9

Facultad de Ingeniería 2EJ

Virus informáticos: causas, efectos y soluciones

Ingeniero en Computación.
Héctor Martín Badillo Rojas
José Ramón Gallardo Hernández
Víctor Hugo Bustamante Vallín

1995

Finalmente aquí esta el trabajo que concluye mis estudios profesionales, quiero agradecer sinceramente a todas las personas que colaboraron conmigo para lograr concluirlo, y lo dedico especialmente:

A mis papás Ma. de Jesús Vallín y a Margarito Bustamante, por la vida, el cariño recibido, el apoyo de siempre y por tanta insistencia.

A mis hermanos Gemma, Ruth, Marco y Odette, por convivir conmigo, aguantarme y apoyarme cuando lo necesité.

A Yolanda, por inspirar y llenar mi vida con su amor.

V. Hugo B. V.

Nunca es tarde para empezar... y para terminar lo comenzado.

A mis padres, que con su esfuerzo pude tener una carrera profesional. A través de este trabajo, quiero expresarles mi profundo agradecimiento, respeto y amor.

A mis hermanos, quienes me motivaron por su dedicación y tenacidad.

Con todo mi amor a mi esposa, quien todo este tiempo estuvo a mi lado apoyándome, hasta en los momentos más difíciles. Mil gracias.

A todos los que de una u otra forma colaboraron en el enriquecimiento de este documento.

Es indescriptible la satisfacción del deber cumplido.

José Ramón

A mis padres, Joaquín y Ana por el apoyo que me brindaron siempre para la finalización de una etapa tan importante de mi vida.

A mi esposa Olimpia por permitirme compartir lo mejor de su vida conmigo.

A mis hijos Héctor y Olimpia por ser parte fundamental de mi existencia.

A mis hermanas Aida y Araceli por esa unión tan especial que siempre hemos mantenido.

Héctor

Virus Informáticos: causas, efectos y soluciones

Héctor Martín Badillo Rojas
José Ramón Gallardo Hernández
Víctor Hugo Bustamante Vallín

Contenido

Introducción *1*

Los sistemas de cómputo actuales se caracterizan por manejar grandes cantidades de datos, los cuales en la mayoría de los casos representan la información valiosa de una empresa o a la empresa misma.

Hasta hace algunos años se concebía a los sistemas de cómputo como elementos con un grado de seguridad altamente confiable. Desgraciadamente se han desarrollado programas que atacan las deficiencias de diseño de los sistemas operativos y se enfocan a violar sus mecanismos de protección. Este tipo de desarrollos se han convertido en un serio problema de seguridad y representan una amenaza para conservar la información en las computadoras y redes de computadoras.

En la actividad diaria de la comunidad informática son muchas las ocasiones en las que se plantea la siguiente pregunta: ¿por qué es tan difícil encontrar una solución al problema de los virus? Muchas pueden ser las respuestas: porque faltan conocimientos en procedimientos de seguridad informática, porque existen fallas u omisiones en los diseños de las aplicaciones, porque siempre existen individuos con el afán de venganza o simplemente por el deseo de satisfacer un reto personal desarrollando programas que alteren el funcionamiento de los sistemas.

¿Qué tan serio es el problema?, las medidas de seguridad tradicionales nunca fueron pensadas para evitar que programas de aplicación fueran creados con el objeto de violarlas.

El fenómeno de los virus sorprendió a la mayor parte de los usuarios de computadoras. Los métodos tradicionales que se utilizan para proteger la información son por lo general poco efectivos contra los programas nocivos y la falta de conocimiento en informática crea mitos

acerca de este problema, que afectan negativamente los esfuerzos para mantener la integridad de la información. Un ejemplo importante es la dependencia en los respaldos como línea frontal de defensa contra los virus. El respaldo de un sistema infectado también puede infectarse debido a que los virus permanecen ocultos por meses o incluso por años antes de activarse y causar daños. Adicionalmente, nuevos virus atacan los programas estándar de respaldo y restauración de modo que la información contenida en los elementos de respaldo se dañan irremediablemente. Los sistemas existentes de seguridad también son, generalmente, poco efectivos contra los virus. La mayor parte de los sistemas fueron diseñados para permitir el acceso sólo a los individuos confiables. Sin embargo los virus son invisibles al ojo humano y casi siempre son diseminados inconscientemente por manos amistosas. Incluso los sistemas de seguridad pueden infectarse y propagar la infección a través de los canales normalmente restringidos, por ejemplo, programas que piden contraseñas de acceso (passwords).

El presente trabajo de investigación es un esfuerzo por presentar los elementos que conforman el problema de los virus informáticos, para que se conozcan sus alcances y limitaciones. Así mismo, se pretende mostrar la estructura y funcionamiento de estas aplicaciones, y con base a ello proponer posibles soluciones.

Estudiar el problema de los virus plantea un compromiso que no se puede tomar como un todo a la ligera. El planteamiento de este estudio tiene en realidad dos partes principales. El estado de los sistemas de información y su relación con los virus y sus orígenes está contemplado en los capítulos 2, 3 y 4. Estos capítulos describen los conceptos clave que están detrás de este fenómeno.

El capítulo 2 es una investigación sobre los orígenes y causas de la aparición de los virus en el contexto informático. Con base a la información obtenida se hace una reconstrucción histórica de la evolución de los virus. También, se hace una clasificación de las personas que desarrollan programas nocivos. En el capítulo 3 se analizan los efectos propiciados por la aparición de los virus informáticos. Dentro del capítulo 4 se describen los elementos fundamentales de seguridad relacionados a los sistemas de cómputo.

La segunda parte que está compuesta por los capítulos 5 al 10, es un análisis sobre las distintas formas en que los virus y otros agentes nocivos se presentan, así como su estructura, funcionamiento y los mecanismos que se desarrollan para reducir sus efectos.

Existen diferentes clasificaciones que se han dado para los virus informáticos; en el capítulo 5 se presenta una visión propia. En el capítulo 6 se contempla el análisis y diagnóstico de programas nocivos, mostrando un estudio de su estructura y funcionamiento. No dejan de ser importantes las defensas y sistemas de protección actuales; en el capítulo 7 se presenta un panorama general de éstas. El capítulo 8 es una propuesta para combatir de manera estructurada y a nivel corporativo, la proliferación de virus informáticos. Dentro del capítulo 9 se presenta un estudio de algunos casos reales sobre virus informáticos. Finalmente, el capítulo 10 se propone una metodología para aquellos que desean elaborar programas antivirus.

