia antes mencionada se presenta el siguiente algoritmo.

Proceso: obtener una **señal_de_peligro** (**archivo_actual**)
Si el el **archivo_actual** se encuentra comprimido *entonces*
Se procede de descompresión de **archivo_actual**
Programa_actual = extraer del **archivo_actual** el programa
Programa_descifrado = emular al **programa_actual** para descifrar o analizar alguna operación sospechosa.
Programa_reducido = eliminar las instrucciones irrelevantes **Programa_descifrado**
Señal_de_peligro = extraer una representación abstracta del **programa_reducido**

Proceso: comparar **verificar_virus(señal_de_peligro_actual, señal_de_peligro_anterior)**
Si **señal_de_peligro_actual** = **señal_de_peligro_anterior** *entonces*
Actualizar la **señal_de_peligro_anterior** en la base de datos con la señal de **peligro_actual**.
Fin del proceso
Por el contrario
Señal_de_virus=**señal_de_peligro_actual**–s**eñal_de_peligro_anterior**
Se envia al **Servidor_T** la **Señal_de_peligro_virus**

Proceso: **eliminar_posible_virus_1** (**archivo_actual**)
señal_de_peligro_anterior= obtener de la base de datos la firma del (**archivo_actual**)
señal_de_peligro_actual = obtener una **señal_de_peligro** (**archivo_actual**)
Si verificar_virus(**señal_de_peligro_actual, señal_de_peligro_anterior)** *entonces*
Si **reconstruir_archivo**(**archivo_actual**, **señal_de_peligro_anterior**) *entonces*
Activar la **señal_de_ peligro_virus** para prevenir infecciones futuras
Por el contrario
Activar la **señal_de_ peligro_virus** para alertar futuras infecciones
Se envia al **Servidor_T** la**(Señal_de_peligro_virus)**

Proceso: **buscar_virus**(**directorio**)
archivo_actual = **obtener_archivo**(**directorio**)
Si **contaminado**(**archivo_actual**) *entonces*

eliminar_posible_virus_2(**archivo_actual**)
Por el contrario
Si el archivo tiene señal_peligro(archivo) *entonces*
eliminar_posible_virus_1 (archivo_actual) *entonces*
Por lo contrario
Registrar en la base de datos la **señal_de_peligro**(**archivo**)

El proceso "**reconstruir_archivo"** del algoritmo antes mencionado se obtiene del experimento realizado con los reconstruccion de estructuras que se vió en el Capítulo 3.

El proceso "**contaminado**(**archivo_actual**)" del algoritmo antes mencionado se obtiene del experimento realizado con las búsquedas heuristicas que se vió en el Capítulo 3.

El proceso "**eliminar_posible_virus_2**(**archivo_actual**)" del algoritmo antes mencionado se obtiene por medio de la estructura de reconstrucción explicado en el Capítulo 4.

Todos los procesos descritos en esta parte de la presente tesis se encuentran en el disco adjunto, cabe recordar que es una herramienta experimental y que únicamente muestra las reconstrucciones de archivos que tienen la extesión EXE y COM. Esta herramienta se encuentra construida con el compilador C++ Builder 5 de la compañía Borland y utiliza una base de datos del tipo Paradox.

1.3 Simulador de granularidad fina

Para realizar experimentos en ambientes más amplios se propone realizar un simulador de granularidad fina, en el cual se lleva todo el proceso de infección en ambientes locales y bajo equipos de cómputo virtuales. Este simulador permite experimentar con modelos de propagación del tipo SIS, jerárquico y espacial explicados en el Capítulo 4.

Primero se analizará el concepto simulación, que es el proceso de diseñar un modelo [66] de un sistema real para llevar a cabo experiencias con el mismo, con la finalidad de comprender el comportamiento del sistema o de evaluar nuevas estrategias dentro de los límites impuestos por un criterio o conjunto de ellos, para el funcionamiento del sistema. En el contexto científico, la simulación será reproducir aparentemente una realidad compleja en un entorno controlado llamado sistema simulado [67]. Por un sistema simulado, se entiende que es un conjunto de entidades llamados componentes o elementos que están caracterizados por sus atributos y que se relacionan entre sí y con el entorno, además pueden evolucionar con el tiempo. El objetivo de la simulación es conseguir un entorno controlado para reproducir los virus informáticos, con sus relaciones internas entre elementos para estudiar las salidas del sistema en función de las entradas [68].

1.4 Un modelo

Un modelo es una abstracción de la realidad que captura la esencia funcional del sistema, con el detalle suficiente como para que pueda utilizarse en la investigación y la experimentación en lugar del sistema real, con menos riesgo, tiempo y costo.

Un modelo es un conjunto formado por otros dos conjuntos:

- Un conjunto de variables.
- Un conjunto de relaciones entre las variables u objetos del modelo.

Las variables del modelo representan habitualmente magnitudes físicas del sistema que se modela; las relaciones describen su comportamiento ante una cierta clase de situaciones.

1.5 El diseño de un simulador

El proceso de descripción del diseño del simulador, permite experimentar con las propiedades epidemiológicas frente al modelo del Sistema Inmune. Primero se describen los componentes que intervienen dentro del simulador y del modelo. Cabe recordar que el simulador creado analiza el comportamiento de los virus, así como el experimentar con aspectos que se han considerado un factor de propagación en los sistemas de cómputo [68,69].

1.6 Componentes que simulan una propagación Susceptible Infecta Susceptible (SIS)

El primer componente y el más importante llamado *Computador*, el cual posee el siguiente diseño y características [70]:

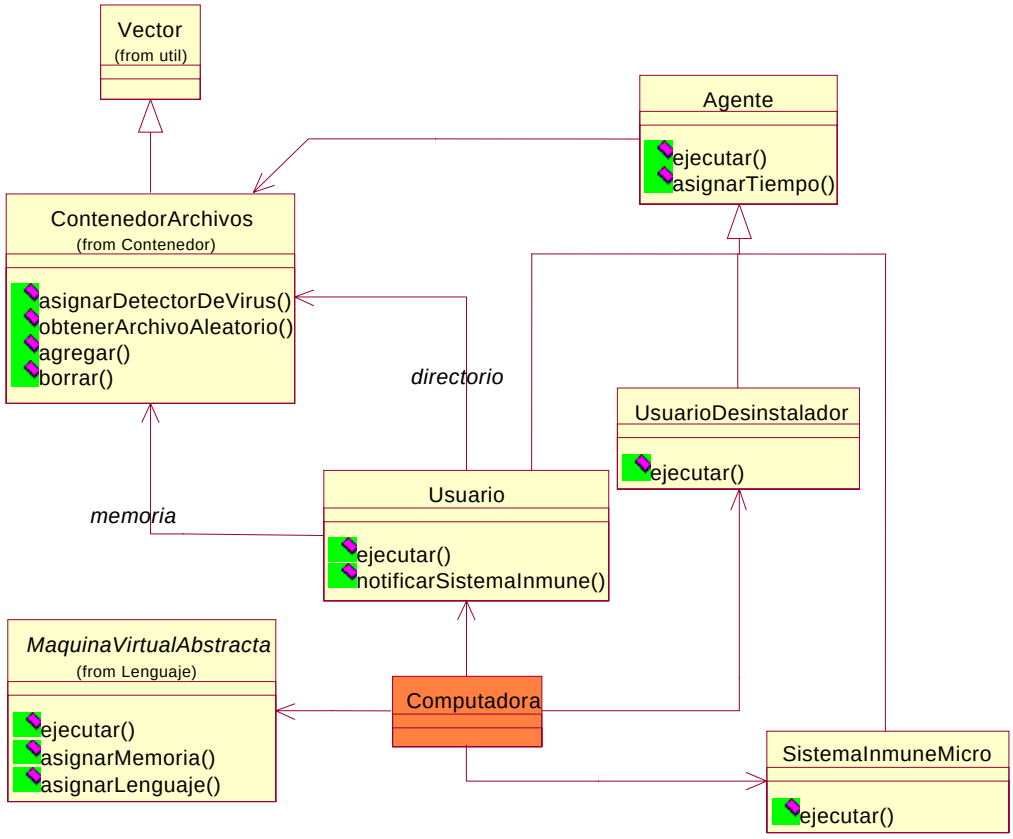


Figura 1-3 Diseño del Computador

Un *Computador* deberá tener memoria, directorio, sistema operativo, microprocesador (*MaquinaVirtualAbstracta*), así como algunos procesos que intervienen dentro del manejo o ejecución de un equipo de cómputo, que son:

Usuario: Es el encargado de tomar los programas que estan en forma estática para pasarlos a una máquina en la cual se interpreta en forma dinámica.

Usuario Desinstalador: Hilo que elimina los procesos estáticos.

Sistema Inmune Micro: Hilo que visualiza los cambios producidos en los programas en forma estática y que es el encargado de generar la señal de peligro (simulando al sistema inmune local).

En el componente llamado *Computador* se experimenta un modelo de pragación SIS y los resultados son muy semejantes a los propuestos por IBM.

1.7 Componentes que simulan una propagación en forma jerárquica

El componente que contiene un grupo de trabajo es llamado Grupo de Trabajo y se basa en el patrón de diseño llamado Compuesto, pero con la modificación de que los componentes deberán ser asociados en forma dinámica, como se observa en la Figura 5-4.

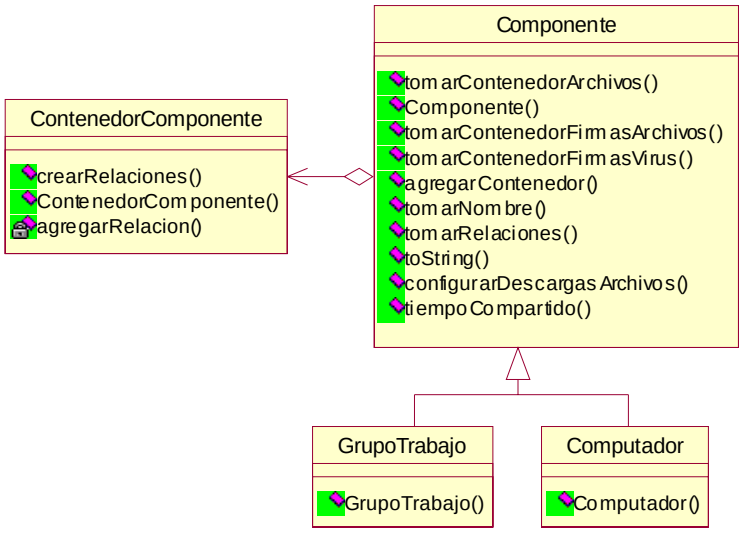


Figura 1-4 Diseño del componente grupo de trabajo

El patrón está basado en la filosofía de Swing [71], la cual utiliza componentes que son contenedores a diferencia del simulador que usa la composición en lugar de la herencia. En este caso todos los componentes tienen agregado un contenedor que le permite crear objetos más complejos, por el diseño de la interfaz gráfica. Este es el único componente que tiene a otros componentes (Computador), y esto se realizó con el objetivo de simplificar el diseño dentro del simulador. Así los usuarios solamente configurarán las características del grupo.

El *ContenedorComponente*, es el encargado de crear la relación de asociación entre los componentes en forma dinámica, para ello deberán poseer las dos características: el nombre y las relaciones de asociación entre los otros componentes de la clase base *Componente*.

Este componente permite realizar experimentos de propagación de un virus en forma jerárquica. Los resultados mostrados son semejantes a los propuestos por IBM.

1.8 Componentes que simulan una propagación del tipo espacial

Este componente hereda de un *Computador* y se le llamó *Internet*. Además el componente tiene dos caracteristicas: tiempo de retraso y nombre

Otros componentes que intervienen dentro del simulador son el programador de virus y el programador normal (creador de programas sin virus), los cuales tienen el objetivo de crear programas que sean interpretados por el *Computador*.

Para ser más eficiente al momento de crear programas y no consumir memoria innecesaria, se utilizó el patrón *Constructor* "Builder" [70], con el fin de dejar al constructor especializado la creación de los programas.

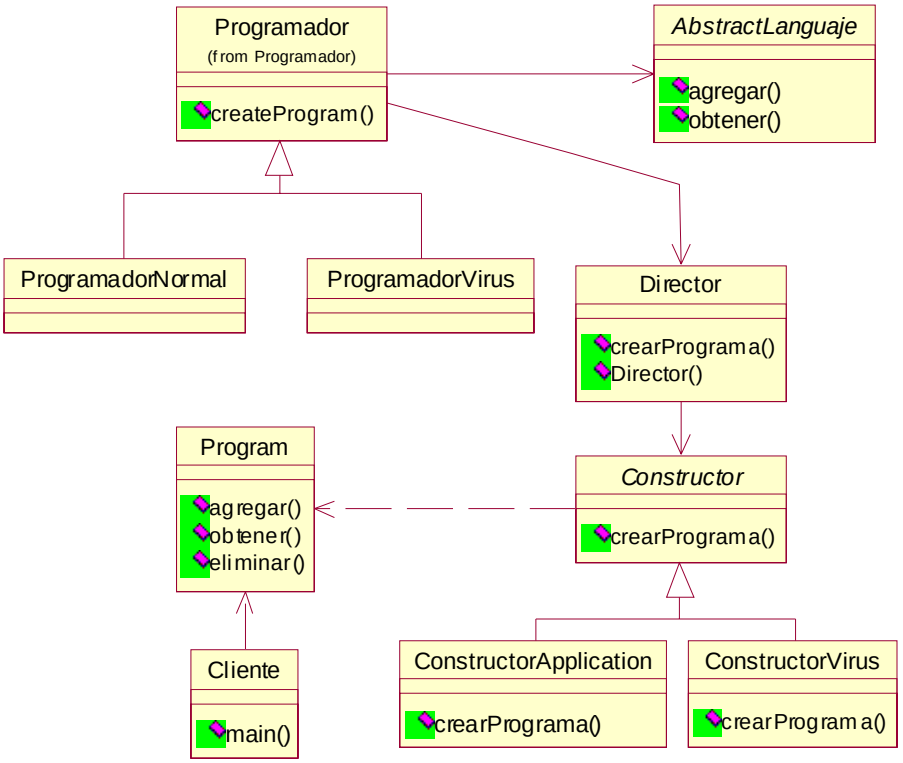


Figura 1-5 Descripción de la utilización del patrón Constructor

1.9 Componentes que simulan al sistema inmune

Otro de los componentes que se ha creado es el llamado *Servidor T*, su objetivo principal es recibir todas las firmas o mensajes de alteración que los programas han sufrido. El diseño de este componente se puede ver en la Figura 5-1:

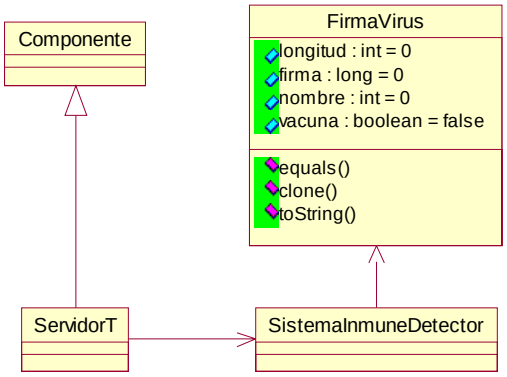


Figura 1-6. Diseño del componente llamado Servidor T

Otro de los componente que intervienen es conocido como *Servidor B*, que es el encargado de crear la vacuna específica para erradicar del sistema a un virus específico. El diseño de este componente se puede ver en la Figura 5-1:

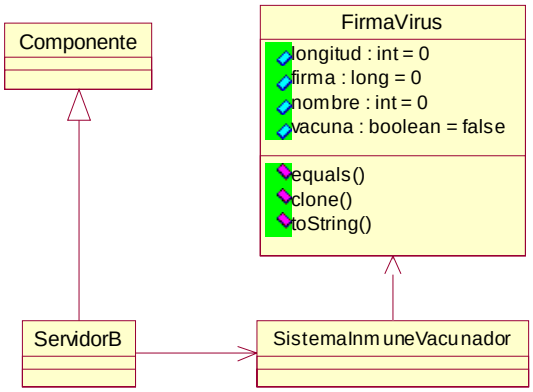


Figura 1-7 Diseño del componente llamado Servidor B

La clase *SistemaInmuneVacunador* es la encargada de obtener la cura apropiada del virus. Este proceso toma tiempos diferentes y esto es debido a la complejidad del virus.

Si se requiere analizar más patrones de diseño del simulador se puede consultar el Apéndice A.

1.10 El diseño de la interfaz gráfica del simulador

El simulador presenta un diseño gráfico que permite modelar la mayoría de los componentes que existen hoy en día, y que intervienen en forma directa en la propagación de los virus informáticos. Para ello se posee una barra de herramientas como se muestra en la Figura 5-8.

Figura 1-8 Barra de herramientas del simulador.

Dentro de la barra de herramientas que se muestra en la Figura 5-8 se observan unos componentes. Cada uno de estos componentes tiene una relación con los medios de propagación que existen hoy en día. A continuación se describe la funcionalidad de cada uno.

1.10.1 Grupo de trabajo

	El componente llamado “Grupo de Trabajo” es un tipo de Servidor que tiene varios subcomponentes designados “Computadores”. Desde el punto de vista del modelo, éste se configuran en forma dinámica dentro del mismo.