Orígenes y causas 2

Orígenes

No existe ninguna información precisa que permita reconstruir los orígenes de los virus. La razón principal de esto es que muchas instituciones, las grandes empresas y los organismos gubernamentales científicos o militares ocultan la realidad respecto a este fenómeno, para no reconocer la vulnerabilidad de sus sistemas de cómputo. Tomando como base diversos documentos, se ha podido establecer una liga lógica de los orígenes y causas de los virus informáticos. Cabe reconocer que en algunos casos, los hechos que se presentan son las conclusiones obtenidas del análisis de la información.

Han sido mencionados los virus de las computadoras como un problema informático que surgió en la década de los ochentas, pero la idea de un programa que trabajara como lo que ahora conocemos como virus computacional nació en los primeros días de la era de la computadora electrónica.

En 1949 el modelo de un programa que funcionaba como un virus fue presentado por John von Newmann, uno de los fundadores de la Comunidad de la Computación. En su artículo titulado "Theory and Organization of Complicated Automata", von Neumann mostró teóricamente que los programas de computadora pueden autorreproducirse, lo cual no dejó convencidos a algunos de sus colegas. Esto es comprensible pues la primera computadora electrónica se desarrolló diez años después.

A finales de los cincuentas, un grupo de científicos interesados y convencidos de la teoría de von Newmann le dieron vida a ésta. En los laboratorios Bell de la AT&T, H. Douglas McIlroy, Victor Vysottsky y Robert Morris, desarrollaron un programa llamado Creeper cuya única función era reproducirse. Para contrarrestar los efectos de este programa fue diseñado Reaper, cuya misión era la de buscar y destruir a Creeper; una vez que ya no encontraba copias de Creeper, Reaper se autodestruia.

Basándose en el funcionamiento de Creeper, McIlroy propuso un juego llamado Darwin, el cual era una batalla entre varios programas. Cada jugador desarrollaba un programa que llamaban "organismo" y éste tenía la capacidad de autocopiarse en la memoria. Al inicio del juego, cada jugador dejaba corriendo su programa en cualquier lugar de la memoria e intentaba destruir los "organismos" del oponente sobrescribiendo su código en el área ocupada por éstos. El jugador que ha obtenido la mayor cantidad de "organismos" al finalizar el juego era el ganador. McIlroy inventó además, un "organismo" que no se podía destruir, con el cual sólo ganó pocos juegos, porque aparentemente no era muy eficiente en su reproducción.

Al concepto de programas que se duplican y se destruyen unos a otros en la memoria de la computadora se le ha llamado Core Wars. Cuando Core Wars era jugado en una computadora el daño era mínimo, pues sólo se perdían las aplicaciones que estaban corriendo en ese momento y bastaba con apagar la máquina para cancelar su deterioro, pero el efecto de un juego divertido dejó ver las consecuencias peligrosas que podía acarrear.

A finales de los cincuentas, un grupo de científicos interesados y convencidos de la teoría de von Newmann le dieron vida a ésta. En los laboratorios Bell de la AT&T, H. Douglas McIlroy, Victor Vysottsky y Robert Morris, desarrollaron un programa llamado Creeper cuya única función era reproducirse. Para contrarrestar los efectos de este programa fue diseñado Reaper, cuya misión era la de buscar y destruir a Creeper; una vez que ya no encontraba copias de Creeper, Reaper se autodestruia.

Basándose en el funcionamiento de Creeper, McIlroy propuso un juego llamado Darwin, el cual era una batalla entre varios programas. Cada jugador desarrollaba un programa que llamaban "organismo" y éste tenía la capacidad de autocopiarse en la memoria. Al inicio del juego, cada jugador dejaba corriendo su programa en cualquier lugar de la memoria e intentaba destruir los "organismos" del oponente sobrescribiendo su código en el área ocupada por éstos. El jugador que ha obtenido la mayor cantidad de "organismos" al finalizar el juego era el ganador. McIlroy inventó además, un "organismo" que no se podía destruir, con el cual sólo ganó pocos juegos, porque aparentemente no era muy eficiente en su reproducción.

Al concepto de programas que se duplican y se destruyen unos a otros en la memoria de la computadora se le ha llamado Core Wars. Cuando Core Wars era jugado en una computadora el daño era mínimo, pues sólo se perdían las aplicaciones que estaban corriendo en ese momento y bastaba con apagar la máquina para cancelar su deterioro, pero el efecto de un juego divertido dejó ver las consecuencias peligrosas que podía acarrear.

Para 1965, la idea de este juego se extendió a otros centros de investigación como el Instituto Tecnológico de Massachusetts y el Centro de Investigación de Xerox en Palo Alto California. En este último fue desarrollado Worm, un programa diseñado por John F. Shoch. El propósito de Worm era hacer el mayor uso posible de una red experimental de computadoras que utilizaba un medio ambiente distribuido (Xerox fue quien desarrolló el protocolo Ethernet), para lograrlo se copiaba y se ejecutaba de una máquina a otra. Esta compañía hizo públicos sus experimentos hasta 1982.

De la misma manera, la idea de programas que se autorreproducen fue guardada en secreto por sus desarrolladores hasta 1983, cuando Ken Thompson (creador de la versión original de UNIX), lo dio a conocer. Thompson aceptó hablar en una conferencia para la Asociación para la Maquinaria de la Computación, en la cual explicaba el funcionamiento de programas que se regeneran.

En noviembre de 1983, Fred Cohen define formalmente el término "virus de computadora". En aquel tiempo, Cohen era un estudiante de doctorado en la Universidad del Sur de California que asistía a un seminario de seguridad, coordinado por el profesor Len Adleman. En una presentación él mostró a sus compañeros el prototipo de cinco virus que corrían en una VAX 11/750 con el sistema operativo UNIX. Cada prototipo obtenía el completo control del sistema en menos de una hora. Así mismo, mostró que resultados similares podrían ser obtenidos en equipos IBM VM/370 y máquinas DIGITAL con sistema operativo VMS. El profesor Len Adleman, le sugirió que le llamara a su creación "virus de computadora". Las investigaciones del ahora Dr. Cohen están enfocadas en gran parte a los virus en las computadoras.

Para 1965, la idea de este juego se extendió a otros centros de investigación como el Instituto Tecnológico de Massachusetts y el Centro de Investigación de Xerox en Palo Alto California. En este último fue desarrollado Worm, un programa diseñado por John F. Shoch. El propósito de Worm era hacer el mayor uso posible de una red experimental de computadoras que utilizaba un medio ambiente distribuido (Xerox fue quien desarrolló el protocolo Ethernet), para lograrlo se copiaba y se ejecutaba de una máquina a otra. Esta compañía hizo públicos sus experimentos hasta 1982.

De la misma manera, la idea de programas que se autorreproducen fue guardada en secreto por sus desarrolladores hasta 1983, cuando Ken Thompson (creador de la versión original de UNIX), lo dio a conocer. Thompson aceptó hablar en una conferencia para la Asociación para la Maquinaria de la Computación, en la cual explicaba el funcionamiento de programas que se regeneran.

En noviembre de 1983, Fred Cohen define formalmente el término "virus de computadora". En aquel tiempo, Cohen era un estudiante de doctorado en la Universidad del Sur de California que asistía a un seminario de seguridad, coordinado por el profesor Len Adleman. En una presentación él mostró a sus compañeros el prototipo de cinco virus que corrían en una VAX 11/750 con el sistema operativo UNIX. Cada prototipo obtenía el completo control del sistema en menos de una hora. Así mismo, mostró que resultados similares podrían ser obtenidos en equipos IBM VM/370 y máquinas DIGITAL con sistema operativo VMS. El profesor Len Adleman, le sugirió que le llamara a su creación "virus de computadora". Las investigaciones del ahora Dr. Cohen están enfocadas en gran parte a los virus en las computadoras.

En mayo de 1984, la revista Scientific American publicó un artículo de A. K. Dewdney que describía un Core Wars desarrollado por él, y ofrecía a los lectores la oportunidad de enviarles el código del mismo por sólo dos dólares.

En forma simultánea a los eventos mencionados, las primeras manifestaciones de virus se dieron a principios de los años setentas. Fueron desarrolladas específicamente para mainframes, redes de computadoras y para la familia de las computadoras Apple II.

Rabbit fue reportado en el año de 1974; corría en computadoras IBM 360 con el sistema operativo ASP. Era un programa que tenía la capacidad de copiarse a sí mismo y mandar original y copia nuevamente a la cola de procesos para ejecución. Como consecuencia provocaba que el sistema se hiciera más lento y se saturara la memoria.

En la UNIVAC 1108 había un programa con características similares a los virus actuales. Animal es up juego que ha sido escrito para muchos tipos de máquinas. El juego consistía en veinte preguntas básicas para adivinar el nombre de un animal que el usuario pensaba, y podía recordar cada animal que adivinaba. Mientras el usuario jugaba, el programa se copiaba a sí mismo dentro de cada directorio. Animal era inteligente porque revisaba si el archivo ya tenía una copia del mismo, y si contenía una versión vieja se actualizaba. Después de permanecer en el sistema cierto tiempo, en vez de reproducirse, Animal jugaba un último juego diciendo al usuario "good bye" y entonces se autodestruía.

En octubre de 1980, se distribuían mensajes de estado en la red ARPAnet (Red de la Agencia de Proyectos de Investigación Avanzada del Departamento de la Defensa de Estados

Unidos). Esos mensajes fueron pensados para ser borrados inmediatamente por un recolector de basura llamado Redux. Pero en algún lugar cerca de Los Angeles California, el código de Redux fue modificado de tal forma que los mensajes se enviaban pero nunca se borraban. Los sistemas se saturaban y finalmente se caían. El tiempo en que la red estuvo fuera de servicio fue de casi tres días y hasta mediados de los ochentas recuperó completamente su velocidad normal de funcionamiento.

Para los años de 1981 y 1982, un virus se hizo presente en las máquinas Apple II, se conoció como Elk Cloner y desplegaba el siguiente texto:

ELK CLONER:
THE PROGRAM WITH A PERSONALITY
IT WILL GET ON ALL YOUR DISKS
IT WILL INFILTRATE YOUR CHIPS
YES, IT'S CLONER
IT WILL STICK TO YOU LIKE GLUE
IT WILL MODIFY RAM, TOO
SEND IN THE CLONER!

Este virus era muy eficiente en la autorreproducción, pero se podía evitar poniendo etiquetas de protección a los discos. Este programa se insertaba dentro del sistema operativo en los comandos RUN, LOAD, BLOAD y CATALOG; y podía detectar si ya se había copiado.

Los primeros virus que causaron infecciones y daños de consideración aparecieron en 1986 en Estados Unidos de Norteamérica. El virus Brain tiene la distinción de ser el primer virus para computadoras personales con sistema operativo DOS, que hace su aparición en los Estados Unidos, proveniente de un laboratorio de prueba. De acuerdo al Centro de Cómputo Académico de la Universidad de Delaware, fue reportado el 22 de octubre de 1987. Asimismo, el periódico JournalBulletin de Richmond, perdió valiosa información a causa de Brain, que se identificaba con copyright y el nombre, dirección y teléfonos de una compañía llamada BRAIN en Lahore, Pakistán.

En diciembre de 1987, la compañía IBM se vio obligada a diseñar un programa antivirus para desinfectar su sistema de correo interno, pues éste fue contagiado por un virus que hacía aparecer en las pantallas de las computadoras en la red un mensaje navideño. El virus presentaba un mensaje con un árbol al lado, y pedía al usuario que tecleara la palabra "CHRISTMAS". Si se tecleaba la palabra, el virus se introducía en la lista de distribución del correo electrónico y se seguía diseminando por la red.

En la edición del periódico Infoworld de abril 11 de 1988, apareció el reporte de una infección hecha por un virus en las computadoras personales de La Agencia de Protección del Ambiente de la NASA. Contratistas de la NASA le vendieron a ésta, computadoras Macintosh infectadas. Este virus se activa después de dos, cuatro o siete días de tener la infección y borra toda la información en los discos. Es conocido como el virus Scores.

El manifiesto de la Brigada Roja, organización terrorista, menciona como uno de sus objetivos la destrucción de los centros de cómputo de sus enemigos por medio de atentados que

incluyen los programas de computadora. En Japón al menos una vez este grupo dejó sin poder utilizar por varias horas las líneas de trenes de la ciudad de Tokio usando programas destructivos de computadora.