La forma en que se presentan los datos se puede observar en la Figura 5-9:

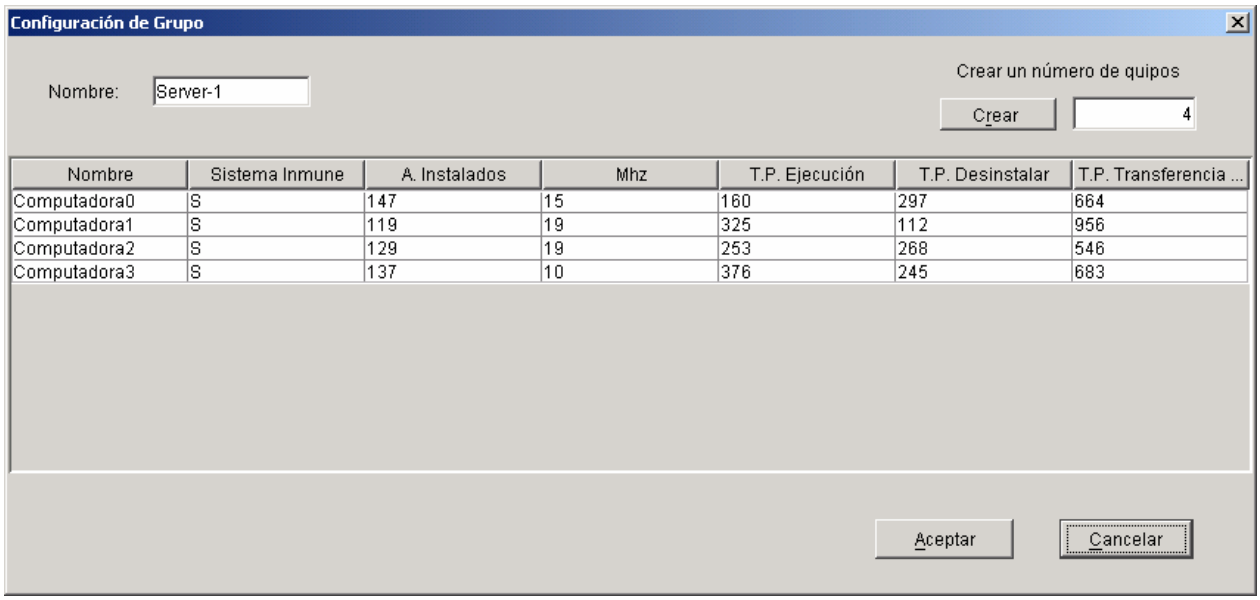


Figura 1-9 Forma para configurar los equipos de cómputo.

Nombre: El Nombre del equipo que se encuentra instalado en la red.

Sistema Inmune: Indica si el modelo del sistema inmune local está instalado en el equipo, lo que permite enviar la señal de peligro para predecir en qué momento se proporciona la propagación del virus.

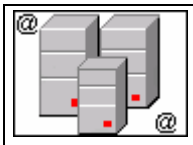
Archivos Instalados: Es el número de los archivos “sanos”, por así llamarlos, instalados al inicio de la simulación.

Mhz (Instrucciones): La velocidad de cada procesamiento.

Tiempo promedio de ejecución: Es el tiempo promedio en que el "*Objeto Usuario*" toma un archivo del directorio y es colocado dentro de la memoria, para ser ejecutado.

Tiempo promedio para desinstalar: Es el tiempo promedio en que se eliminan los archivos del Sistema de Archivos (sólamente si no se encuentra en uso).

1.10.2 Servidor de Internet

	Servidor de Internet: Este componente permite enviar información a otro componente del mismo tipo, sin tener que crear una relación visual, ya que su objetivo es mover la información [72] de un lugar a otro en un tiempo determinado.

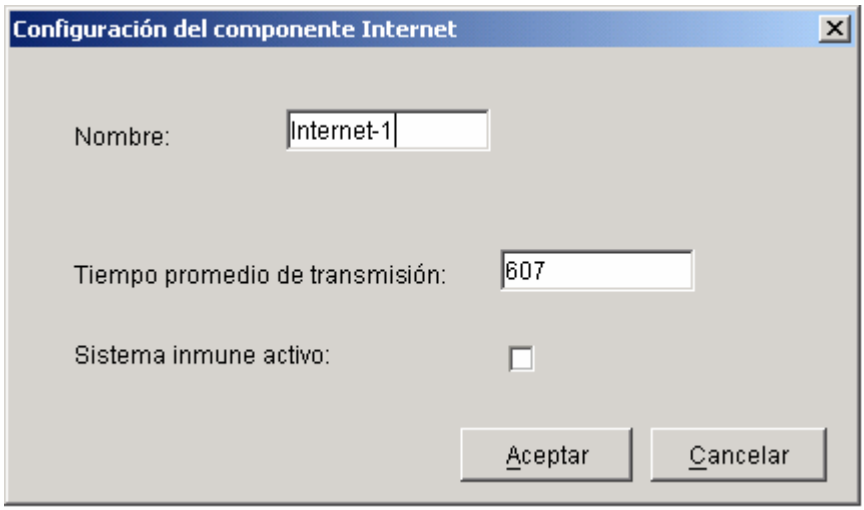
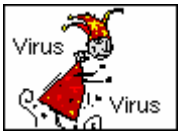


Figura 1-10 Configuración del componente Internet.

Nombre: Nombre del componente

Tiempo promedio de transmisión: Es el tiempo en que la información es leída en un lugar y escrita en otro componente "Internet".

1.10.3 Programador de virus

	Programador de virus: El objetivo de este componente es el dar la facilidad de colocar en un lugar específico la creación de virus informáticos.

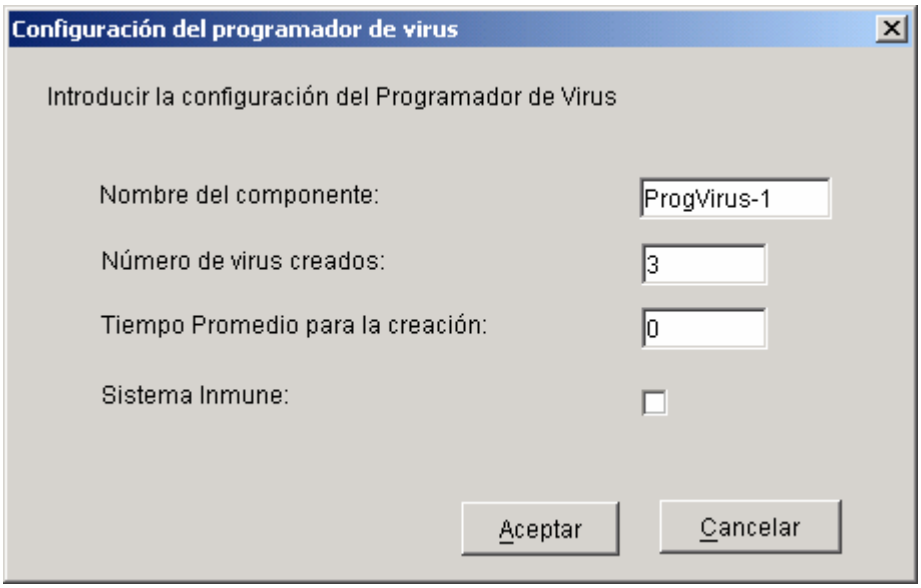


Figura 1-11 Configuración del componente programador de virus.

Las características de configuración que se presentan en la Figura 5-11 del componente son:

Nombre del componente: Es el nombre con el que se identifica el componente y permite obtener los resultados específicos.

Número de virus creados: Es el número de los virus que se crean durante la simulación.

Tiempo promedio para la creación: El tiempo promedio que toma la creación entre los virus.

Sistema Inmune: Es el indicador donde el componente posee un recolector de representaciones abstractas.

1.10.4 Programador normal

	Programador Normal: Este componente tiene dos funciones básicas, el de crear y actualizar programas así como el de colocarlos dentro de un archivo.

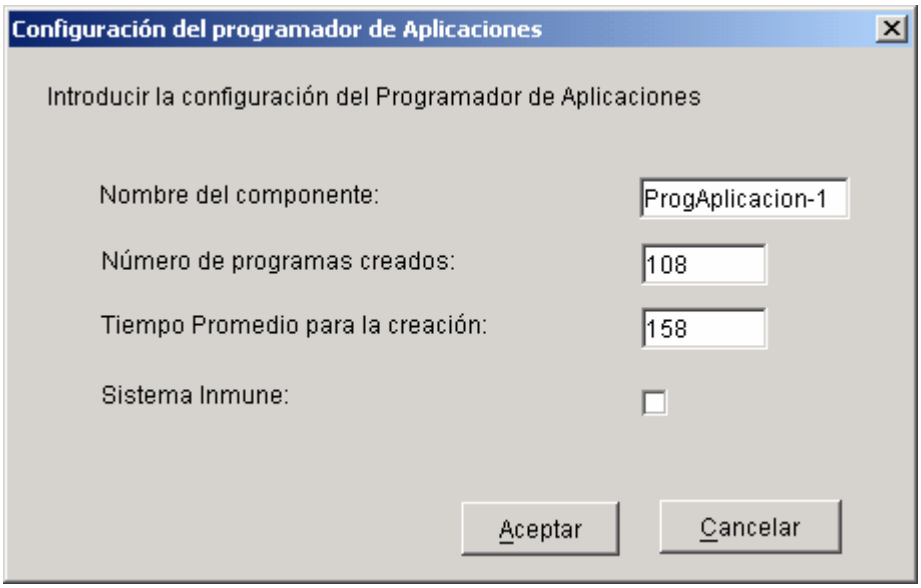


Figura 1-12 Configuración del componente programador de aplicaciones.

Las características de configuración del componente son:

Nombre del componente: Es el nombre con que se le conoce al componente durante la simulación.

Número de programas creados: Es el número de programas creados que manejará el componente.

Tiempo promedio de creación: Es el tiempo en que se actualizan o crean nuevos programas.

Sistema Inmune: Es el indicador donde el componente posee un recolector de representaciones abstractas.

1.10.5 Componentes de la respuesta inmune

Los siguientes componentes son los encargados de la respuesta inmune y de la relación que existe entre la información.

	Servidor T: Este componente es el encargado de recolectar las alteraciones de todos los programas que tengan el sistema inmune a nivel micro, así como realizar el diagnóstico para predecir en dónde se realiza una propagación de virus.

	Servidor B: Este componente es el encargado de darle prioridad a la creación de la vacuna específica contra el virus informático.

	Relación: Este componente permite asociar todos los objetos anteriores. La relación que se origina es del tipo agregación, y el objetivo es permitir obtener información de otro objeto.

1.11 Funcionamiento básico del simulador

Ahora bien, ya que se tienen definidos todos los objetos que intervienen dentro del simulador, una forma de estudiar el cómo interactuar con los objetos, es realizar un estudio basado en un ejemplo que permita modelar algún sistema, y predecir la propagación en caso de que un virus se infiltre dentro del sistema.

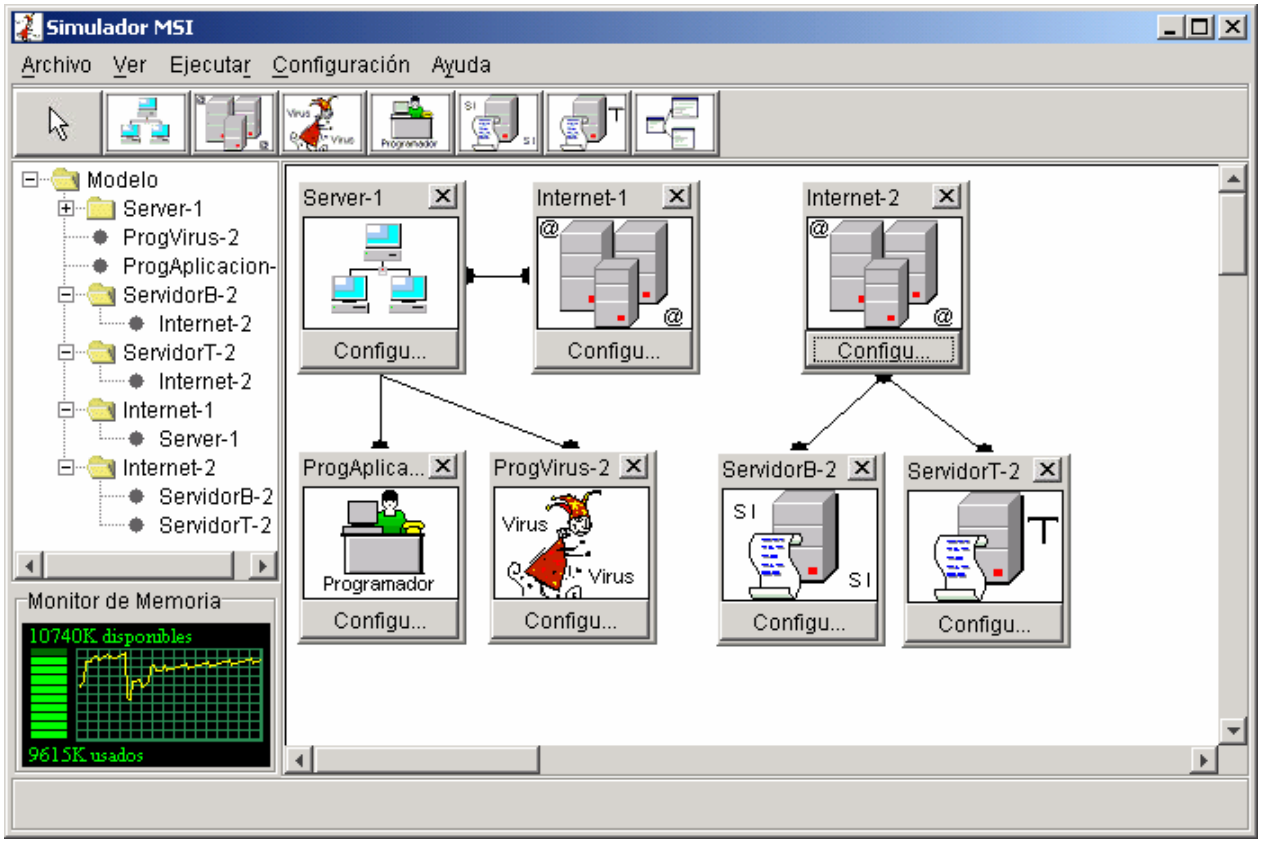


Figura 1-13 Un simulador visual para diagnosticar la propagación de los virus

En la Figura 5-13 se observa un ejemplo en el que intervienen varios componentes. Primero se ve a *Server-1* que está compuesto por un *Grupo de Trabajo* el cual posee cinco objetos conectados entre sí. El componente copia archivos e información del componente marcado con el nombre de *ProgAplicacion* y *ProgVirus-2*. Otro componente que interviene es *Internet-1* a diferencia del *Server-1*, éste puede copiar archivos y enviar archivos al componente llamado *Internet-1*, se observa la relación de navegabilidad doble.

El componente *Server-1* está relacionado en forma automática con el *Server-2*, por lo que no hay que establecer ninguna relación visual. El componente llamado *Server-2* permite la interacción con los componentes llamados *ServidorB-2* y *ServidorT-2* que son los encargados de la respuesta inmune y la vacuna respectivamente.

1.12 Las gráficas de las curvas de propagación

En la Figura 5-14 se observa la gráfica del componente *Computadora* que se encuentra contenida dentro del *Server-1*, se observan algunos resultados interesantes, en donde los virus informáticos se extinguen inmediatamente antes de llegar a causar daño mayor al sistema de cómputo.

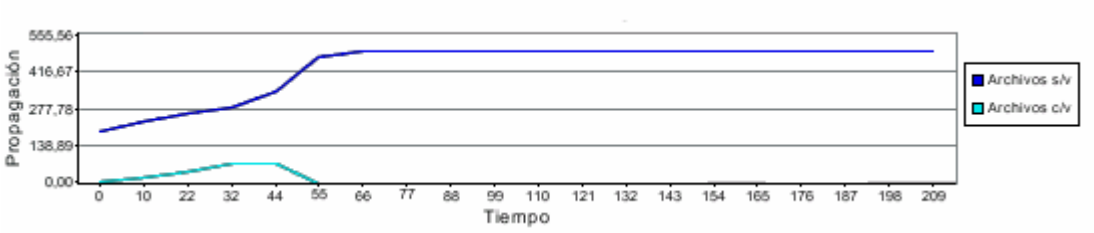


Figura 1-14 Curva de propagación de un virus en el simulador.

Otra gráfica que se observa es la siguiente (Figura 5-15):

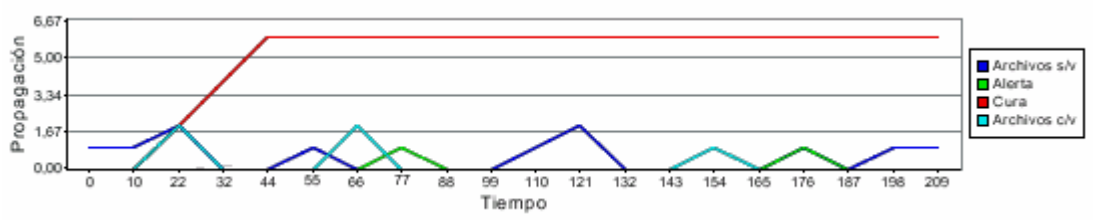


Figura 1-15 Curvas de propagación de los virus y su cura en el simulador.

El componente *Internet-1* de la Figura 5-15 tiene más mensajes de cura que mensajes de archivos contaminados. Además se observa que los mensajes de alerta que se producen se originan después de la creación de un virus.

Otra gráfica que se observa está dada en la Figura 5-16, la cual genera la respuesta inmune inmediata, que es un indicador para la presencia del virus en análisis. El componente que proporciona esta información es llamado *Internet*:

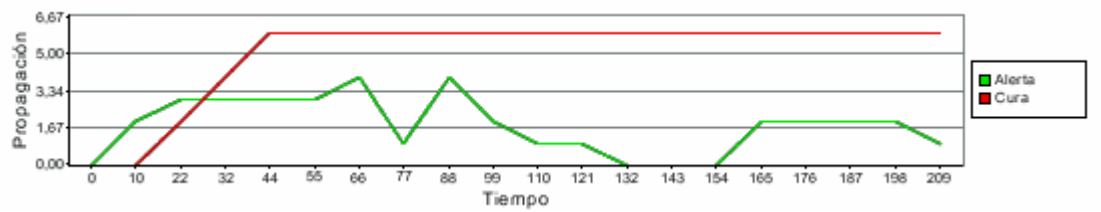


Figura 1-16 Curva de respuesta inmune en el simulador.