El virus de la Organización para la Liberación de Palestina (OLP), detectado en la Universidad Hebrea de Jerusalem es otro ejemplo de esta actividad terrorista y fue programado para activarse el viernes 13 de Mayo de 1988, que era el 40 aniversario del último día en que Palestina existió como nación.

Aldus Corporation, una empresa de gran prestigio, lanzó al mercado originales de su programa FeeHand para Macintosh infectados por un virus llamado Macintosh Peace, MacMag o Brandow. Este virus se elaboró para poner un mensaje de paz en las pantallas de las computadoras, con el fin de celebrar el aniversario de la introducción de Macintosh II al mercado, el 2 de marzo de 1988. Richard R. Brandow, editor de la revista MacMag en Montreal, Canadá, contrató a un programador para realizar el mencionado virus, que pronto se propagó por medio de los servicios de un BBS (Bulletin Board Servicies), servicio de información compartida por computadora vía modem.

El 2 de noviembre de 1988, dos importantes redes estadounidenses de computadoras conectadas a la Internet, ARPAnet y MILnet (Red Militar de los Estados Unidos), fueron víctimas de un programa nocivo que se introdujo en ellas, y en unas cuantas horas infectó cientos de máquinas, paralizó por dos días a más de 60 mil computadoras y afectó instalaciones militares de la NASA, universidades y centros privados.

El culpable del contagio informático más extenso ocurrido hasta hoy resultó ser Robert Morris Jr, un estudiante de 23 años, hijo de un prominente responsable de seguridad de la NASA, Robert Morris, (aquel que contribuyó a desarrollar Core Wars). En su poder se encontraron códigos a los que no tenía derecho de acceso y similares a los utilizados para propagar el virus. Morris introdujo el programa nocivo de la Internet como parte de un experimento que quería hacer operar sin ser detectado y sin producir daños. Aunque el experimento fue calificado como el pasatiempo de un estudiante aburrido, este hecho evidenció la enorme vulnerabilidad de los sistemas informáticos. Sin embargo, los efectos de tal acción han sido lo más sorprendente de todo: el joven Robert Morris es ahora considerado una dualidad contrastante: mitad delincuente mitad genio.

A finales de 1988, comenzó a aparecer en miles de monitores de computadoras personales la imagen de una pelotita rebotando. Este virus fue llamado Ping-pong o virus de Turín, por su lugar de origen. Aunque más tarde se determinó que no afectaba los sistemas de información, causó desconfianza en la comunidad informática por la novedad y la falta de conocimiento sobre sus efectos.

A partir de esta fecha, la apariciones de nuevos virus se hicieron más frecuentes y alarmantes. Ultimamente los medios de comunicación han provocado mayor temor en los usuarios. El último caso que se conoce de gran impacto a nivel mundial, fue el provocado por el virus Michellangelo en marzo 6 de 1992, fecha en que se conmemora el natalicio del escultor y pintor italiano Miguel Angel. En México no podemos de dejar de mencionar los graves efectos causados por el virus NATAS ó SATAN a su aparición en 1994.

Aunque de hecho, existen virus más dramáticos y difíciles de detectar, Miguel Angel mostró que el problema se ha extendido más allá de las esferas universitarias y de centros de investigación o instituciones. Los medios de comunicación se han encargado de hacer de los virus un tema más cotidiano y popular.

Causas

Los virus se han convertido en un problema importante de nuestro tiempo, y aunque no es posible establecer una razón general que incite a las personas a escribir programas nocivos, se pueden mencionar algunas causas particulares.

- El número de personas que utilizan equipo de cómputo se ha incrementado geométricamente.

- Existen actualmente millones de computadoras de todas clases. Así por ejemplo, se tenían registradas para finales de 1994 100 millones de copias del sistema operativo DOS, para computadoras personales, lo que ha provocado que éste sea un sistema operativo muy estudiado,

- Poca seguridad.

- Los programas y formatos de datos cada vez son más estandarizados, por lo que aumenta la posibilidad de que un virus en un sistema pueda funcionar en otro distinto.

- El aumento en la utilización de redes de computadoras que permiten conectar decenas o cientos de sistemas.

- El aumento en la copia ilegal de programas.

- La falta de información adecuada sobre el tema.

El propósito por el cual se escriben los programas nocivos puede ser muy variado, algunas de las razones podrían ser las siguientes:

Reto técnico

La mayoría de los accesos no autorizados a los sistemas de cómputo parecen ser atractivas a jóvenes programadores para experimentar nuevas emociones y poner a prueba sus habilidades y conocimientos en computación. Tal vez esta es la razón principal de los llamados "hackers", como el caso de Robert Morris Jr., quienes muchos jóvenes admiradores le han llamado "ULTIMATE BACK", el término original empleado para describir a programadores quienes son capaces de escribir código eficiente y elegante que provoca la envidia de sus colegas.

Protección

En algunos casos, lo que motiva a desarrollar programas nocivos es el esfuerzo para penetrar en los sistemas de cómputo sin ser detectados y así, sin el deseo de causar daño, entender mejor la protección de los mismos. Sin duda, tales penetraciones a los sistemas de seguridad tienden a mostrar las habilidades de los hackers, los cuales son contratados como consultores en seguridad.

Venganza

Existen algunos casos en los cuales el propósito principal de crear programas nocivos está ligado a un resentimiento, por ejemplo, alguna persona que pretende adquirir programas útiles a precios muy bajos, entoces adquiere programas destructivos y en respuesta a esta situación actúa en forma similar.

Por ideales

Muchos de los agresores ven a la información como algo público que no debería de ser atesorada y prohibirse el acceso a la misma. Estas personas se catalogan como "socialistas" de la información, quienes creen que todos los sistemas deben tener acceso abierto para manipularla como lo deseen.

Sabotaje potencial

Cada vez se tienen más evidencias de productos terroristas destinados a invadir sistemas de computadoras. Lo anterior se desprende de los reportes hechos por la Central Intelligence Agence (CIA) y la National Security Agency (NSA), quienes están experimentando con el uso de virus como un arma estratégica. Algunos analistas predicen que este tipo de terrorismo electrónico se hará más constante con el tiempo.

Terrorismo computacional

El terrorismo ya no se limita a atentados contra seres humanos y edificios o aviones. Ahora hay una nueva forma de terrorismo: el terrorismo computacional. Las variantes son muchas, desde tomar la torre de control de un aeropuerto y provocar la colisión de aviones al proporcionar información errónea, hasta la destrucción de la información de los sistemas gubernamentales.