En la gráfica de la Figura 5-16 se observa que las alertas se dan durante toda la simulación y esto es debido a que los programas se encuentran en evolución. En el tiempo 44 se observa que el total de curas ha sido alcanzada, en ese instante muestra que existen 6 tipos de virus diferentes; aunque en el ejemplo se utilizaron únicamente 3 virus, al producirse la sinergia el sistema inmune encontró 6 virus diferentes.

Capítulo 6

1 Resultados y conclusiones

1.1 Resultados del sistema inmune local

En las pruebas realizadas, se tomaron como base a trescientos virus primarios, se le llama así porque ellos presentan características novedosas y más de mil virus que son modificaciones de los anteriores, los resultados que se obtuvieron se muestran en la Tabla 6-1.

Para realizar las pruebas, primero se tomó un archivo de carnada llamado BAIT con el fin de examinar el comportamiento en la forma más sencilla. Posteriormente se infecta con diferentes tipos de virus nuevos, para luego reconstruir de forma automática el archivo original basándose en su representación abstracta.

Tipo de virus	Tamaño asv	Tiempo análisis	Tamaño acv	Tiempo diagnóstico	Tamaño Avc	Tiempo reconstrucción
Agregado	7,023	1seg	7,884	0.5 seg	7,023	2 seg
Sobrescribe	10,478	1seg	10,478	0.5 seg	X	X
Oculto	10,471	1.2seg	11,724	0.5 seg	10,471	2seg
Polimórfico	20,183	1.5seg	20,549	0.7 seg	20,183	3 seg
Metamórficos	19,669	1.7seg	25,546	0.7 seg	19,669	5 seg

Tabla 1-1 La comparación de la reconstrucción de los diferentes virus

donde:

- Asv = archivo sin virus.
- Acv = archivo con virus.

Las pruebas mostradas en la Tabla 6-1, se realizaron con un equipo de cómputo que tiene las siguientes características: un microprocesador Intel a 1.6GH con un disco duro de 49MB a 7200RPM. El tiempo de diagnóstico es relativamente bajo ya que solamente se trata de verificar una firma basada en una suma de validez (checksum). El algoritmo utilizado en esta prueba fue el CRC-32. Si existe alguna diferencia entre el archivo original y el actual, se procede a la reconstrucción y validación del proceso.

Durante las pruebas que se hicieron con los diferentes archivos infectados, se localizaron los siguientes problemas al utilizar la representación abstracta, por lo que se realizaron ciertos ajustes:

Se identificó el principio del proceso: Esto varía con cada formato de archivo ejecutable, por lo que se debe tener cuidado al manejar diferentes formatos.

Se identificó una parte final del archivo: Esto es de forma directa y se realiza al extraer una cadena o suma de validación al final del archivo.

Se almacenó la cabecera del archivo: Cada archivo que se ejecuta posee una estructura diferente que indica al sistema operativo cómo cargar a éste en memoria, así como la indicación de algunas variables que son de importancia, por ejemplo un archivo EXE de la compañía Microsoft utiliza 24 bytes (ver el Capítulo 3).

Otro resultado de interés, es la cantidad de nodos que se genera por cada firma.

Nodos	Tamaño	Nombre del archivo
8	369	C:\peligro\virusdes\virmax2.com
61	5,447	C:\peligro\virus2\virusa.com
63	5,813	C:\peligro\virusdes\choice.com
45	7,023	C:\peligro\virus1\sinvirus1.com
55	10,471	C:\peligro\virus3\virusa.com
56	10,885	C:\peligro\virusdes\more.com
132	16,389	C:\peligro\virusdes\doskey.com
179	19,669	C:\peligro\virus5\sys.com
179	19,669	C:\peligro\virusdes\sys.com
200	20,183	C:\peligro\virus4\keyb.com
201	20,549	C:\peligro\virusdes\keyb.com
179	22,613	C:\peligro\virusdes\diskcopy.com
221	30,261	C:\peligro\virusdes\mode.com
373	51,077	C:\peligro\virusdes\format.com

Tabla 1-2 Cantidad de nodos que genera cada archivo

Como se observa en la Tabla 6-2, la cantidad de nodos en la firma no depende directamente del tamaño sino de la complejidad del programa. En las pruebas realizadas, se eliminaron algunas instrucciones que generan más nodos que no son necesarios, estos son los saltos cortos; ya que dentro del lenguaje ensamblador se conocen instrucciones que generan bifurcaciones o saltos que son inmediatos. Algunas de las instrucciones no pueden llegar a más de 64K (por ejemplo je, jz) por lo que este tipo de instrucciones, se toman como simples flujos en el análisis.

La cantidad de nodos que genera la firma al momento de encontrar una alteración, dependerá del virus que encuentre.

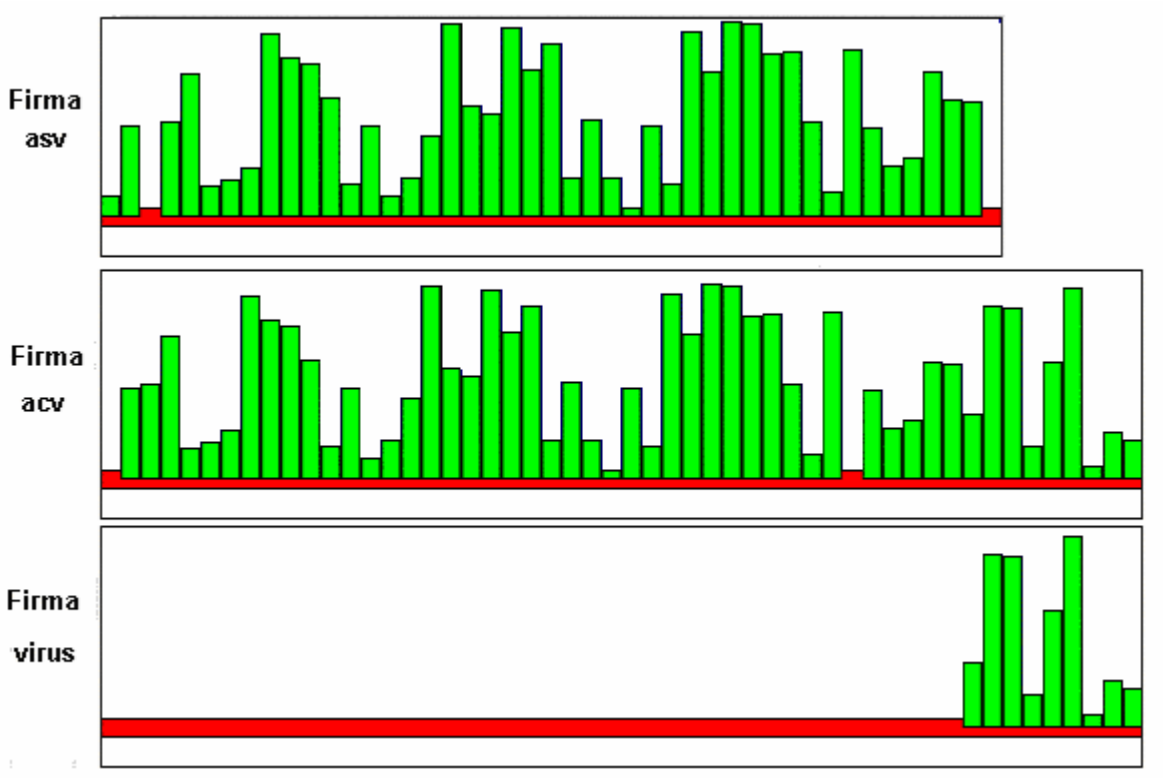


Figura 1-1 Comparación de los nodos en un archivo infectado

Por lo regular depende del virus, en el caso de la Figura 6-1 se observa que el virus se encuentra al final del archivo, pero al principio del programa. Observe que al principio de la firma los 3 primeros nodos son cambiados. Esto es debido a que los virus se han insertado dentro del programa al principio pero su código está al final del archivo, por lo explicado en el Capítulo 3.

En las pruebas se experimentó con un virus de 344 bytes, el cual produjo una firma, como la que se ve en la Figura 6-2.

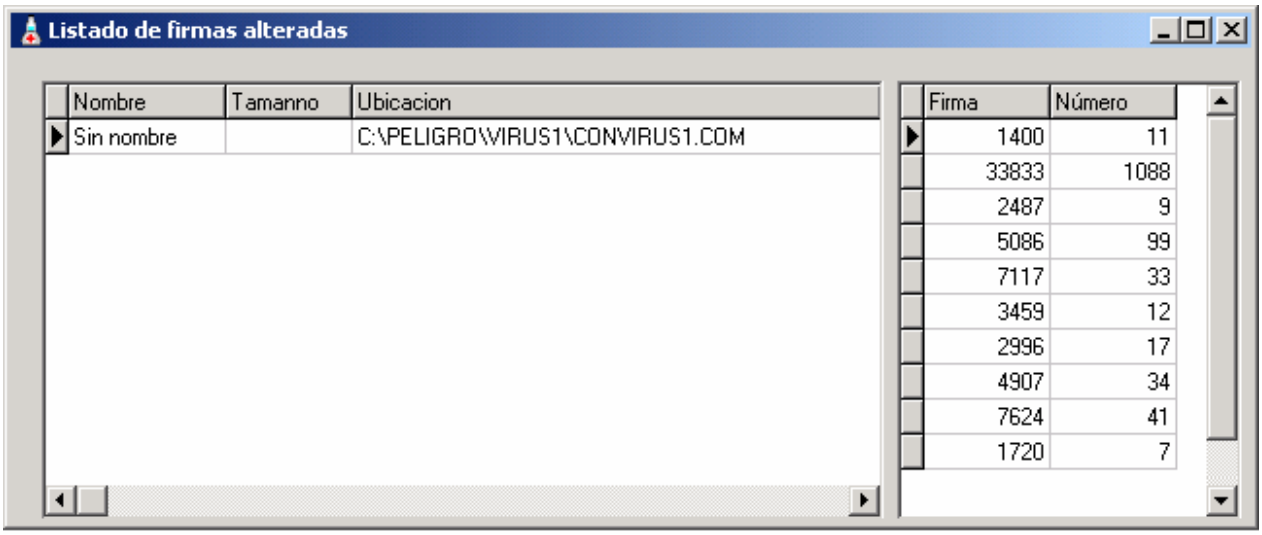


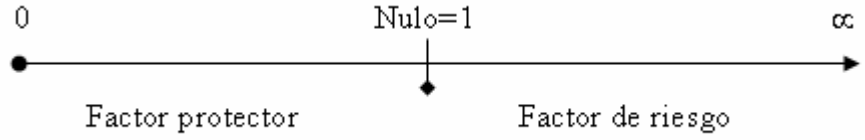
Figura 1-2 Lista de una firma.

Esta es la firma que se envía a un servidor llamado *Servidor T*, en donde se centralizan todas las alteraciones existentes de todos los equipos que se encuentran conectados a este servicio. En el *Servidor T*, se analizan todas las alteraciones similares para emitir una firma de identificación (ver Capítulo 5). En las pruebas realizadas no se produjo ningún falso positivo o falso negativo. La prueba se realizó con más de 10,000 archivos ejecutables con una efectividad de un 98% en la detección. No obstante, falta por realizar pruebas en un ambiente más amplio.

1.2 Los resultados del simulador del sistema inmune

Primero se realizaron algunas pruebas para tomar medidas que cuantifiquen la discrepancia en la ocurrencia de infección en algunos modelos que definen la presencia o no de ciertos virus informáticos. Estas medidas pueden calcularse tanto para dos eventos de dos objetos, como para un solo evento en dos objetos. Para cálculos de riesgo, el resultado es de la siguiente manera:

R = 1 Indica ausencia de asociación, no-asociación o valor nulo.
R < 1 Indica asociación negativa, factor protector.
R > 1 Indica asociación positiva, factor riesgo.



La interpretación de estas medidas se basa en el hecho de que si se dividen dos cantidades entre sí y el resultado es 1, éstas son necesariamente iguales ya que pueden tener o no las características estudiadas. Por lo contrario, en el momento en que la razón es mayor que 1, el factor se encuentra asociado positivamente con el riesgo de enfermar (ver Capítulo 4) y la

probabilidad de contraer el padecimiento será mayor entre los expuestos. Si el resultado es menor que 1, el factor protege a los sujetos expuestos contra esa enfermedad.

Para las pruebas dentro del simulador, primero se toman dos equipos que tuvieran las mismas características, pero la *Computadora-0* presentará un sistema inmunológico al nivel micro y la *Computadora-1* no presentará ninguna resistencia contra un virus nuevo.

Nombre	Sistema Inmune	A. Instalados	Mhz	T.P. Ejecución	T.P. Desinstalar	T.P. Transferencia ...
Computadora0	S	117	5	200	250	900
Computadora1	N	117	5	200	250	900

Figura 1-3 Configuración del grupo de trabajo

Y se tiene el siguiente modelo:

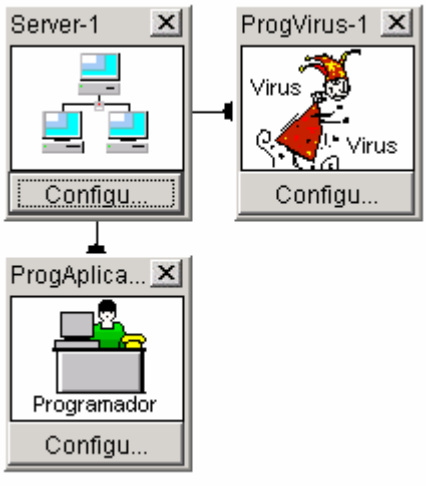


Figura 1-4 Un modelo visual de los equipos con y sin sistema inmune en el nivel micro

Los resultados obtenidos se pueden observar en la Figura 6-5 y la Figura 6-6:

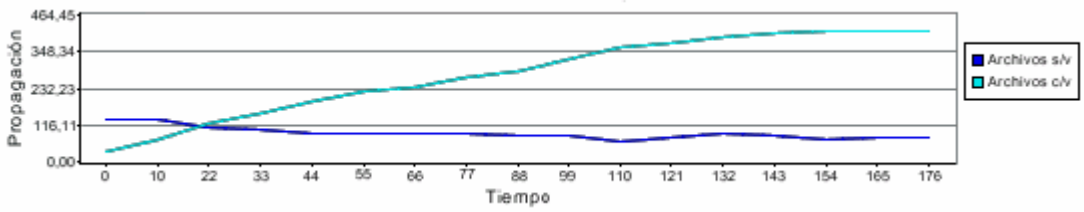


Figura 1-5 Gráfica de propagación del sistema inmune en el nivel micro activado

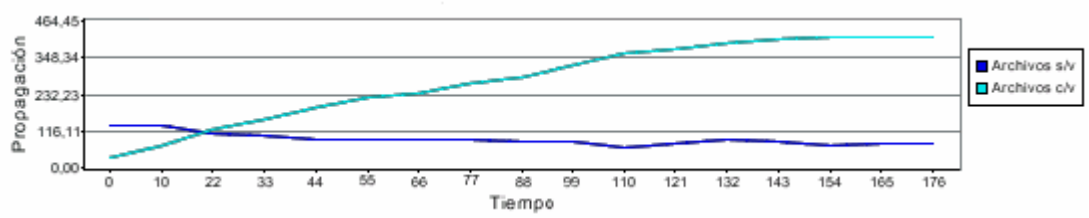


Figura 1-6 Gráfica de propagación con el sistema inmune en el nivel micro desactivado

Primeras conclusiones, como se observa en la Figura 6-5 y Figura 6-6 existen diferencias importantes. Primeramente se observa la cantidad de archivos con los virus en la *Computadora-0*, lo que hace que tarde en sobrepasar la cantidad de archivos sin virus con respecto a la *Computadora-1*. La *Computadora-0* presenta una resistencia a la enfermedad, mientras que en la *Computadora-1* la propagación es más rápida.

Para estimar esto de una manera no visual se realiza un cálculo de riesgo (observar la Tabla 6-3), basándose en los dos censos que se realizaron:

Computadora 0			Computadora 1		
Infectados	Muestra	Riesgo	Infectados	Muestra	Riesgo
37	178	0,2079	42	186	0,2258
66	207	0,3188	76	215	0,3535
87	227	0,3833	129	248	0,5202
113	237	0,4768	159	266	0,5977
138	250	0,552	198	297	0,6667
169	278	0,6079	227	323	0,7028
205	305	0,6721	242	338	0,716
236	334	0,7066	272	370	0,7351
250	350	0,7143	293	382	0,767
283	387	0,7313	331	422	0,7844

Tabla 1-3 Cálculo del riesgo en equipos de cómputo

Infectados = archivos que tienen virus.

Muestra = total de archivos de muestra.

Csi = con sistema inmune

Ssi = sin sistema inmune

Existen diferentes procedimientos para cuantificar la importancia de la asociación. Uno de ellos utiliza el cociente de los dos riesgos, que indica cuanto más probable es el primero frente al suceso del segundo grupo. Es lo que se conoce como **Riesgo Relativo** (**RR**).

Como se ve, el contar con un sistema inmunológico dentro de cada equipo es un factor protector contra un virus que no ha sido diagnosticado.

No obstante, la acción de proteger no impide que se propaguen los virus, por eso se creó una firma que sea fácil de manejar y que permita operaciones rápidas al momento de realizar el diagnóstico.

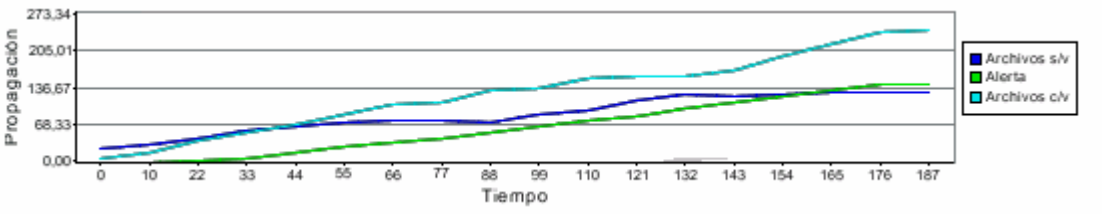


Figura 1-7 Gráfica de propagación del mensaje de alerta

La señal de peligro que es una firma, se envía a un *Servidor T*, donde se predice si la evolución que existe se trata de un virus o simplemente de una actualización. En la Figura 6-7 se observa como la señal de alerta acompaña en todo momento a la curva de propagación del virus, en este caso no prevalece la disminución, debido a que en el modelo mostrado en la Figura 6-4 no existe el componente llamado *ServidorT* quien es el encargado de realizar un diagnóstico final.

Para estudiar este comportamiento se ha cambiado el modelo visto en la Figura 6-4 de la siguiente manera:

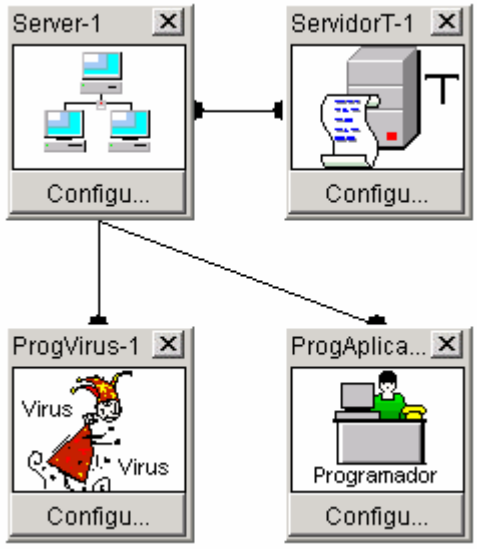


Figura 1-8 Un modelo visual de los equipos con SI y sin SI

Los resultados obtenidos se muestran en las siguientes gráficas:

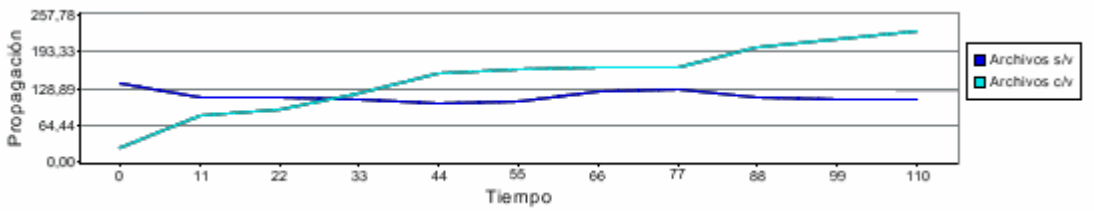


Figura 1-9 Gráfica de propagación con el SI en el nivel micro y macro activado

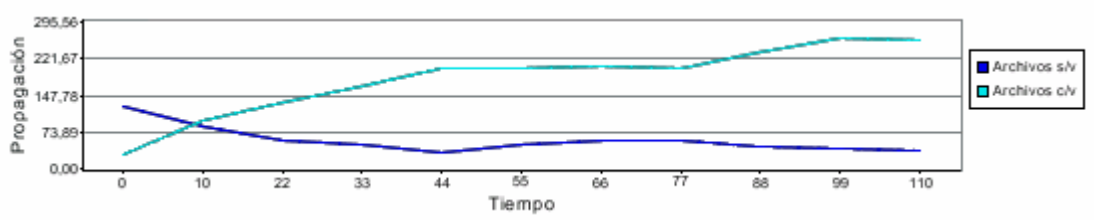


Figura 1-10 Gráfica de propagación sin nivel micro y con el nivel macro activado

En la Figura 6-9 y Figura 6-10 el riesgo de contraer un virus ha disminuido pero no ha desaparecido, ya que lo único que se agregó fue un *ServerT*, con la finalidad de diagnosticar en qué momento se produce un nuevo virus y alertar a otros equipos que se encuentren conectados.

Computadora 0			Computadora 1		
Infectados	Muestra	Riesgo	Infectados	Muestra	Riesgo
28	170	0,1647	33	164	0,2012
86	203	0,4236	101	191	0,5288
96	212	0,4528	136	196	0,6939
123	236	0,5212	170	222	0,7658
158	264	0,5985	206	244	0,8443
166	275	0,6036	207	260	0,7962
168	294	0,5714	210	273	0,7692
170	301	0,5648	205	268	0,7649
203	318	0,6384	237	288	0,8229
218	332	0,6566	266	312	0,8526

Tabla 1-4 El cálculo de riesgos en equipos de cómputo con sistema inmunológico

Como se observa en la Tabla 6-4, el riesgo de contraer un virus ha disminuido pero no ha desaparecido, por lo que se agrega un nuevo componente llamado *Serveridor B* que es el encargado de encontrar la vacuna.

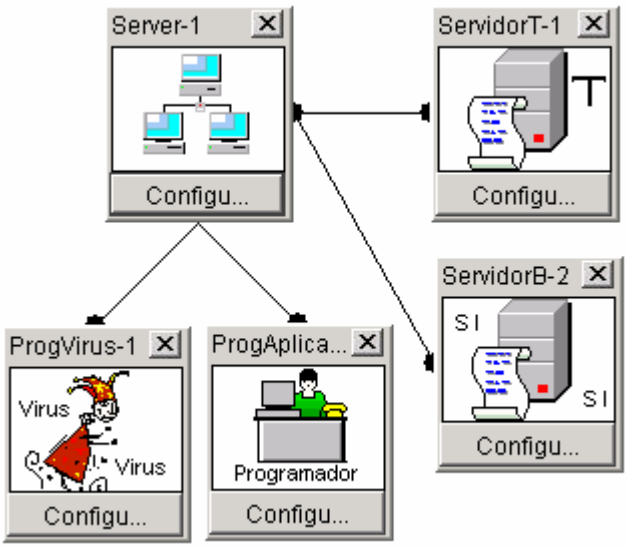


Tabla 1-5 Modelo visual de equipos con el sistema inmune completo

Los resultados obtenidos en las gráficas son las siguientes:

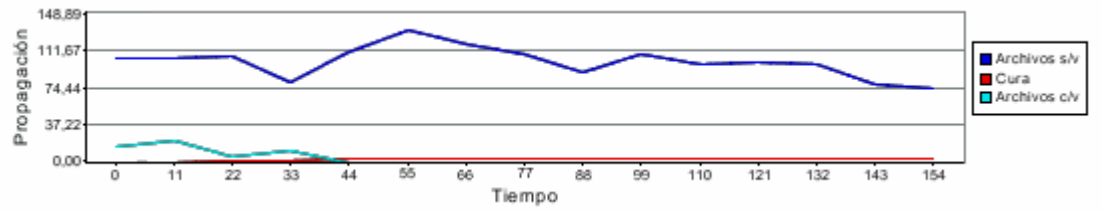


Figura 1-11 Gráfica de propagación de un virus detenido con el sistema inmune completo.

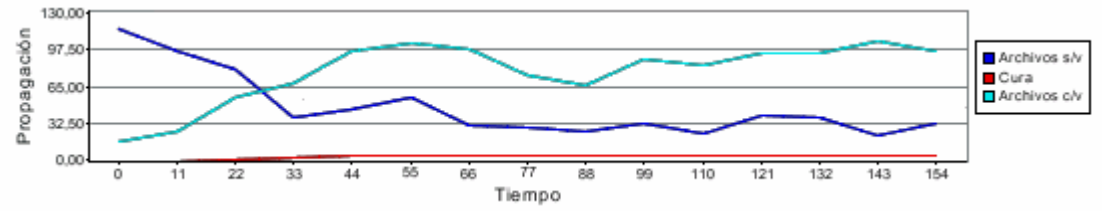


Figura 1-12 Grafica de la propagación de un virus sin sistema inmune

De estas gráficas se puede observar que en la *Computadora-0* el virus desaparece inmediatamente después de ser un riesgo para los demás, cosa que no sucede con la *Computadora-1* donde el virus llega a dominar en el sistema.

Computadora 0			Computadora 1		
Infectados	Muestra	Riesgo	Infectados	Muestra	Riesgo
17	123	0,1382	19	136	0,1397
23	128	0,1797	28	126	0,2222
8	116	0,069	57	138	0,413
13	94	0,1383	70	110	0,6364
0	111	0	98	144	0,6806
0	134	0	105	162	0,6481
0	120	0	99	132	0,75
0	110	0	77	108	0,713
0	92	0	67	94	0,7128
0	109	0	91	125	0,728

Tabla 1-6 Comparación entre una computadora con SI (Sistema Inmune) y sin SI

En la Tabla 6-6 se observa, que el virus es eliminado en su totalidad de la *Computadora-0* en un tiempo menor, comparado con la *Computadora-1* que llega a tener un nivel de riesgo 6 veces mayor que el de la *Computadora-0*.

Es interesante analizar el *Server-1*, que es una memoria o tubería donde pasa la información de los dos equipos. Este tiene el siguiente comportamiento:

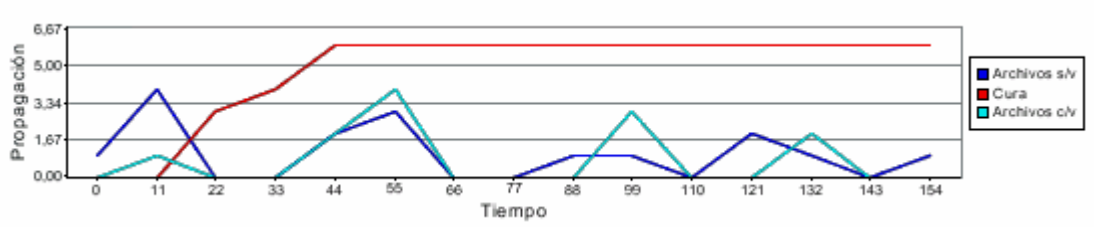


Figura 1-13 Gráfica de propagación de un servidor con un sistema inmune respaldado

Se observa que en la gráfica de la Figura 6-13 el virus intenta propagarse en algunos momentos pero de inmediato se proporciona una eliminación de éste, ya que hay un equipo que absorbe, por así llamarlo, el virus para ser eliminado. También se ve que la propagación de la vacuna tiene mayor velocidad a la propagación que presentan los virus informáticos, por lo que se tiene con esto, que la vacuna sea más rápida que la propia “enfermedad”.

1.3 Conclusiones

En esta parte del documento se analizan los puntos alcanzados por la presente tesis, basándose en los objetivos planteados al inicio de ésta. De la misma manera, se describen en forma resumida los resultados y trabajos futuros que pueden realizarse al tomar como base este trabajo.

El principal objetivo que se planteo en este trabajo es la creación de un modelo del sistema inmune que permita minimizar la curva de propagación de los virus informáticos. Para lograr este objetivo general, se muestra en el Capítulo 4 la construcción y definición de los componentes que debe tener este modelo para eliminar y prevenir los virus informáticos. Una de las conclusiones importantes fue el de crear y obtener una firma, llamada señal de peligro "SP", lo que da resultados importantes al realizar la detección de un virus y la reconstrucción del archivo huésped. Asimismo facilita la emisión de una señal de respuesta llamada señal de respuesta inmune SRI, ayudando así a estabilizar la propagación en forma global.

Las conclusiones de los objetivos específicos que se plantearon en este trabajo son:

- En el análisis de los virus informáticos se encontró que el 90% de los virus existentes tienden a realizar el tipo de evolución metamórfico para dificultar su análisis, es por esto que la señal de peligro deberá ser sintetizada para una mejor respuesta inmune.
- Durante las formas de contaminación que presentaron los virus informáticos se encuentra en un 90% los virus que modifican el estado inicial de su huésped.
- Se diseño e implanto un antivirus para analizar los problemas que estos presentan, a continuación se enuncian algunos de dichos problemas:
    - Los antivirus tienen que mantener una base de datos completa de todos los virus y vacunas.
    - El tiempo de búsqueda es proporcional al número de virus que existan aumentados en un factor por firmas de identificación que se encuentren duplicadas.
- Se creó una herramienta que elimina los virus presentes y futuros de acuerdo con las condiciones ya descritas en el Capítulo 3.
- Se creó un simulador de granularidad fina el cual permite experimentar con diferentes formas de propagación, además de permitir realizar planes de contingencia en caso de una nueva infección.

1.4 Trabajos futuros

A continuación describiremos de manera somera algunos de los trabajos futuros de los cuales puede ser base el estado actual de *"Un modelo del sistema inmune para prevenir y eliminar virus informáticos"* y que pudieran considerarse como tesis de Maestría o Licenciatura.

1.4.1 Sistema inmune local

Durante la extracción de la llamada señal de peligro, se ha realizado una representación abstracta de una gramática específica realizada para el lenguaje ensamblador 8086/80386.

Este análisis se realizó con varios filtros, uno de los principales es el llamado emulador que permite descifrar a los virus informáticos. La herramienta *"Un sistema inmune local"* que se desarrolló en el presente trabajo fue diseñada con las instrucciones más comunes para cifrar a los virus descritos en el Capítulo 3. Pudiéndose ampliar el emulador para descifrar a cualquier tipo de virus informático, ayudado por un eliminador de instrucciones irrelevantes como los casos que se presentan en el Apéndice B.

Para el ambiente distribuido se realizó una aproximación a un generador de antídotos para los virus informáticos. Aunque este módulo solamente permite trabajar con archivos binarios puros, es posible ampliarlo para trabajar con múltiples formatos de archivos ejecutables.

Asimismo los conceptos de Servidor T y Servidor B pueden mejorarse creando una técnología de comunicación que permita tener la información suficiente para solucionar el problema.

Crear *un sistema inmune de código abierto* que permita a los investigadores y estudiantes comprender de una menera técnica el desarrollo y la implantación del modelo estudiado, así como generar una herramienta que permita eliminar los virus informáticos y que sea de libre distribución.

1.4.2 Simulador

El simulador propone en su diseño muchos aspectos del desarrollo de una aplicación real, ya que se muestran muchos de los aspectos que se ven involucrados dentro de la propagación y contaminación de un virus. Es por ello que el simulador puede sufrir cambios que permitan experimentar con diferentes virus, para ello se ha realizado un diseño basado en el patrón Interprete y el patrón Constructor. Ellos permiten la creación de nuevos virus dentro del simulador y permiten analizar su comportamiento dentro de un ambiente distribuido.

El simulador puede proporcionar más información sobre la propagación de un virus informático con sólo realizar cálculos sobre el riesgo que estos generan al momento de propagarse, pudiéndose presentar un informe completo de los pormenores que se presentaron en los casos de estudio. Para lograr esto, hay que realizar una metodología que permita la captura de cierta información dentro del simulador, así como entender y analizar los resultados presentados por el mismo.

Los componentes que presenta el simulador de granularidad fina únicamente cubren aspetos esenciales para un correcto funcionamiento. Se pueden agregar componentes que permitan optimizar el envío de señales de peligro entre los llamados servidores T.

Capítulo 8

CONCLUSIONES.

En esta parte del documento, se analizan los puntos alcanzados por la presente tesis, basándose en los objetivos planteados al inicio de ésta. De la misma manera, se describen en forma resumida los resultados y trabajos futuros que pueden realizarse al tomar como base este trabajo.

El principal objetivo que se planteó en este trabajo es la creación de un modelo del sistema inmune que permite minimizar la curva de propagación de los virus informáticos, para lograr este objetivo general, se muestra en el Capítulo 4 la construcción y definición de los componentes que debe tener este modelo para eliminar y prevenir los virus informáticos. En donde una de las conclusiones importantes fue el de crear y obtener una firma, llamada señal de peligro “SP”. Lo que da resultados importantes al realizar la detección de un virus y la reconstrucción del archivo huésped. Asimismo facilita la emisión de una señal de respuesta llamada señal de respuesta inmune SRI, ayudando así a estabilizar la propagación en forma global.

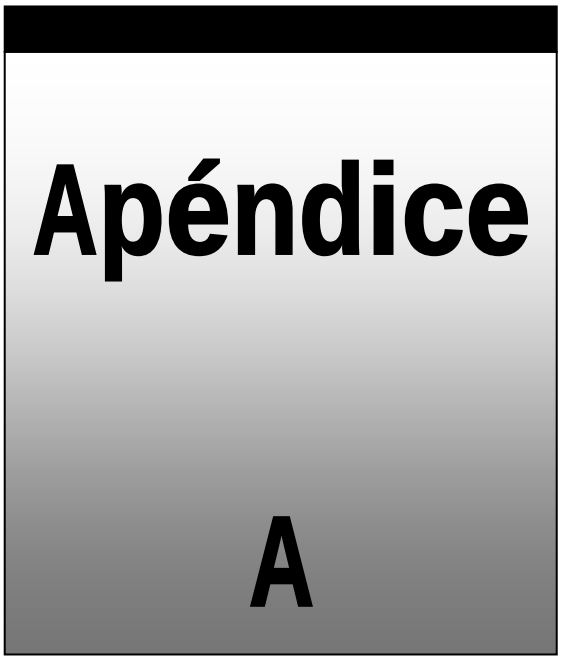
Las conclusiones de los objetivos específicos que se plantearon en este trabajo son:

- En el análisis de los virus informáticos se encontró que el 90% de los virus existentes tienden a realizar el tipo de evolución metamórfico para dificultar su análisis, es por esto que la señal de peligro deberá ser sintetizada para una mejor respuesta inmune.

- Durante las formas de contaminación que presentaron los virus informáticos son contaminaciones del estado inicial (principio del programa), por lo descrito en el Capítulo 3.

- Se realizó un antivirus para analizar los problemas que estos presentan, a continuación se enuncian algunos de ellos:
    - Los antivirus tienen que mantener una base de datos completa de todos los virus y vacunas.
    - El tiempo de búsqueda es proporcional al número de virus que existan aumentados en un factor por firmas de identificación que se encuentren duplicadas.
- Se creó una herramienta que elimina los virus presentes y futuros de acuerdo con las condiciones ya descritas en el Capítulo 3.