Según expertos en sistemas de seguridad para computadoras, las computadoras están tan mal protegidas que prácticamente se pueden considerar indefensas. Con software y hardware convencional, un terrorista prodría ser capaz de paralizar el sistema telefónico de una región o modificar la información de un banco.

Perpetradores

Haciendo un pequeño análisis acerca de estos casos existe una variedad de perpetradores, algunos de los cuales pueden ser fácilmente clasificados por sus motivos maliciosos. A continuación se mencionan algunos casos:

Empleados

Muchas de las agresiones son causadas por empleados autorizados, que en lugar de ver por los intereses de su organización actúan para su propio beneficio.

Desarrolladores de software

Algunos desarrolladores de software deliberadamente se protegen, en forma negativa contra copias no autorizadas de su software. Estos contienen "bugs" que fueron deliberadamente insertados en el código del software para protegerse de la piratería. Como consecuencia, muchas de las grandes firmas de software ven disminuidas sus ventas, ya que los usuarios censuran esta clase de esquemas de protección.

PranKsters

La palabra "PranKsters" es más usada que el término "hacker" para describir a usuarios jóvenes de computadoras (entre los 10 y 19 años), que como intento para desarrollar sus habilidades computacionales, deliberadamente, pero con intenciones no maliciosas intentan infiltrarse en sistemas de cómputo a los cuales no tienen acceso, provocando por su falta de experiencia que esta infiltración salga de su control y provoque grandes problemas.

CyberPunks

Este término se utiliza para describir a individuos con habilidades computacionales pero con una actitud antisocial, quienes deliberadamente tratan de dañar sistemas de cómputo únicamente por diversión y satisfacción personal.

Profesionales

Existe otro tipo de personas cuyo único interés es el lucro. Desarrollan programas nocivos en el anonimato para después vender la solución al problema, con lo cual obtienen grandes ingresos.

V.I.R.U.S. PROTECTION
Pamela Kane
Bantan Computers Books
1989

Computer Viruses
Jonathan L. Mayo
Windcrest
1990

ROGUE PROGRAMS Viruses, Worms, and Trojan Horses
Lance J. Hoffman and Van Nostrand Reinhold
1991

Scientific American, mayo de 1984
"In the game called Core Wars hostile programs engage in a battle of bits"
A. Kee Dewdney

Scientific American, marzo de 1985
"A Core War bestiary of viruses, worms and other threats to computer memories"
A. Kee Dewdney

Scientific American, enero de 1987
"A program called MICE nibbles its way to victory at the first Core Wars tournament"
A. Kee Dewdney

American Scientist, volumen 76, mayo/junio 1988
"Computer Viruses"
Peter J. Denning

TIME, septiembre de 1988
"Invasion of the Data Snatchers"
Phillip Elmer-De Witt

Communications of the ACM, Vol. 27 No. 8, agosto de 1984
"Reflections on Trusting Trust"
Ken Thompson

Ferreira, Gonzalo
"Virus en las computadoras"
Macrobit
1991

PC-TIPS, diciembre de 1988
"¿Ya vacunó a su PC?"
Alberto Rojas Ponce

Proceedings of the IEEE, vol. 63, No. 9, septiembre de 1975 "The Protection of Information in Computer Systems"
Jerome H. Saltzer, Michael D. Schroeder

PC-Semanal
"Terrorismo Computacional"
Martin Cheek
25 de marzo de 1992 , p. 26

"Theory and Organization of Complicated Automata"
John von Neumann

Infoworld de abril 11 de 1988

"The complete computer virus handbook"
Waterhouse Price
London Pitman

Repercusiones 3

Los virus informáticos son hoy una realidad reconocida por las empresas dedicadas a la fabricación de software y hardware, e inclusive las oficinas de gobierno los reconocen como un problema que mina su productividad en el área de la computación. La existencia de los virus es consecuencia de la naturaleza inquisitiva de los seres humanos. El desarrollo de virus, en sus inicios, no necesariamente puso en peligro la integridad de la información en los sistemas de cómputo. La pregunta de que si es posible escribir un programa que haga una réplica de él en otros, promovió el avance de la ciencia en el área de la computación y tenía fines académicos. Sin embargo, como siempre sucede, existen personas que usan estos descubrimientos para fines destructivos.

Se habla mucho de infecciones virales que afectaban las computadoras de centros de investigación, de institutos de educación o de grandes empresas. Esto provoca mucho temor y desaliento en la comunidad informática mundial que ha tenido contacto con esta plaga. Por fortuna, el temor al contagio ha servido para concientizar a los usuarios, a fin de que utilicen discos de programas originales y no confíen de las copias ilegales que les ofrecen.

La Computer Virus Industry Association (CVIA), fundada por John McAfee en 1989, reportó que en 1990 -sólo en Estados Unidos- más de 500 formas de "infecciones por virus" afectaron a unas 200 mil computadoras. No obstante, es posible que aproximadamente un 50% de los casos no se hayan denunciado. Los costos generados por los virus informáticos son muy altos, fundamentalmente por concepto de pérdida de información que tuvo que ser nuevamente generada.

Los costos de estas infecciones son gigantescas. Las estimaciones de los daños producidos por la infección Internet, solamente en noviembre de 1988 llegaron a más de 100 millones de dólares en tiempo de máquina y accesos perdidos, así como costos de trabajo directo para recuperación y limpieza. Y esto fue de un virus que supuestamente no intentaba causar daños.

Si el término virus informáticos (nombre que se le ha dado a estos programas) hubiera sido otro, tal vez no se habría producido el histerismo que actualmente vive la comunidad informática. Lo anterior, aunado a los rumores alarmantes, hacen que los nuevos usuarios sientan un temor infundado hacia las computadoras. Hay incluso quienes justifican ante sus superiores su ineficiencia, argumentando a ataques virales los trabajos que tardan demasiado tiempo en entregar o que tienen muchos errores.

Los virus han dado origen a una gran controversia. Mientras que los usuarios opinan que es una acción terrorista y de manifiesta falta de ética, los fabricantes de software opinan que en algunos casos se justifica la utilización de esquemas de protección que -aunque no se llamen virus- contengan códigos muy parecidos. Por un lado se cuestiona la legalidad de incluir o no un esquema de protección tipo virus en el software original, mientras que por el otro prevalece la duda de si es ético hacerlo. Tal situación ha provocado que los departamentos de justicia de los gobiernos se preocupen por legislar y aplicar sanciones contra aquellos individuos que atenten contra la información; pero aún no se han puesto de acuerdo en definir si debe catalogar como un delito administrativo o un crimen computacional.