- Se creó un simulador de granularidad fina, el cual permite experimentar con diferentes formas de propagación, además de permitir realizar planes de contingencia en caso de una nueva infección.



Apéndice A. Diseño de un simulador basado en UML

Resumen

Se busca presentar una representación del simulador basándose en patrones de diseño y UML, y mostrar los aspectos más relevantes que intervienen durante la simulación.

Objetivos

- Definir los casos de usos que intervienen en la propagación y eliminación de virus informáticos.
- Presentar el diseño de módulos que compone el simulador, basándose en los patrones de diseño.

A.1 Casos de Uso

Caso de uso:	Programa
Actores:	Programador de Virus
Propósito:	Crear un programa con virus que pertenezca al lenguaje de una máquina.
Resumen:	Un programador que crea un programa con virus y lo coloca en el contenedor para disposición de todos los demás actores.
Tipo:	Primario y Esencial.

Caso de uso:	Virus
Actores:	Programador de Virus
Propósito:	Módulo que se programa y se incorpora dentro de un programa para que tenga la capacidad de autocopiado.
Resumen:	Es la parte esencial de un virus, ya que se diseña el virus que se experimentará en el sistema.
Tipo:	Primario y esencial.

Caso de uso:	Máquina
Actores:	Programador de Virus
Propósito:	Es la encargada de ejecutar cualquier programa, pero en el que se incluyen las reglas sintácticas y semánticas de un lenguaje.
Resumen:	Una máquina es la encargada de proporcionar el lenguaje que interpreta.
Tipo:	Primario y esencial

Caso de uso:	Lenguaje
Actores:	Programador de Virus
Propósito:	Proporciona las reglas e instrucciones de un lenguaje estructurado.
Resumen:	Crea las reglas estructuradas para que una máquina ejecute cualquier programa.
Tipo:	Primario y Esencial.

Caso de uso:	Contenedor
Actores:	Programador de Virus
Propósito:	Contiene los archivos con programas.
Resumen:	Contiene los archivos temporalmente, para después ser enviados a otros dispositivos de almacenamiento.
Tipo:	Primario y Esencial.

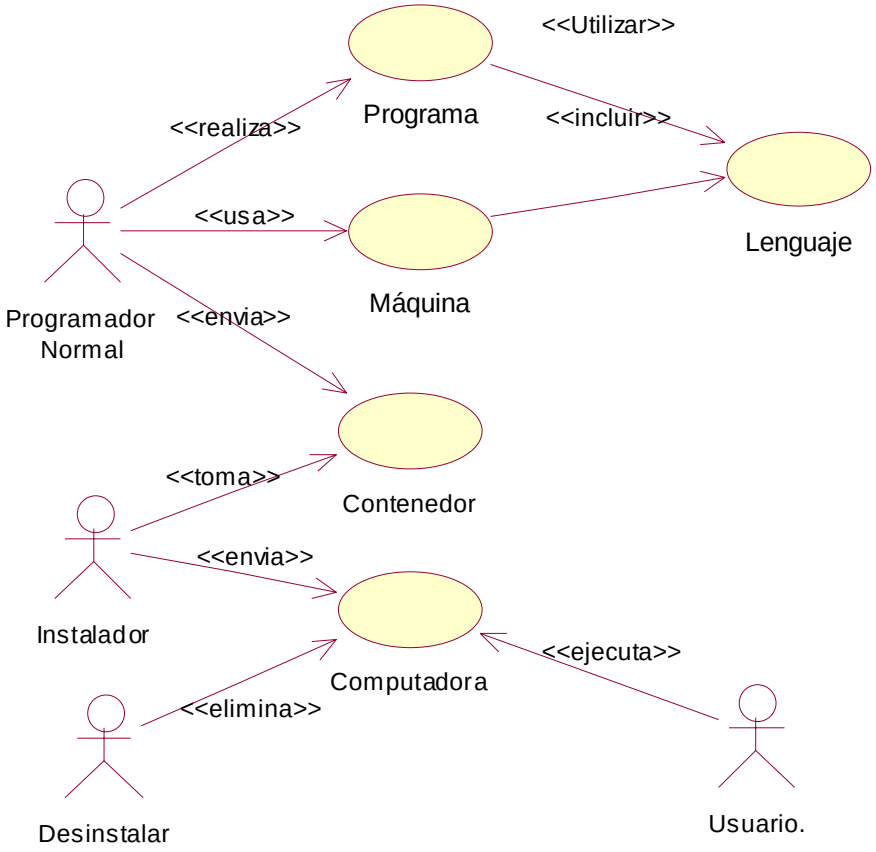


Figura A-1 Los casos de uso para una computadora

A.2 Singleton

Garantiza que sólo se llegue a crear una instancia de la clase *ConfigurarServidorB* y provee un punto de acceso global a él, mediante el método estático *Tomar Referencia*. Todos los objetos que utilizan una instancia de esta clase usan la misma instancia.

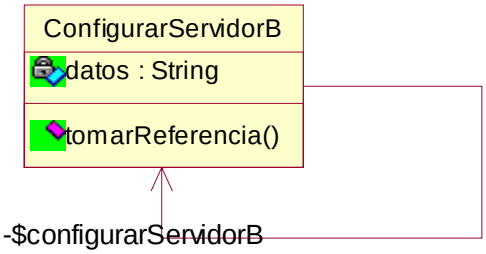


Figura A-2 Patrón Singleton

Descripción de la clase:

La clase *ConfigurarServidorB* es la encargada de presentar una interfaz gráfica que permite configurar todos los objetos de este tipo de *servidor*.

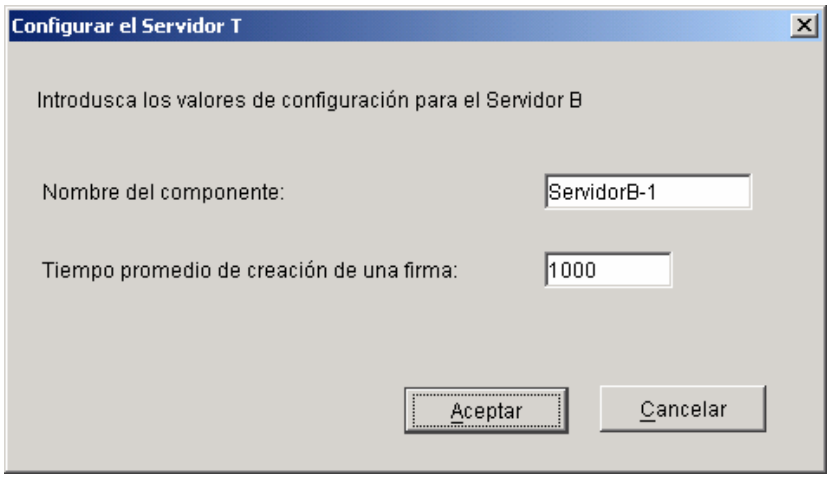


Figura A-3 Configuración del Servidor T.

Otras clases que son similares a ésta son:

ConfigurarEquipos, ConfigurarGrupo, ConfigurarProgramadorNormal, ConfigurarProgramadorVirus, ConfigurarServidorT.

A.3 Estrategia (Strategy)

Encapsula algoritmos de validación de flujos relacionados en clases que son subclases de una superclase llamada *Checksum*. Esto permite la selección de un algoritmo de validación, que varia según el objeto y también le permite la variación en el tiempo.

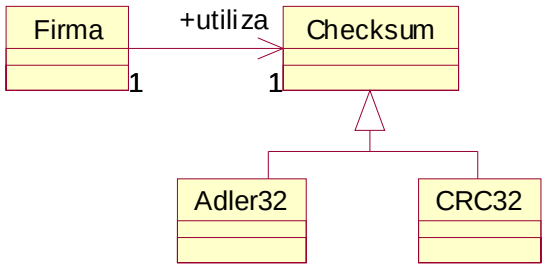
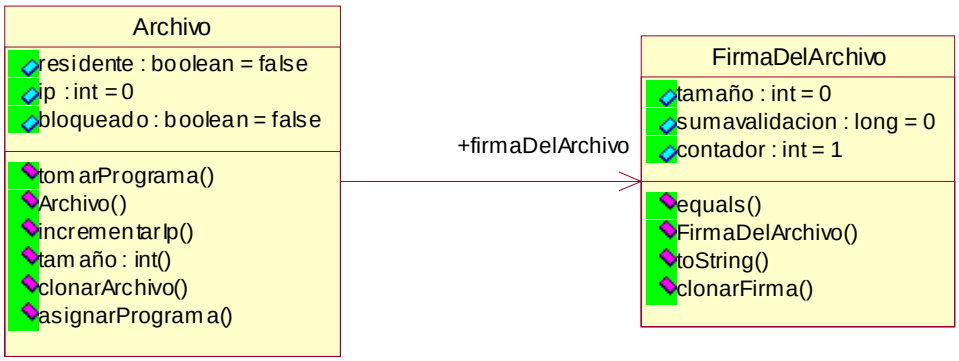


Figura A-4 Patrón Estrategia (Strategy).

En el momento en que un programa ha sido firmado se utilizan un *checksum*[1] y para estimar alguna alteración. La firma obtenida se utiliza para realizar una verificación rápida.



[1] Suma de validación

A.4 Observador (Observer)

Permite al objeto *DetectorVirus* captar dinámicamente las dependencias entre objetos *Archivo*, de tal forma que el objeto *ContenedorArchivos* notificará a los objetos dependientes de él en el momento en que cambia su estado, siendo actualizados automáticamente.

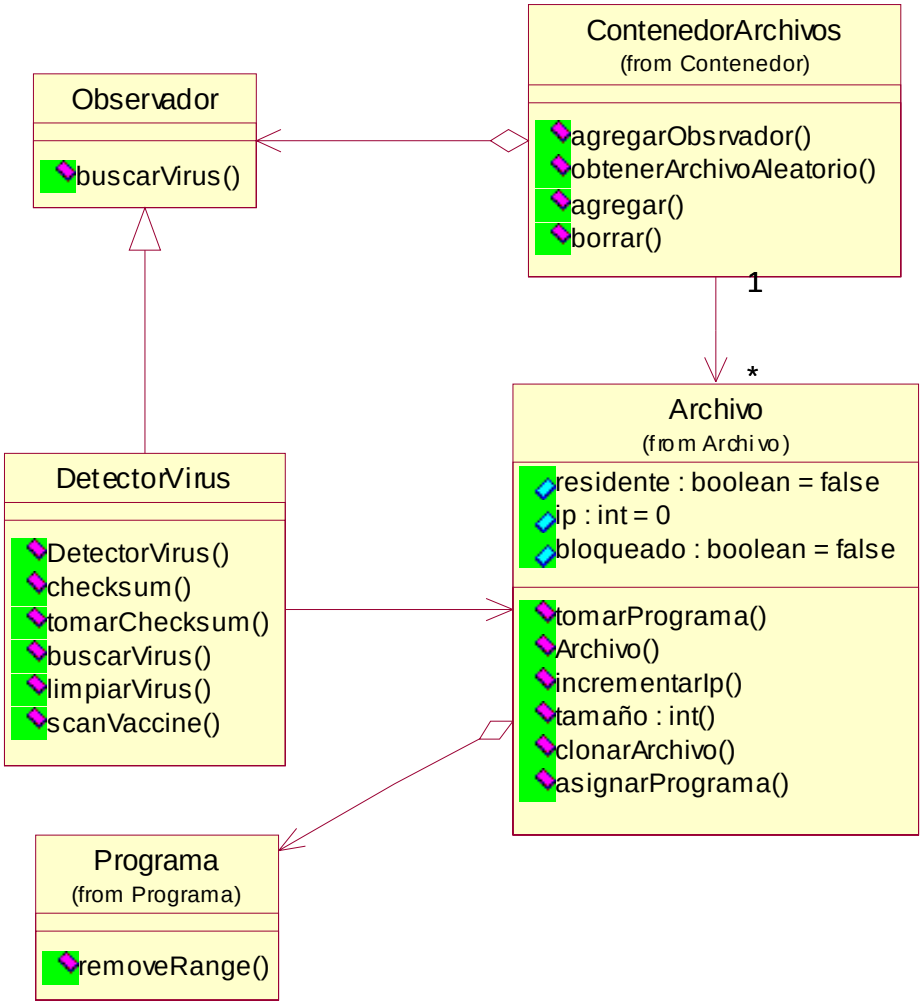


Figura A-5 Patrón Observador (Observer).

Este patrón se ha utilizado para observar si un archivo ha cambiado y realizar un análisis de firmas. Todo contenedor de archivos deberá ser "observado" para crearles una firma en el momento en que ingresen.

A.5 Prototipo (Prototype)

El objeto *Editor* crea objetos personalizados sin conocer la clase exacta o los detalles de su creación. Su tarea es dar objetos prototipo a un objeto que inicia la creación de objetos como es *IconJPanel*. El objeto de creación e iniciación crea objetos para mandar sus objetos prototipo que hagan una copia de sí mismos.

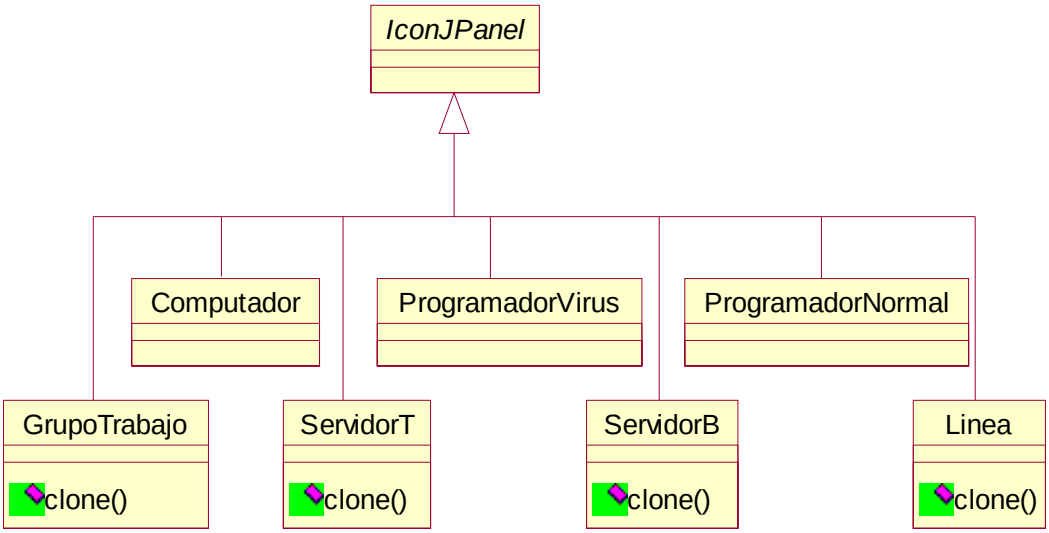


Figura A-6 Patrón Prototipo (Prototype).

Con el Editor Visual se crean los diferentes modelos, dentro del editor existe una barra de herramientas que permiten crear diferentes objetos y tratarlos sin conocer su clase exacta.

Figura A-7 Barrar de herramientas.

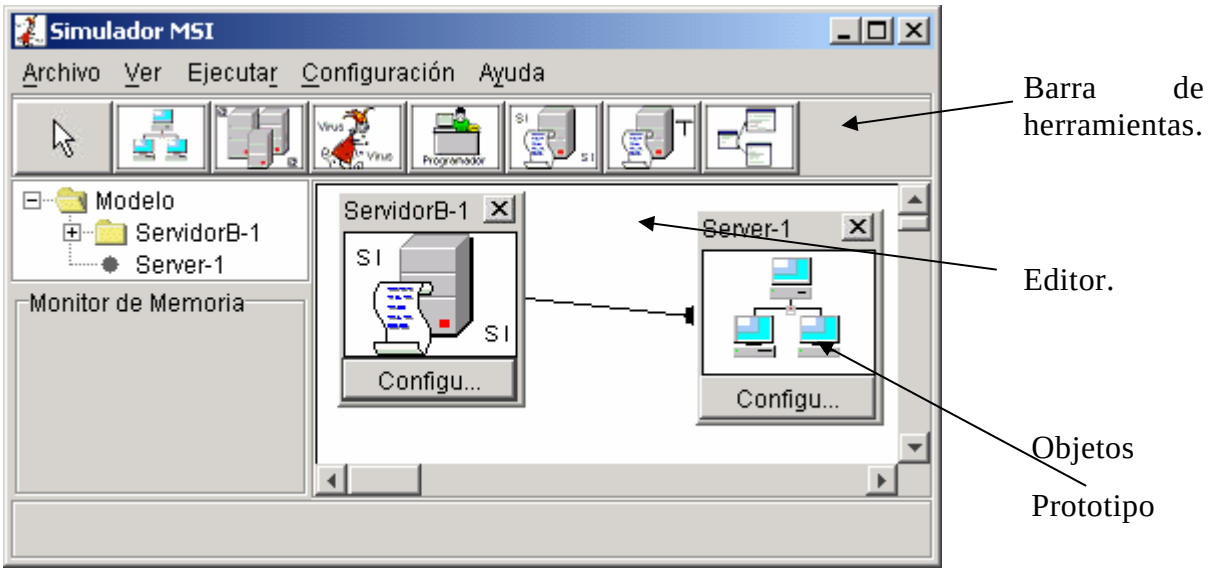


Figura A-8 Vista del simulador.

A.6 Compuesto (Composite)

Permite construir objetos complejos mediante composición recursiva de objetos similares. El patrón Composite también permite que los objetos del árbol sean manipulados por un manejador consistente, para requerir todos los objetos hay una superclase o una interfaz común. Permite a los clientes tratar de la misma manera tanto a objetos individuales como a compuestos.