Los virus han sido un dolor de cabeza pero, finalmente, han hecho recapacitar a la comunidad en diferentes aspectos: Para empezar, los virus mostraron que los sistemas de cómputo

son muy vulnerables. Trátese de micros, minis o mainframes, éstos programas han invadido todo tipo de sistemas. Los esquemas de claves (login) y contraseñas (password) son deficientes, fáciles de sobrepasar y han demostrado su relativa inutilidad.

Un efecto positivo de la aparición de los virus de computadora, es la preocupación por mejorar los esquemas y procedimientos de seguridad en los sistemas de cómputo. Cada vez más los sistemas se convierten en verdaderas fortalezas que garantizan la seguridad de la información; esto no sólo se aplica a las computadoras, sino al personal que labora con ellas.

Los usuarios de computadoras se preocupan por documentarse más respecto a temas relacionados con seguridad y virus informáticos, asistiendo a seminarios, conferencias y cursos. Esto ha producido un incremento en la cultura computacional.

Otro aspecto importante de mencionar, es el nacimiento de una nueva industria de software: la de los antivirus, orientada a erradicar programas nocivos. No sería extraño que algunas de estas mismas compañías dedicadas a salvaguardar la seguridad de los usuarios, fueran las creadoras de los virus.

Asimismo, los virus se han convertido en un tema de interés para los investigadores, quienes ocupan gran parte de su tiempo en estudiar su funcionamiento, estructura y efectos. Algunos de estos estudios han logrado avances en el desarrollo de algoritmos genéticos, que forma parte de la teoría de Inteligencia Artificial.

Con respecto al aspecto legal, todos los estados de la Unión Americana tienen estatutos que citan específicamente ciertas formas de abuso de la computadora y lo catalogan como un crimen.

Los temas de "seguridad en las computadoras" y "crímenes basados en computadora" han sido muy tratados en la década de los ochentas. Gran variedad de medidas han sido instituidas, puestas en vigor y observadas para intentar que los centros de cómputo no sean vulnerables a las intervenciones humanas intencionales o accidentales.

Algunas de estas medidas enfocan el tema de la ética al uso prudente de las computadoras. En la copia "pirata" de un programa de software se están violando las leyes de derechos de autor, convirtiéndose esto técnicamente en un crimen. Sin embargo es prácticamente imposible controlar esta acción y asignar una sanción para la misma, debido a que son miles de personas las que la realizan.

Es posible establecer entonces, responsabilidades sobre el uso de software y hardware para los empleados de un centro de cómputo o personas que tienen acceso a los mismos.

Muchas asociaciones de computación han generado lineamientos y reglas, con el propósito de establecer una normatividad. Algunos de estos organismos son internacionales y sus respectivos códigos de ética son los siguientes:

- Sociedad Británica de la Computación (British Computer Society), "Código de Conducta" (Code of Conduct).

- Asociación para la Administración y Procesamiento de Datos (Data Processing Management Association), "Código de Etica, Estandars de Conducta y Ejecución de Procedimientos" (Code of Ethics, Standards of Conduct and Enforcement Procedures).

- Asociación para la Maquinaria de la Computación (Association for Computing Machinery), "Conducta Profesional y Procedimientos de Ejecución de la Asociación para la Maquinaria de la Computación" (Professional Conduct and Procedures for the Enforcement of de ACM Code of Profesional Conduct).

- Instituto de Ingenieros Eléctricos y en Electrónica (Institute of Electrical and Electronic Engineers), "Código de Etica" (Code of Ethics).

- Instituto para la Certificación de Profesionales de la Computación (Institute for Certification of Computer Professionals), "Código de Etica y Prácticas Seguras" (Code of Ethics and Good Practices).

- Asociación de Seguridad en Sistemas de Información (Information Systems Security Association), "Código de Etica" (Code of Ethics).

Los códigos de ética citados anteriormente tratan principalmente los siguientes puntos:

1. Dignidad y mérito de otras gentes.

2. Integridad personal y honestidad.

3. Responsabilidad en el trabajo.

4. Manejo confidencial de la información.

5. Seguridad pública, salud y bienestar.

6. Participación en sociedades profesionales para determinar estándares al ejercer la profesión.

7. Tener presente que el conocimiento público y el acceso a la tecnología es equivalente al poder social.

BYTE, noviembre de 1990
"Stomping the Nasties"
Hugh Kenner

BYTE, enero de 1991
"Real Artifical Life"
Richard Marlon Stein

BYTE, enero de 1991
"Genetic Algorithms"
Peter Wayner

ComputerWorld
Abril 24 1989
"Manejo de la amenaza de los virus"

"Virus en las computadoras"
Gonzalo Ferreira
2a edición, Macrobit, 1991

PC-Semanal
"Un nuevo enfoque al problema de los virus"
Manuel López Michelone
25 de marzo de 1992, pág 15

Aspectos de seguridad 4

Seguridad en sistemas de cómputo

La seguridad en los sistemas de cómputo no sólo está constituida por guardias, sistemas de alarmas o detectores infrarrojos, los cuales son considerados como seguridad física. El concepto de seguridad en sistemas de cómputo va mas allá, está orientado a proteger a la información desde el mismo interior de la computadora. Las dos formas de seguridad (la física e interna) son válidas siempre y cuando estén orientadas a proteger al sistema contra vandalismo y posibles accidentes. La seguridad en los sistemas de cómputo involucra a tres grandes áreas: contabilidad, protección del sistema y protección de los datos.

Contabilidad

Esta área se avoca a la posibilidad de registrar a quienes trabajan y qué trabajo están efectuando. Esto se logra mediante la identificación del usuario y el registro de todas sus actividades desde que ingresa al sistema hasta que termina o sale de él.

Protección del sistema

Esta área está orientada a restringir el acceso a los recursos de hardware disponibles en el sistema de cómputo. Se otorgan privilegios para el uso de los recursos dependiendo del tipo de usuario.

Protección de datos

Esta área está orientada al control de acceso a la información ya sea para ser leída, copiada, alterada o borrada por parte de algún usuario del sistema.