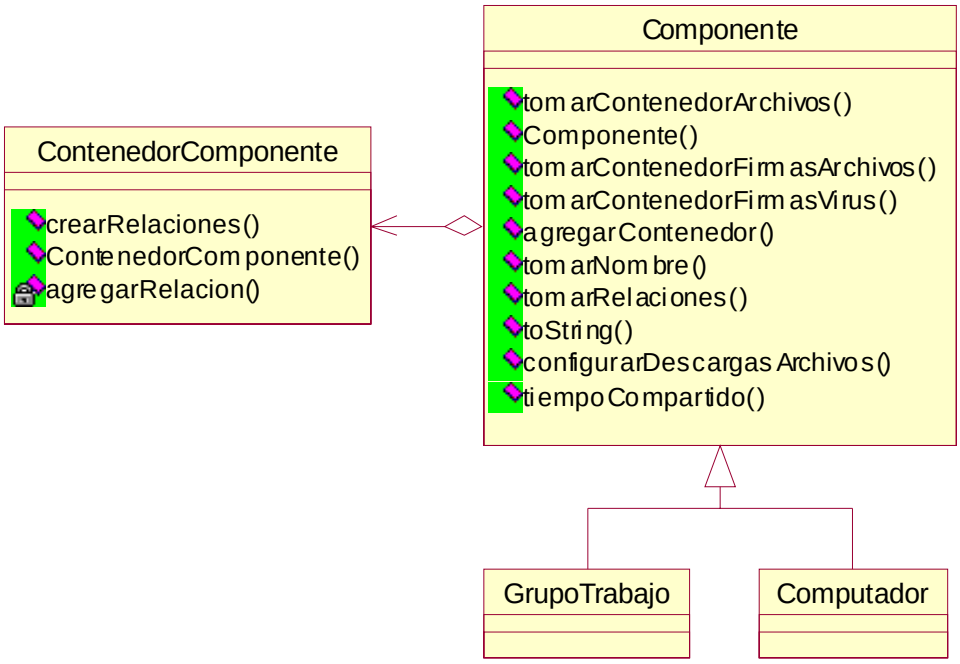


Figura A-9 Patrón Compuesto (Composite).

Este patrón se utiliza para tener un objeto llamado *GrupoTrabajo* que estará compuesto por objetos *Computadores*. Para analizar mejor dónde se da esta configuración se observa la siguiente Figura.

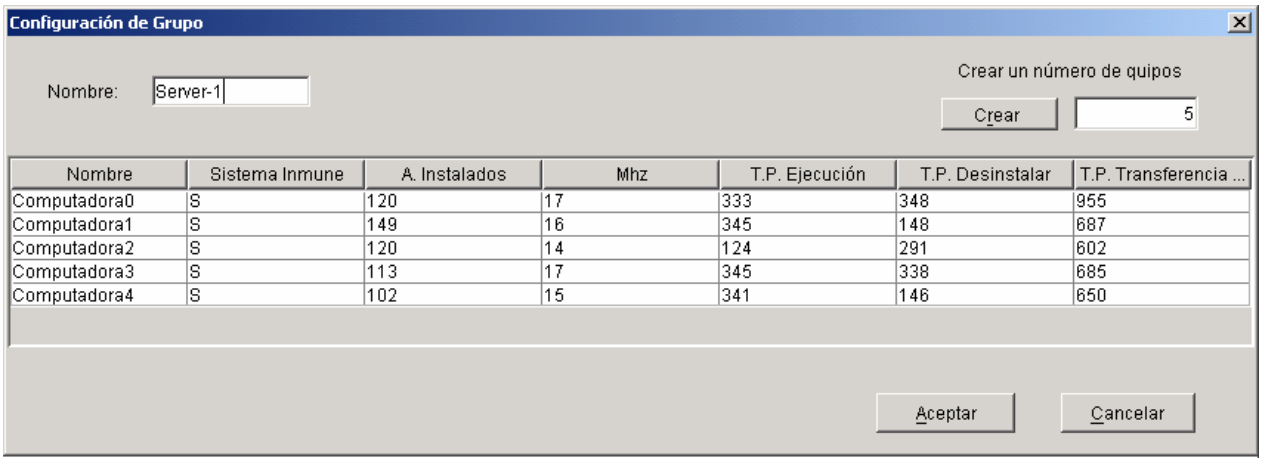


Figura A-10 Configuración del Grupo de Trabajo.

En la Figura anterior se observa cómo un *Grupo de Trabajo* puede llegar a contener *n* números de equipos de cómputo y estar conectados entre ellos. Estos equipos de cómputo tienen configuraciones diferentes.

A.7 Decorador (Decorator)

Extiende la funcionalidad de un objeto dinámicamente de tal modo que es transparente a sus clientes, utilizando una instancia de una subclase de la clase original que delega las operaciones al objeto original. Además provee una alternativa flexible para agregar una funcionalidad a una clase.

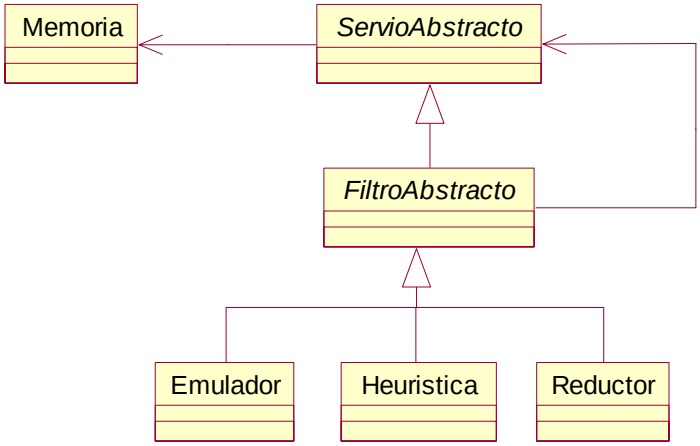


Figura A-11 Patrón Decorador (Decorator).

En el momento en que se trata de incrementar funcionalidad en forma dinámica, es útil este patrón debido a que cada evolución de virus presenta nuevas formas en que logra codificar. Es por ello que el Filtro se debe incrementar y permitir descodificar a los virus para presentarlo en forma más pura.

A.8 Estado (State)

Encapsula los estados de un objeto como objetos separados, cada pertenencia en una subclase separada de una clase abstracta. Permite a un objeto cambiar su comportamiento en el momento en que su estado interno cambia. El objeto parecerá haber cambiado de clase.

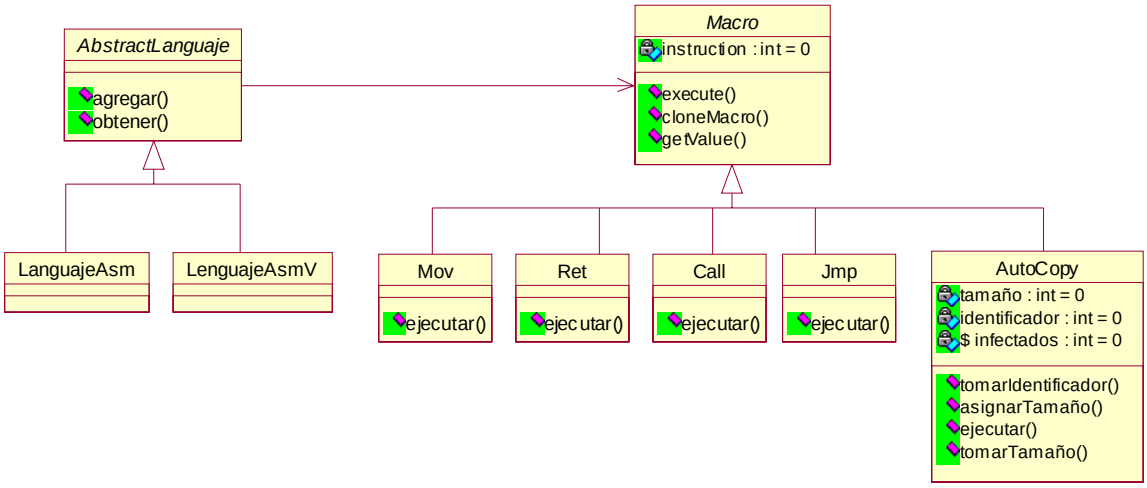
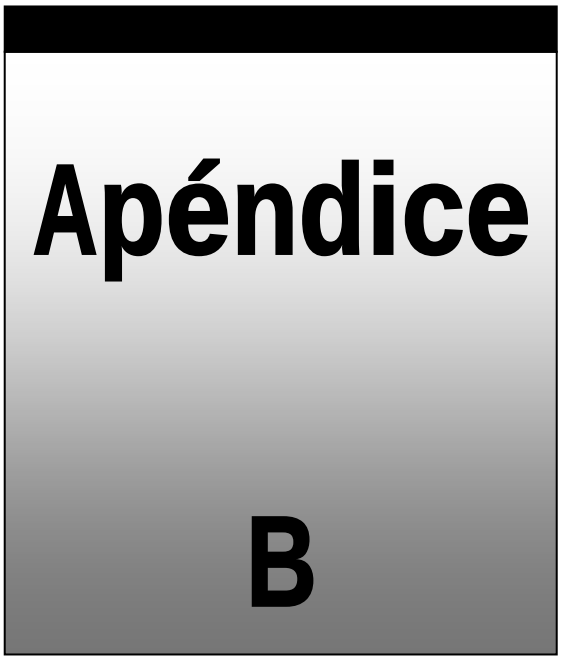


Figura A-12 Patrón Estado (State).

Una forma de crear aplicaciones normales y aplicaciones con virus informáticos es utilizar un lenguaje y el patrón *State* que permiten cambiar su comportamiento en el momento en que su estado interno cambia.

Apéndice A. Análisis de códigos en ensamblador para detectar instrucciones irrelevantes



Resumen

En este apéndice se muestran algunas partes del código que utiliza el virus Zombie y se ha tratado de explicar su emulación; así como algunos de los problemas que surgen al momento de realizarla.

Objetivos

- Presentar técnicas que son utilizadas por los virus para evolucionar un programa.
- Análisis de frecuencias de códigos en el lenguaje ensamblador 8086/368
- Técnicas y funciones que se utilizan para la reducción de los códigos irrelevantes.

A.1 Frecuencia de Instrucciones en programas ejecutables del tipo PE

Primero se muestra la frecuencia de alguna de las instrucciones del microprocesador Intel 8086/386 encontradas en más de 15,000 archivos ejecutables de la plataforma Microsoft Windows.

Total de archivos procesados: 1,500

Total de instrucciones procesadas: 41,000,000

Código	Frecuencia	%	Utilizado como:
8B	6588972	15%	mov modr/m
FF	2736427	6%	push modr/m
E8	2509098	6%	call
83	2240872	5%	cmp/add modr/m (incluye add esp, xx después call)
89	2045125	4%	mov modr/m
8D	1573296	3%	lea modr/m
50	1423289	3%	push eax
74	1269798	3%	jz
6A	1064820	2%	push xx
85	1001107	2%	test r,r
0F	939376	2%	0F xx
56	882376	2%	push esi
75	845429	2%	jnz
33	781974	1%	xor r,r
53	740703	1%	Push ebx
66	738157	1%	operand-size modifier prefix (-->16-bit)
EB	734922	1%	jmp xx
68	705038	1%	push imm32
57	679402	1%	push edi
C7	639613	1%	mov modr/m, imm
E9	616969	1%	jmp
C3	518251	1%	retn
5E	515151	1%	pop esi
3B	503023	1%	cmp r,r
55	467792	1%	push ebp
51	465043	1%	push ecx
59	454977	1%	pop ecx (after call)
C2	423134	1%	retn n
5B	388365	0%	pop ebx
5F	378583	0%	pop edi
B8	361314	0%	mov eax, c
5D	357410	0%	pop ebp
52	303136	0%	push edx

Tabla B-1 Frecuencia de instrucciones de algunos programas ejecutables.

A.2 Emulación del código

Para obtener las frecuencias de los programas los creadores del virus Zombie proponen una forma de Emular y obtener los códigos correspondientes, para ello se muestra la siguiente parte del código, que presentan algunas instrucciones que son semánticamente interpretadas.

```
1.  DWORD nxt = ip + len;
2.  DWORD rel = NONE;
3.
4.  BYTE o = memb[ip];
5.  WORD w = memw[ip];
6.
7.  if (o == 0x2E) // CS:
8.  {
9.   o = memb[ip+1];
10.  w = memw[ip+1];
11. }
12.
13. if (((o&0xF0)==0x70)||((o&0xFC)==0xE0)||(o==0xEB)) // jcc,jcxz,loop/z/nz,jmps
14. {
15.  rel = nxt + (char)(memb[nxt - 1]);
16.  arg2[ip] = 1;
17. }
18.
19. if (((w&0xF0FF)==0x800F)||(o==0xE8)||(o==0xE9)) // jcc near,call,jmp
20. {
21.  rel = nxt + (long)(memd[nxt - 4]);
22.  arg2[ip] = 4;
23. }
24.
25. if ((o==0xEB)||(o==0xE9)) nxt=NONE; // jmps,jmp
26.
27. if (((o&0xF6)==0xC2)||(o==0xCF)||((w&0x38FF)==0x20FF))
28.  nxt=NONE; // ret/ret#/retf/retf#/iret/jmp modrm
```

Código B-1 Parte del emulador basado en el lenguaje ensamblador.

Se observa en el Código B-1 que se trata de manejar los saltos que tiene un programa, para esto se maneja un arreglo de *memb* el cual contiene el código del programa en análisis, así como un *ip*, que es un apuntador a la instrucción que se encuentra en ejecución o también llamada “cabeza de cinta”. La primera comparación que se realiza en el Código B-1 y la línea 5 es para conocer si el segmento de código se ignora, debido a que la emulación no trabaja con los segmentos de pila, las comparaciones en las líneas 10, 15, 20, son utilizadas para realizar los cálculos que identifican los saltos que existen dentro del programa, y en la línea 21 se ignora de nuevo los diferentes retrocesos, ya que como se ha dicho no maneja una pila.

A.3 Diferentes técnicas para crear y detectar la mutación

Lo primero es crear un archivo PE como un placebo; también conocido como BAIT, posteriormente se le aplica una rutina para provocar la metamorfosis.

```
#include <vcl.h>
#include <stdio.h>
#pragma hdrstop
//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
        int a=0;
        if ( a )
                goto c1 ;
        else
                goto c2 ;
c1:
        printf("A es igual a cero \n");
        goto c3;
c2:
        printf("A es igual a uno \n");
c3:
        return 0;
}
//---------------------------------------------------------------------
```

Código B-2 Ejemplo de un programa en el lenguaje C para ser mutado.

Este código se escribió en lenguaje C aunque se han utilizado instrucciones que no son frecuentes; estas permiten observar claramente cómo se analiza el código bajo una emulación, pero antes se verá como queda compilado el código en el lenguaje ensamblador 8086/386.

```
:004012F8 55                    push ebp
:004012F9 8BEC                  mov ebp, esp
:004012FB 51                    push ecx
:004012FC 33C0                  xor eax, eax
:004012FE 8945FC                mov dword ptr [ebp-04], eax
:00401301 837DFC00              cmp dword ptr [ebp-04], 00000000
:00401305 7502                  jne 00401309
:00401307 EB0D                  jmp 00401316
:00401309 6840214000            push 00402140
:0040130E E88D050000            Call 004018A0
:00401313 59                    pop ecx
:00401314 EB0B                  jmp 00401321
:00401316 6854214000            push 00402154
:0040131B E880050000            Call 004018A0
:00401320 59                    pop ecx
:00401321 33C0                  xor eax, eax
:00401323 59                    pop ecx
:00401324 5D                    pop ebp
:00401325 C3                    ret
```

Código B-3 Programa compilado en el lenguaje ensamblador.

Para analizar más claramente como se ha creado se presenta el siguiente código:

```
NEWEXE.cpp.8: int main(int argc, char* argv[])
004012F8 55               push ebp
004012F9 8BEC             mov ebp,esp
004012FB 51               push ecx
NEWEXE.cpp.10: int a=0;
004012FC 33C0             xor eax,eax
004012FE 8945FC           mov [ebp-0x04],eax
NEWEXE.cpp.11: if ( a )
00401301 837DFC00         cmp dword ptr [ebp-0x04],0x00
00401305 7502             jnz +0x02
NEWEXE.cpp.14: goto c2   ;
00401307 EB0D             jmp +0x0d
NEWEXE.cpp.16: printf("A es igual a cero \n");
00401309 6840214000       push 0x00402140
0040130E E88D050000       call CC3250MT._printf
00401313 59               pop ecx
NEWEXE.cpp.17: goto c3;
00401314 EB0B             jmp +0x0b
NEWEXE.cpp.19: printf("A es igual a uno  \n");
00401316 6854214000       push 0x00402154
0040131B E880050000       call CC3250MT._printf
00401320 59               pop ecx
NEWEXE.cpp.21: return 0;
00401321 33C0             xor eax,eax
NEWEXE.cpp.22: }
```

Código B-4 Lenguaje C y su representación en Ensamblador 80386

A.3.1 Mutación con operaciones del tipo NOP

Se modifica el programa en forma mutable con operaciones del tipo *NOP* intercaladas en el programa, lo que dará el siguiente código:

```
:00401372 55                    push ebp
:00401373 90                    nop
:00401374 8BEC                  mov ebp, esp
:00401376 90                    nop
:00401377 51                    push ecx
:00401378 90                    nop
:00401379 33C0                  xor eax, eax
:0040137B 90                    nop
:0040137C 8945FC                mov dword ptr [ebp-04], eax
:0040137F 90                    nop
:00401380 837DFC00              cmp dword ptr [ebp-04], 00000000
:00401384 90                    nop
:00401385 7503                  jne 0040138A
:00401387 90                    nop
:00401388 EB10                  jmp 0040139A
:0040138A 6840314000            push 00403140
:0040138F 90                    nop
:00401390 E892060000            Call 00401A27
:00401395 90                    nop
:00401396 59                    pop ecx
:00401397 90                    nop
:00401398 EB0D                  jmp 004013A7
:0040139A 6854314000            push 00403154
:0040139F 90                    nop
:004013A0 E882060000            Call 00401A27
:004013A5 90                    nop
:004013A6 59                    pop ecx
:004013A7 33C0                  xor eax, eax
:004013A9 90                    nop
:004013AA 59                    pop ecx
:004013AB 90                    nop
:004013AC 5D                    pop ebp
:004013AD 90                    nop
:004013AE C3                    ret
```

Código B-5 Programa basado en el 80386 mutado con operación Nop intercalada.