Seguridad en redes de computadoras

Las comunicaciones y las redes computacionales han extendido el uso completo de las computadoras. Actualmente no hay forma de tener a una computadora segura dentro de una red si ella está conectada con otras que no son seguras. Una computadora no confiable puede obtener información que se transmite entre dos computadoras supuestamente confiables. Esto hace que la red se convierta en insegura. Los modems conectados a computadoras son un problema potencial, cualquier persona con una computadora en cualquier lugar puede entrar a una red a través de modems (vía telefónica) y conectarse a una computadora emulando una terminal de un nodo conocido de la red. Los actuales sistemas de seguridad para modem solucionan este problema con el concepto de "Llamada de regreso" (Call back), que consiste en que las computadoras que reciben una llamada de otra emisora, contestan la llamada con otra llamada de regreso asegurándose de que la emisora es quien dice ser. Sin embargo esto produce que el costo de la comunicación se carge al nodo que recibió la primera llamada. Un medio para mejorar la seguridad en las redes es el encriptamiento de datos, sin embargo esto produce un incremento en los costos de transmisión, y aún no hay métodos de encriptamiento que no hayan sido violados.

El sistema Kerberos, desarrollado como resultado del proyecto Athena del Instituto Tecnológico de Massachusetts (MIT), es uno de las nuevas formas para administrar la seguridad en redes. Consiste en dedicar una tercera computadora para controlar la comunicación entre otras, verificando la autenticidad de los usuarios que desean tener acceso a nodos remotos. Este sistema definitivamente incrementa la seguridad pero requiere de muchas utilerías adicionales. Kerberos ha sido aceptado por la "Fundación de Sistema Abiertos" (Open Software Fundation) para la autentificación de computadoras en una red.

Modelo básico de seguridad

La seguridad en las computadoras es un concepto relativamente nuevo. Las primeras investigaciones tuvieron lugar a finales de los años sesentas y principios de los setentas. Sin embargo, en los modelos existentes de seguridad a principios de los ochentas, la información aún se protegía en papel, cinta o disco bajo la responsabilidad de una o varias personas a cargo y dentro de una habitación con candados y alarmas. Como es sabido, esto ya no representa una garantía respecto a las nuevas formas de violación a los sistemas de cómputo. Se requieren de nuevos modelos de seguridad, que nuevamente mantengan confiable la información de las computadoras. Ahora también, la tarea de identificar a los usuarios de computadora, de monitorear su actividad y el control del acceso recae en el hardware y software de la computadora. La parte de seguridad del hardware consiste de un mecanismo para restringir regiones de memoria y prevenir que el software se ejecute con la intervención de software o

datos de otras regiones.

El sistema operativo es el software de administración de seguridad. A la combinación de hardware y sistema operativo responsables por reforzar los procedimientos de seguridad, se le conoce como "Base de la Computación Confiable" (Trusted Computing Base).

El modelo de seguridad actual se fundamenta en cinco elementos principales:

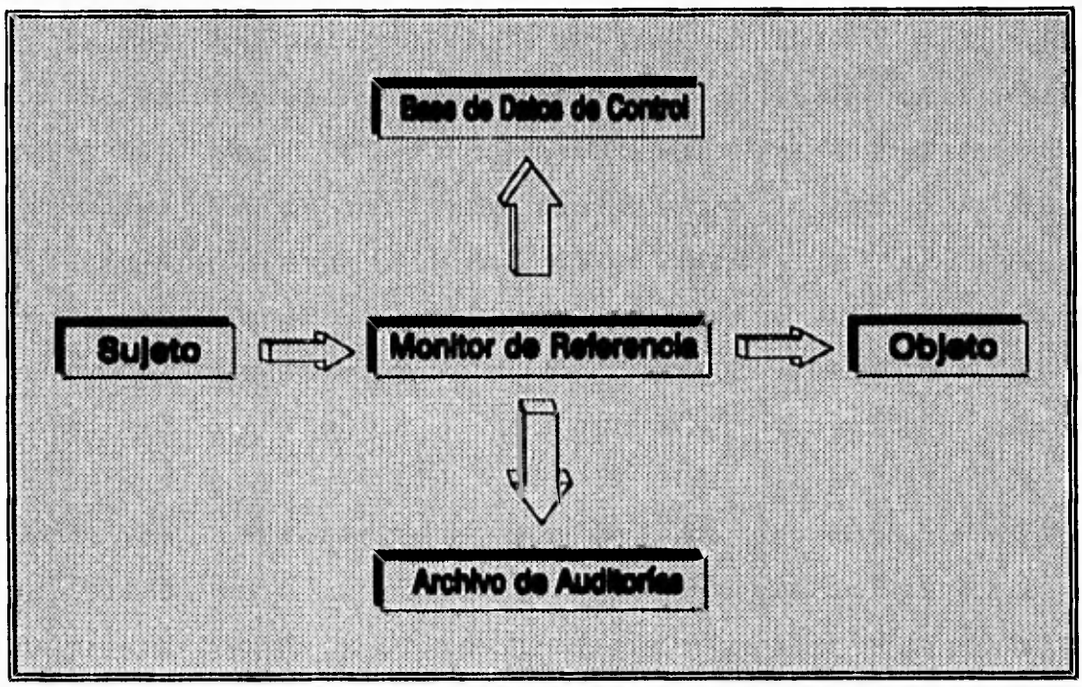
- Sujetos
- Objetos
- Monitores de referencias
- Base de datos de control y atributos
- Archivo de auditoría

Los sujetos son entidades activas (personas o programas) en pro de los usuarios.

Los objetos son entidades pasivas que contienen información (archivos, dispositivos y el sistema mismo). Por ejemplo, si un usuario desea modificar un archivo con algún editor, al editor se le considerará el sujeto, mientras que al archivo se le nombrará el objeto.

Un monitor de referencias es el software que media entre la conexión de sujetos y objetos. El monitor no es el sistema operativo, es una pequeña sección de programa que es responsable por el control del acceso de los sujetos a los objetos. El monitor de referencia hace uso de dos recursos, la base de datos de control y atributos y el archivo de auditoría. La base de datos de

control contiene información acerca de los objetos, así como datos acerca de qué sujeto tiene acceso y cuál es el tipo de acceso. El archivo de auditoría mantiene un registro de todas las acciones realizadas por los sujetos sobre los objetos. La interacción de estas partes se ilustra en la siguiente figura:



Por ejemplo, en un sistema multiusuario, para poder ver el contenido de un archivo se requiere de privilegios. El sujeto sería el comando para leer el archivo, el monitor de referencias es el encargado de observar en la base de datos de control si el sujeto tiene permisos de lectura (asociados al usuario que ejecuta el comando), registrando la transacción en un bitácora, que

representa el archivo de auditoría y en la que se indica la operación sobre el archivo, que es el objeto en cuestión.