A.3.2 Eliminación de códigos irrelevantes NOP

Para extraer una firma se debe optimizar el Código B-1. El cual presenta instrucciones o u operaciones de tipo NOP que se deberá omitir.

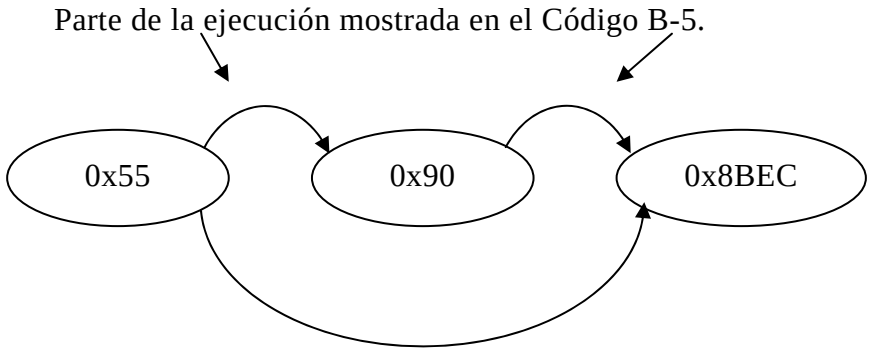


Figura B-1 La eliminación de instrucciones irrelevantes NOP

A.3.3 Una mutación con instrucciones irrelevantes

Otra forma de mutar un código es intercalar las instrucciones irrelevantes como se observa en el siguiente ejemplo:

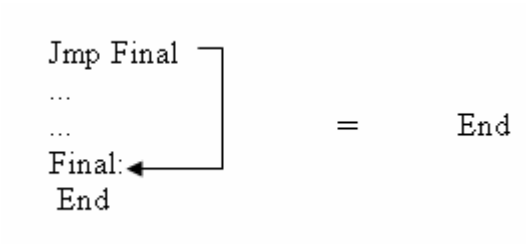
```
:00401229 E9CF0C0000          Jmp 00401EFD
:0040122E EBF8                jmp 00401228
:00401230 5D                  pop ebp
:00401231 EB03                jmp 00401236
:00401233 95                  xchg eax,ebp
:00401234 93                  xchg eax,ebx
:00401235 89C3                mov ebx, eax
:00401237 90                  nop
:00401238 90                  nop
:00401239 832DEC32400001      sub dword ptr [004032EC], 00000001
:00401240 C3                  ret
```

Código B-6 Mutar un programa con inserción de otras instrucciones irrelevantes

Eliminación de instrucciones irrelevantes basado en saltos innecesarios.

Aparentemente parte del Código B-3 y el Código B-6 son diferentes, pero semánticamente son iguales; ahora bien si se optimiza el Código B-6, se ve que son prácticamente iguales para ello se debe tomar la siguiente regla que ayuda a optimizar de una mejor forma. Toda

instrucción que ejecute un salto hacia un retorno o final de un proceso podrá ser optimado como el mismo final del proceso.



Código B-7 La eliminación de las instrucciones irrelevantes mediante saltos

En el Código B-6 se observa un salto en la dirección *00401231*. Es obvio que no existe esa dirección en el código; pero a la mitad de la instrucción marcada con la dirección *00401235* es donde el micro procesador Intel realizaría la referencia, por lo que sí se desensambla esa instrucción *C3* sería un *ret*, que es la que se está buscando. Si se utiliza la eliminación de instrucciones irrelevantes daría el mismo código.

```
:00401229 E9CF0C0000              Jmp 00401EFD
:0040122E EBF8                    jmp 00401228
:00401230 5D                      pop ebp
:00401231 EB03                    jmp 00401236
:00401233 95                      xchg eax,ebp
:00401234 93                      xchg eax,ebx
:00401235 89C3                    mov ebx, eax
:00401237 90                      nop
:00401238 90                      nop
:00401239 832DEC32400001          sub dword ptr [004032EC], 00000001
:00401240 C3                      ret
```

Código B-8 Eliminado las instrucciones irrelevantes

Otra forma de mutar el código es cambiar las instrucciones durante la ejecución. Para observar este fenómeno se observa el siguiente ejemplo:

```
:00401326 55                    push ebp
:00401327 9C                    pushfd
:00401328 802D3013400005        sub byte ptr [00401330], 05
:0040132F 9D                    popfd
:00401330 90                    nop
:00401331 EC                    in al, dx
:00401332 9C                    pushfd
:00401333 802D30134000FB        sub byte ptr [00401330], FB
:0040133A 9D                    popfd
:0040133B 8B4510                mov eax, dword ptr [ebp+10]
:0040133E 9C                    pushfd
:0040133F C0054713400001        rol byte ptr [00401347], 01
:00401346 9D                    popfd
:00401347 C55508                lds edx, dword ptr [ebp+08]
:0040134A 9C                    pushfd
:0040134B C00D4713400001        ror byte ptr [00401347], 01
:00401352 9D                    popfd
:00401353 807D0C00              cmp byte ptr [ebp+0C], 00
:00401357 7410                  je 00401369
:00401359 C605F032400001        mov byte ptr [004032F0], 01
:00401360 C605F132400001        mov byte ptr [004032F1], 01
:00401367 EB15                  jmp 0040137E
```

Código B-9 La mutación de un programa por el cambio de las instrucciones

Para minimizar las instrucciones que se han agregado al Código B-1 y que se ven en el Código B-9, se utiliza un emulador para determinar ciertas operaciones con el mismo segmento de código. Como ejemplo se puede corregir el Código B-9 de la siguiente forma:

Primero se observa que existen instrucciones que permiten realizar operaciones aritméticas; una de ellas se ve en la dirección *0040132*, donde realiza una resta con la dirección *00401330* que tiene el código *90 NOP*, si se le resta *05* dará *8B*. Las instrucciones *pushfd* y *popfd* se utiliza para que no existan cambios con respecto al registro de banderas, ya que estos cambian en el momento en que hay un acarreo.

Otro aspecto que puede ayudar a la detección inmediata de un programa es obtener su estadística de instrucciones, para esto se analiza Código B-1 en forma completa:

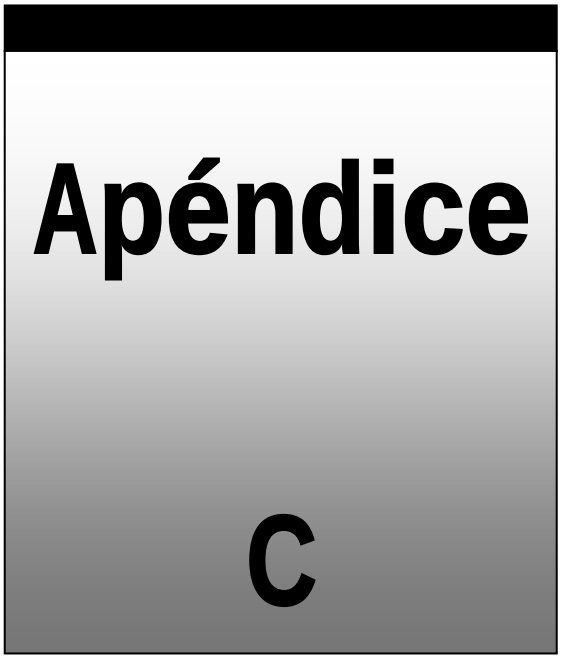
A.4 Estadística de instrucciones utilizadas con diferentes técnicas de mutación

Código	Programa Normal		Mutado con instrucción NOP		Mutado con instrucciones JMP	
0xEB					**395**	**43%**
0x90			**391**	**42%**		
0xFF	63	12%	63	6%	63	6%
0x8B	55	10%	55	6%	55	6%
0xE8	53	10%	53	5%	53	5%
0xC3	30	5%	30	3%	30	3%
0x89	26	4%	26	2%	26	2%
0x83	25	4%	25	1%	25	1%
0x55	16	3%	16	1%	16	1%
0x8D	16	3%	16	1%	16	1%
0x59	14	2%	14	1%	14	1%
0x5D	14	2%	14	1%	14	1%
0x33	13	2%	13	1%	13	1%
0x75	12	2%	12	1%	12	1%
0xA3	12	2%	12	1%	12	1%
0x64	11	2%	11	1%	11	1%
0x68	11	2%	11	1%	11	1%
0xA1	11	2%	11	1%	11	1%
0xC7	11	2%	11	1%	11	1%
0x53	10	1%	10	1%	10	1%
0x50	9	1%	9	0%	9	0%
0x5B	9	1%	9	0%	9	0%
0x80	9	1%	9	0%	9	0%
0x74	7	1%	7	0%	7	0%
0x66	6	1%	6	0%	6	0%
0xB8	6	1%	6	0%	6	0%
0xBA	6	1%	6	0%	6	0%

Tabla B-2 Comparación de instrucciones de los diferentes motores de mutación

Se observa que el introducir instrucciones llamadas irrelevantes producen un incremento notable, aunque el motor de mutación que se utilizo podría ser modificado para que el código sea visible. Esto provocaría que la velocidad de la emulación se vea disminuida y la detección sea producida por búsquedas heurísticas.

Apéndice A. Los analizadores heurísticos



Resumen

Se busca presentar los problemas más comunes que presentan los analizadores heurísticos al momento de realizar la detección de un virus informático.

Objetivos

- Definir el concepto de analizador heurístico dentro del ámbito de los virus informáticos.
- Presentar la funcionalidad y la técnica de este tipo de buscador.

A.1 Los Analizadores heurísticos

La palabra heurística fue creada por los griegos, su forma original era *heuriskein* donde su significado era descubrir. Hoy el término es utilizado en el área de las ciencias de la computación, para describir los algoritmos que son efectivos para resolver problemas complejos en una forma más rápida, pero con un rendimiento menor a la solución óptima. Para analizar este concepto, se toma el ejemplo típico del vendedor viajero, que es un problema clásico en la computación, en el que se han ideado soluciones conocidas como heurísticas.

Un vendedor tiene que viajar por varias ciudades diferentes, para visitar cada ciudad, por lo menos una vez, ya que la intención es minimizar el tiempo del viaje y el costo de los aviones. Al estudiar el problema, se tomaría un tiempo considerable para estimar la solución ideal. Los científicos propusieron los algoritmos heurísticos para ayudar a resolver el problema en un menor tiempo. Por su naturaleza un algoritmo heurístico no garantiza ser la mejor solución posible, aunque en algunas ocasiones este tipo de algoritmos se aproxima a la solución óptima. Lo que se hace es crear el itinerario del viajero, bajo este método sin que se tenga que esperar cien años para realizar su viaje.

En el campo de los antivirus se pretende resolver problemas complejos; algunos de los problemas con que se encuentran son aún más difíciles de resolver que el problema del vendedor viajero, de hecho algunos problemas de antivirus se consideran sin solución. En otras palabras, sería casi imposible proponer una mejor solución exacta en un tiempo determinado y la única manera de resolverlos es usar algoritmos heurísticos.

Si un programa de cómputo tiene virus, es un problema que no puede ser solucionado: es imposible escribir un programa que sea capaz de realizar un diagnóstico con el 100% de éxito con los virus existentes y los que falten por escribirse. Lamentablemente existe el problema de discernir entre un programa con virus y un programa normal por así llamarlo, los investigadores de antivirus han ideado distintos métodos heurísticos para ayudar a discernir a los virus informáticos. Hoy en día, la mejor técnica conocida es la identificación de firmas, se piensa que esta técnica no es heurística aunque de alguna manera sí lo es.

Un programa de antivirus que busca firmas, debe mantener una base de datos que contenga a todas ellas. Como se mencionó, cada firma es una sucesión corta de *n* bytes que se extraen del cuerpo del virus. Los antivirus tienen firmas diferentes para cada virus y comprenden un porcentaje pequeño del conjunto total de los bytes que componen la lógica de virus. Esta característica que tienen los antivirus reduce la probabilidad de una identificación falsa para algún programa que no esté infectado.

Los antivirus que usan la técnica de búsqueda por firmas, identifican si un programa contiene esas firmas; pero se ven limitados al momento de diagnosticar con seguridad si el programa se encuentra infectado con un virus. En el momento en que el programa es identificado con un virus, éste podría tener datos al azar para impedir que el diagnóstico sea correcto. Mientras exista la posibilidad de que se identifique un virus dentro de un programa, se puede decir que este método es heurístico.

La técnica de buscar una firma es una tecnología empleada por programas de antivirus. El problema que tienen estos es el de mantener la base de datos actualizada, ya que los escritores de virus crean nuevas especies variantes en un menor que las vacunas generadas por los antivirus.

Como se ha visto, la tecnología que tienen los antivirus, emplea métodos heurísticos, sin embargo, en la industria de los antivirus, el término heurístico es usado por investigadores de antivirus para discernir y analizar las conductas de los programas por medio de patrones.

Cada vez que un programa se ejecuta, se activan los antivirus para analizar las instrucciones lógicas, así como los datos que contiene el archivo, después se analiza el programa en búsqueda de alguna infección. También este tipo de análisis puede fallar si se encuentra con un archivo que posee los mismos patrones, o simplemente si no tienen registrado algún patrón nuevo.

Los buscadores heurísticos actuales han logrado un índice de descubrimiento en virus nuevos y desconocidos entre el factor de un 70 a un 80%. Estas cifras son aceptadas con éxito por las empresas certificadoras de antivirus, dadas las dificultades del problema. Los diferentes productos antivirus que implementan esta técnica muestran algunas variaciones en los resultados, pero son mínimas.

Para analizar cómo realizan está búsqueda se realizará un ejemplo donde se muestre claramente el uso de está técnica. Por lo que el buscador heurístico típico realizará su análisis de los archivos ejecutables en dos fases:

1. Primero deberá distinguir donde es posible que se encuentre el virus, ya que el archivo puede ser grande y el aplicar esta técnica le llevaría mayor tiempo al antivirus en realizar un diagnóstico. El primer paso es acercarse al código que pueda ser el virus, para ello se debe estudiar las diferentes formas en que se presenta el fenómeno y bajo qué condiciones.

Una vez que el analizador heurístico ha identificado el área probable de la infección, se realiza un análisis lógico del programa contenido en esta región, para determinar las instrucciones que pueden contener. Esto es un problema extremadamente complejo ya que existen muchas maneras de escribir un programa, para esto se observa el Código B-10:

Ejemplo 1

Lenguaje máquina (en hexadecimal)	Instrucciones
B8 00 4C	MOV AX, 4C00
CD 21	INT 21

Código B-1 Programa en lenguaje ensamblador para regresar al sistema operativo

Ejemplo 2	
Lenguaje máquina (en hexadecimal)	Instrucciones
B4 3C	MOV AH, 3C
BB 00 00	MOV BX, 0000
88 D8	MOV AL, BL
80 C4 10	ADD AH, 10
8E C3	MOV ES, BX
9C	PUSHF
26	ES:
FF 1E 84 00	CALL FAR [0084]

Código B-2 En lenguaje ensamblador para regresar al sistema operativo

Los dos ejemplos anteriores realizan la misma función: La de finalizar un programa y el regreso al ambiente MS-DOS, sin embargo, si se observa la sucesión de los bytes de código de máquina que componen cada conjunto de instrucciones, son totalmente diferentes. La primera sucesión de código utiliza un llamado al sistema operativo mediante una interrupción sencilla. La segunda sucesión es más complicada, ya que se realizan algunas operaciones antes y después de la misma petición. Estas son de una forma más directa, mediante la llamada dirección 84h de la tabla de direcciones.

Ejemplo1: B8 00 4C CD 21

Ejemplo2: B4 3C BB 00 00 88 D8 80 C4 10 8E C3 9C 26 FF 1E 84 00

Por lo anteriormente mostrado, se observa que existe un número infinito de maneras en las que se escribe un programa; parece ser una tarea difícil para extraer todos los patrones posibles. Afortunadamente la mayoría de los virus usan rutinas parecidas para tener acceso directo al sistema operativo. Además de la técnica heurística, se supone que se debe discernir entre los dos tipos de conducta. Este problema se resuelve con una heurística dinámica que se comentará más delante.

A.1.1 Buscador heurístico estático y dinámico.

El buscador heurístico estático reconoce varias conductas del programa, este tipo de buscador utiliza una base de datos extensa comparada con la heurística dinámica. En esta base de datos se asocia cada sucesión de bytes de la conducta observada en el programa; sin olvidar que se puede utilizar comodines para realizar una búsqueda más simple.

No	Secuencia de bytes	Operación realizada
1.	B8 ?? 4C CD 21	Finaliza un programa de la forma 1
2.	B4 4C CD 21	Finaliza un programa de la forma 2
3.	B4 4C B0 ?? CD 21	Finaliza un programa de la forma 3
4.	B0 ?? B4 4C CD 21	Finaliza un programa de la forma 4
...		
100.	B8 02 3D BA ?? ?? CD 21	Abre un archivo de la forma 1
101.	BA ?? ?? B8 02 3D CD 21	Abre un archivo de la forma 2

Tabla B-1 Códigos heurísticos.

Si se tiene un programa con la siguiente secuencia:

Secuencia: B4 09 BA 20 01 CD 21 **B8 02 3D BA 12 34 CD 21** CC **B8 FF 4C CD 21**

↑ Firma 100 ↑ Firma 1

Patrones encontrados:

Patrón 100: B8 02 3D BA ?? ?? CD 21 ⇒ Se abre un archivo

Patrón 001: B8 ?? 4C CD 21 ⇒ Termina de está manera

Las firmas o sucesiones de bytes mostradas con anterioridad sirven para identificar una conducta; esto no significa que el programa tenga un virus, únicamente indica que el programa tiene esos patrones.

A principios de los años 90, se presentaron los virus polimórficos [73], los cuales hacían que se incrementara el problema al momento de buscar patrones; debido a que los buscadores heurísticos se basaban en buscar patrones sencillos. Para resolver este problema se toma el concepto de "emulación", el cual sería una herramienta poderosa también conocida como Sandbox. La idea básica es que un programa cree ejecutarse en un sistema verdadero, sin embargo, éste se ejecuta en una máquina virtual lo que provoca que muestre sus conductas, que posteriormente serán clasificadas por un buscador dinámico. El buscador dinámico heurístico puede detectar las llamadas al sistema operativo y así realizar un diagnóstico apropiado.

Sin embargo esta técnica presenta las siguientes deficiencias:

- El crear una máquina virtual completa, implica un costo elevado, es por este motivo que la mayoría de los antivirus, solamente simulan algunas instrucciones del microprocesador, con esto, los creadores de virus, tienen que recurrir a funciones no documentadas o instrucciones del microprocesador matemático. Como el simulador no encuentra dicha instrucción dentro de su base de datos supone que todo se encuentra bien.

Un ejemplo de un virus que engaña a un buscador dinámico:

1. Si la hora actual no es constante se salta a la instrucción 3.
2. Termina.
3. Infectar un programa nuevo.
4. ...

Alguna de estas técnicas se mejoraron al almacenar las banderas de actividad que tiene un programa. En el instante en que el emulador se encuentra de nuevo con el archivo, se realiza una segunda emulación y si éste presenta una actividad diferente se envía un mensaje al antivirus. No se debe olvidar que existe otra manera de detectar el momento en que un archivo ha sido cambiado, y es mediante una suma de validación. Por lo tanto, es más seguro este método, además que el costo de cálculo es mínimo [58].

Otras formas de realizar un análisis heurístico de comportamiento son las realizadas por IBM que utilizan una red neuronal para analizar sectores de arranque y código de programas.Por ejemplo, Symantec con Bloodhound utiliza un sistema experto para analizar conductas diferentes.

Si los buscadores se basan en localizar patrones repetidos, entonces ¿qué sucede con los virus que se escriben en lenguajes que tienen secuencias repetidas de bytes, como Pascal, C y Basic? Dado que el enlazador incrusta cierto código, hace que los virus sean más parecidos en su apariencia, lo que origina diagnósticos falsos, además de complicar la creación de una firma.

Referencias

[1] JhonVon Neumann and A.W.Burks, "Theory of Self-Reproducing Automata," *University of Illinois Press*, 1996.

[2] A.K.Dewdney, "Of worms, viruses and Core War," *Scientific American*, 1989.

[3] Alan Solomon, "A Brief History of PC Viruses," *S&S International*, 1993.

[4] A.K.Dewdney. In the game called Core War hostile programs engage in a battle of bits. Scientific American, 14-22. 1984. Scientific American. Ref Type: Magazine Article

[5] David M.Chess, "Some Common PC-DOS Viruses and What They Mean To You," *High Integrity Computing Laboratory*, 1991.

[6] A.K.Dewdney, "A Core War bestiary of viruses, worms and other threats to computer memories," *Scientific American*, pp. 14-23, 1985.

[7] John F.Shoch and Jon A.Hupp, "The "Worm" Program Early Experience with a Distributed Computation," *ACM*, vol. 25 pp. 172-180, 1982.

[8] Lundell, A., "VIRUS! The Secret World of Computer Invaders that Breed and *Destroy*," *Contemporary Books*, 1989.

[9] Anne E.Webster, "University of Delaware and the Pakistani computer virus," *Computers & Security*, vol. 8 pp. 103-105, 1989.

[10] Vesselin Bontchev, "The Bulgarian and Soviet Virus Factories," *Laboratory of Computer Virology*, 1991.

[11] Vesselin Bontchev, "Macro Virus Identification Problems," *Int.Virus Bull*, pp. 175-196, 1997.

[12] Maria M.Pozzo and Terence E.Gray, "An approach to containing computer viruses," *Computers & Security*, vol. 6 pp. 321-331, 1987.

[13] Sun MicroSystem. Strange Brew Overview. Sun MicroSystem . 1999. Ref Type: Electronic Citation

[14] Frederick B.Cohen, "Computer virus basics.," *A SHORT COURSE ON COMPUTER VIRUSES* Second ed. JohnWiley & Sons, Inc., 1994, pp. 1-28.

[15] JAHN I, R.LÖTHER, and K.SENGLAUB, "Historia de la Biología," *VEB Gustav Fischer Verlag*, vol. 1 1990.

[16] BROCK, T. D., "Milestones in Microbiology," *American Society for Microbiology*, 1961.

[17] Fred Cohen, "Computer viruses, theory and experiments," *Computers & Security*, vol. 6 pp. 22-35, 19987.

[18] Lawrence M.Bridwell and Peter S.Tippett, "ICSA Labs 6th Annual Compuer Virus Prevalence Survey 2000," *ICSA Journal*, vol. 2 pp. 1-61, 2000.

[19] George I.Davida, "Models of Viral Propagation," *VIRUS*, 2000.

[20] Fred Cohen, "Computer Viruses - Theory and Experiments," *Computers and Security*, vol. IFIP-SEC v6#1 1987.

[21] Catherine L.Young, "Taxonomy of computer virus defense mechanisms," *Proc.10th National Computer Security Conference*, pp. 220-225, 1987.

[22] Frederick B.Cohen, "Models of Practical Defenses Against Computer Viruses," *Electrical and Computer Engineering Departament Univerity of Cincinnati*, 1984.

[23] Vesselin Bontchev, "Possible Virus Attacks Against Integrity Programs and How to Prevent Them," *Virus Bull*, pp. 131-141, 1992.

[24] Jeffrey O.Kephart and William C.Arnold, "Automatic Extraction of Computer Virus Signatures," *Virus Bulletin International Conference*, pp. 178-184, 1994.

[25] Kleinbaum DG, Kupper LL, and Morgenstern H., "Principles and quantitative methods. Belmont: Lifetime Learning," *Epidemiologic research.*, 1982.

[26] Skrabanek P., "The poverty of epidemiology.," *Perspect Biol Med*, pp. 180-182, 1992.

[27] W.H.Murray, "The application of epidemiology to computer viruses," *Computers & Security*, vol. 7 pp. 130-150, 1998.

[28] Winfried Gleissner, "A mathematical theory for the spread of computer viruses," *Computers & Security*, vol. 8 pp. 35-41, 1989.

[29] Alan Solomon. Epidemiology and computer viruses. 1990.Ref Type: Report

[30] H.Trottier, M. S. and P.Philippe, "Deterministic Modeling Of Infectious Diseases: Theory And Methods, " *The Internet Journal of Infectious DiseasesTM ISSN 1528-8366*, 2001.

[31] Daniel Bernoulli. Essai d'une nouvelle analyse de la mortalité causée par la petite vérole et des avantages de l'inoculation pour la prévenir. *Mém. Math. Phys. Acad. Roy. Sci. Paris*, 1-45. 1760. Ref Type: Report

[32] Norman T.J.Bailey, "The mathematical theory of infectious diseases and its applications," second edition ed. Oxford University Press, 1975.

[33] Jeffrey O.Kephart, David M.Chess, and Steve R.White, "Computers and Epidemiology," *Published in the IEEE SPECTRUM*, vol. 1 1993.

[34] Jeffrey O.Kephart and Steve R.White, "Directed-Graph Epidemiological Models of Computer VirusesDirected-Graph Epidemiological Models of Computer Viruses," *High Integrity Computing Laboratory IBM Thomas J.Watson Research Center*, pp. 343-359, 1991.

[35] Susan J.Harrington, "The impact of codes of ethics on information system personnel," *ACM*, 1994.

[36] Robert M.Slade, "Antiviral Checklist," *Int.Virus Bull*, 1992.

[37] Steve R.White. Open Problems in Computer Virus Research. 1998. Ref Type: Generic

[38] P Bretscher and Cohn, "A theory of self-nonself discrimination. Science," *Ann Rev Public Health*, 1970.

[39] Mark A.Bedau, n S.McCaskill, Norman H.Packard, Steen Rasmussen¤¤, Chris Adami, David G.Green, Takashi Ikegami, Kunihiko Kaneko, and Thomas S.Ray. Open Problems in Artificial Life. School of Environmental and Information Science, Charles Stuart University. 2000. Ref Type: Abstract

[40] Polly Matzinger, "Tolerance, danger, and the extended family," *Annu.Rev.Immunol.*, 1994.

[41] Patrik D'haeseleer, "An Immunological Approach to Change Detection: Theoretical Results," *University of New Mexico*, 1996.

[42] Anil Samayaji, Steven Hofmeyr, and Stephanie Forrest, "Principles of a Computer Immune System," *Departament of Computer Science*, 1997.

[43] Stephanie Forrest, Alan S.Perelson, Lawrence Allen, and Rajesh Cherukuri, "Self-Nonself Discrimination in a Computer," *IEEE Symposium on Research in Security and Privacy*, 1994.

[44] Stephanie Forrest, Steven Hofmeyr, and Anil Samayaji, "Compuer Immulogy," *ACM*, 1996.

[45] Polly Matzinger, "The real function of the immune system or tolerance and the four d's (danger, death, destruction and distress)," *American Society for Microbiology*, 1996.

[46] Jr.Janeway CA, "Approaching the Asymptote? Evolution and Revolution in Immunology.," *Cold.Spring.Harb.Symp.Quant.Biol*, 1989.

[47] Derick Wood, *Theory of Computation*, 1era ed. Estados Unidos: 1987, pp. 472-489.

[48] Thomas A.Sudkamp, *Languages and Machines*, Second Edition ed. Addison Wesley, 1997.

[49] J.Glenn Brookshear, *Teoría de la computación Lenguajes formales, autómatas y complejidad.*, 1 ed. Máxico: ADDISON WESLEY IBEROAMERICANA S.A., 1999.

[50] Pierre Boullier. Dynamic grammars & semantic analysis. INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE **2322**. 1994. Ref Type: Report

[51] David Lacey, Neil D.Jones, Eric Van Wyk, and Carl Christian Frederiksen, "Proving Correctness of Compiler Optimizations by Temporal Logic," *ACM*, 2002.

[52] Sundkamp, T. A., "Languages and machines: an introduction to the theory of computer," in Addison Wesley Longman (ed.) *an introduction to the theory of computer* Second ed. USA: 1997.

[53] Steven S.Muchnick, *Program Flow analysis: Theory and aplications*, 1 ed. Englewwod Cliffs: University of Kansas, 1981.

[54] James S.Uhl and R.Nigel Horspool, "Flow grammars a flow analysis methodology," *Dept.of Computer Science, University of Victoria*, 1994.

[55] Dorothy Elizabeth Robling Denning, "Cryptography and Data Security," in EDDISON WESLEY (ed.) *Cryptography and Data Security* 1982.

[56] Fred Cohen, "A Cryptographic Checksum for Integrity Protection," *Computers & Security*, vol. 6 pp. 505-510, 1987.

[57] John Kodis, "Fletcher's checksum," *ACM*, 1992.

[58] Yisrael Radai, "Integrity Checking for Anti-Viral Purposes Theory and Practice," *Hebrew University of Jerusalem*, pp. 1-56, 1994.

[59] Doug Varney, "Adequacy of Checksum Algorithmos for Computer Virus Detection," *ACM*, 1990.

[60] Dawkins R, *Viruses of the Mind*, Dennett and his Critics ed. Blackwell Publishers, 1993, pp. 13-27.

[61] Dr.Harold Joseph Highland, "The BRAIN virus: fact and fantasy," *Computers & Security*, vol. 7 pp. 367-370, 1998.

[62] R.Rivest, A.Shamir, and L.Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," *CACM*, vol. 21 pp. 120-126, 1978.

[63] Dimitri Bertsekas and Robert Gallager, "Data Networks," *Data Networks* Second ed. Prentice Hall, 1992.

[64] Georgia Griffiths and G.Carlyle Stones, "The Tea-Leaf Reader Algorithm: An Efficient Implemenation of CRC-16 and CRC-32," *ACM*, vol. 30 pp. 617-620, 1987.

[65] Mark Grand, *Patterns Java*, 1 ed. USA: 1998, pp. 1-458.

[66] John McLean, "The Specification and Modeling of Computer Security," *Center for High Assurance Computer Systems*, 1990.

[67] José M.Garrido Artech Tlouse, *Practical Process Simulation* Boston London: 1998.

[68] Paul A.Fishwick, *Simulation Model Design and Execution* United States of America: 1995.

[69] Dr.Frederick Kuhl Dr.Richard Wea Therly Dr.Judith Dahmann, *Creating Computer Simulation Systems* 1999.

[70] Erick Gamma, Richard Helm, Ralph Johnson, and John Vissides, *Design patterns elements of resusable object oriented software*, 1a ed. 1994, pp. 1-367.

[71] Cay S.Horstmann and Cary Cornell, *Core Java* 2000.

[72] M.W.Eichin and J.A.Rochlis, "With microscope and tweezers: An analysis of the Internet virus of November 1988," *Proc.1989 IEEE Symp.on Security and Privacy*, pp. 326-343, 1989.

[73] Symantec, "Understanding and Managing Polymorphic Viruses.," *The Symantec Enterprise Papers*, 1996.

Glosario

Anticuerpo	Una sustancia química producida por el sistema inmune como respuesta a las bacterias, virus u otras substancias posiblemente dañinas que ayuda a eliminarlas del cuerpo.
Células CD4	Las células blancas especiales de la sangre que coordinan la respuesta inmune para luchar contra las infecciones bacteriales y virales.
Complejo mayor de histocompatibilidad (MHC)	Conjunto de genes que codifican las moléculas del MHC. Estas son proteínas de membrana que tienen asociados péptidos generados en el interior de la célula. El complejo MHC-péptido constituye el blanco de reconocimiento de las células T. El MHC es altamente polimorfo, y es la causa principal del rechazo de injertos.
Epidemia	Aumento del número de virus en una población.
Epidemiología	Ciencia que estudia la enfermedad de una población, su frecuencia, evolución y distribución.
Estabilidad antigénica	Es la probabilidad que el genoma que gobierna la estructura antigénica de un agente cambie, “mutación”.
Estado de Inmunidad	El huésped no es susceptible a la infección o enfermedad.
Foco	Lugar de la aparición de los casos de una enfermedad.
Fracción Inmunogénica	Elemento de acción de la defensa inmune creada por el huésped. Indica si actúa frente al agente.
Hiperendémica	Enfermedad endémica que afecta a una importante parte de la población.
Infección liso génica	Es la integración del ADN del virus con el ADEN del huésped.
Infección lítica	Es la fijación de los virus en la superficie de la célula huésped.
inmunidad	Estado en el que un huésped (susceptible) es resistente a padecer una determinada enfermedad o proceso infeccioso.
inmunización	Acción por medio de la cual se induce o aumenta la resistencia frente a una enfermedad infecciosa, habitualmente mediante la vacunación.
Inmunoglobulina	Anticuerpo.
Latencia	Es cuando la infección se realiza, pero no se muestra los síntomas.

Linfocito T citotóxico	Célula T con capacidad de eliminar otras células y que juega un papel central en la eliminación de patógenos. La mayoría son linfocitos T CD8+, aunque existen también linfocitos T CD4+ que pueden destruir células en algunos casos.
Linfocito T colaborador	Célula T CD4+ efectora, referida también como TH2, que en respuesta al reconocimiento del antígeno asociado a las moléculas de MHC de una célula B produce en esta una serie de estímulos que resultan en una producción de anticuerpos más eficiente.
Linfocito T inflamatorio	Célula T CD4+ efectora, referida también como TH1, que tienen como función principal la activación de macrófagos.
Macrófago	Tipo de glóbulo blanco que ayuda en la lucha del cuerpo contra bacterias e infecciones aislando y destruyendo organismos invasores.
Mutar	La susceptibilidad de mutar algo, es la que posibilita el proceso evolutivo de las formas “vivientes”; y se encuentra en los virus bajo determinadas condiciones; esto se debe por influencias de distintos factores o por manipulación genética.
Pandemia	Epidemia que afecta a una gran parte de la población y que posee una amplia difusión espacial.
Periodo de incubación	Tiempo que abarca desde que el huésped sufre la infección hasta que aparecen los primeros síntomas.
Población	Objetos que se encuentran dentro de un ámbito y son objetos de estudio.
Población en riesgo	Es la población susceptible a una infección en condiciones dadas.
Portador	Ser vivo que contiene un agente patógeno en su organismo en ausencia de enfermedad clínica. y que puede actuar como fuente de infección en un momento dado.
Receptor T (TCR)	Receptor de membrana de los linfocitos T que reconoce en forma específica péptidos asociados a receptores de membrana (moléculas del MHC) de otras células propias.
Recrudescencia	La reaparición de una enfermedad en un huésped cuya infección era del tipo persistente.
Resistencia	Es el mecanismo de defensa natural que tiene el huésped para defenderse del agente y evitar así la aparición de la enfermedad.

Sinergia	El término alude a un mecanismo de exaltación o potenciación del poder patógeno que ejerce un virus sobre otro diferente cuando están juntos; dicho de otra forma el poder de ambos es mayor que la suma de los poderes individuales.
Sistema inmune	El grupo complejo de órganos y células que defienden el cuerpo contra la infección y las enfermedades
Umbral epidemiológico	Es la incidencia de un proceso epidemiológico que se encuentra en curso.
Vacuna	Sustancia que sé induce de una forma artificial para generar la respuesta inmune frente a un agente patógeno.
Vehículo	Son objetos que actuando como intermediario para la transmisión del agente entre seres vivos.
Vigilancia activa	Es una vigilancia epidemiológica en que la información es recolectada de forma dinámica.
Vigilancia epidemiológica	Colección sistemática de información y análisis de la misma, para desarrollar estrategias de defensa.
XML	eXtensible Markup Language. Lenguaje de marcado extensible.
XML-DOM	Objeto COM que se utiliza para la manipulación de documentos XML en diversas aplicaciones.