El Libro Naranja

El gobierno de los Estados Unidos es el más grande usuario de computadoras que requieren de seguridad. Algunos de sus departamentos siguen normas establecidas dentro de leyes por el Congreso para el manejo de documentos especiales. El sistema de clasificación de seguridad define niveles y categorías. Los niveles de seguridad tienen nombres familiares, por ejemplo "Ultra Secreto" (Top Secret), "Secreto" (Secret), "Confidencial" (Confidential) y "No clasificado" (Unclassified). La idea de categorías también corresponde a otro principio de seguridad, el de la "necesidad de conocer". Este principio es más de sentido común: entre más gente conozca un secreto, menos secreto será.

El problema de formalizar esta clasificación para las computadoras, recayó en el Centro Nacional de Seguridad en Sistemas de Cómputo (National Computer Security Center), quien fue el encargado de definir los criterios para determinar cómo una computadora y su software encajaban dentro de estos esquemas. El Centro Nacional de Seguridad en Sistemas de Cómputo (NCSC), diseñó el documento "Evaluación de Normas de Seguridad en Sistemas de Cómputo" (Trusted Computer System Evaluation Criteria) que sirve como base para evaluar la confiabilidad de los sistemas. Popularmente se le conoce como "El Libro Naranja" (The Orange Book) debido al color de sus pastas originales. El Libro Naranja fue publicado por el Departamento de la Defensa de los Estados Unidos en 1985 y se considera el estándar para establecer el grado de

seguridad en los sistemas de cómputo sin enfocarlo a un diseño específico de software y hardware. Las actuales versiones de los sistemas de cómputo están basados en las normas que se encuentran establecidas en este documento.

Definiciones del Libro Naranja

El Libro Naranja establece normas para los siguientes puntos:

- Una clara y precisa documentación de políticas de seguridad.

- Capacidad de identificar en forma precisa a todos los usuarios y llevar un control eficiente de los eventos que realiza cada uno de ellos.

- Que el sistema desarrolle confiablemente sus funciones.

- Guía para administradores y documentación de usuarios para dar soporte y mantenimiento al sistema.

Las pruebas de seguridad para los sistemas del Centro Nacional de Seguridad en Sistemas, involucran tanto al hardware como al software.

El desarrollo de un sistema altamente confiable toma mucho tiempo y es extremadamente caro, así que dependiendo del grado de seguridad con que fue diseñado deberá ser incluido en un determinado nivel de seguridad antes de poder ser certificado.

El Libro Naranja sigue el modelo de seguridad en cómputo Bell-LaPadula, el cual define la relación entre los sujetos y objetos. Esta relación está basada en niveles y categorías e introduce un nuevo término, **dominantes**.

Un sujeto domina a un objeto si el nivel del sujeto es mayor o igual que el nivel del objeto, y la categoría del sujeto incluye todas las categorías del objeto.

Existen dos reglas básicas en el modelo de seguridad de Bell-LaPadula, la primera, con modo de lectura y la segunda con modo de escritura. Para lectura, el sujeto debe dominar al objeto, por ejemplo, se pueden leer objetos del mismo nivel o inferior, el principio es llamado Read-Down y se le conoce como simple seguridad. Para escritura, el objeto deberá dominar al sujeto, por ejemplo, no se puede escribir a un archivo marcado como Secret cuando el proceso esté operando a nivel Top Secret o esté trabajando adentro de una diferente categoría. El principio anterior es llamado Write-Up y protege de usuarios que intencionalmente o accidentalmente destruyen información al escribir datos en archivos pocos seguros.

Requerimientos de seguridad del Libro Naranja

Políticas de seguridad (Security policy)

Para asignar la identificación a sujetos y objetos, debe existir un conjunto de reglas que deberá usar el sistema para determinar a qué sujetos se les permite tener acceso a ciertos objetos.

Marcas o etiquetas (Marking)

Para controlar el acceso a la información guardada en la computadora de acuerdo con las reglas de las políticas obligatorias de seguridad, deberá de ser posible marcar a cada objeto con una etiqueta para identificar el nivel real del objeto (clasificarlo), y/o los modelos de acceso acordes con esos sujetos sean quienes puedan tener acceso potencial a los objetos.

Identificación (Identification)

Cada acceso a la información deberá estar basada en quien está realizando el acceso y qué clase de información está autorizado a utilizar.

Contabilidad (Accountability)

Los sistemas seguros deben de ser capaces de registrar las ocurrencias de eventos relevantes de seguridad.

Aseguramiento (Assurance)

Para asegurarse que los cuatro requerimientos de Políticas de seguridad (Security policy), Marcas o Etiquetas (Marking), Identificación (Identification) y Contabilidad (Accountability) se están utilizando en los sistemas de cómputo, deberá haber una colección de hardware y software unificada e identificada que controle el desarrollo de estas funciones.

Protección continua (Continuous protection)

Los sistema de cómputo no puede ser considerados altamente seguros si los mecanismos básicos de hardware y software que deben de utilizarse en las políticas de seguridad son en sí, los mismos sujetos que no están autorizados a realizar modificaciones y actos de subversión.

Divisiones y niveles

El Libro Naranja define cuatro divisiones, desde la división D (Mínima protección) hasta la división A (Protección verificada). Dentro de estas cuatro divisiones existen un total de siete niveles. Cada nivel incluye todo los elementos de seguridad de los niveles que le preceden; es decir, los niveles están construidos jerárquicamente. Los niveles se identifican por la división a la que pertenecen seguida de un número, los números mayores denotan más seguridad, así por ejemplo, la seguridad del nivel B2 es más grande que la del B1 y la del A1 es más grande que todos los otros niveles. A continuación se muestran las cuatro divisiones con sus respectivos niveles de seguridad:

D-Protección mínima (Minimal protection)

Un sistema que está dentro de esta división no se puede catalogar como seguro. Estos sistemas son esencialmente inseguros y no dan mas allá de una protección física. Cualquier usuario que tenga acceso físico al sistema tendrá acceso a los recursos y a los archivos del mismo. Computadoras personales que estén corriendo DOS, Macintosh que

no estén corriendo UNIX, y Atari Amigas (que también no estén corriendo UNIX) se encuentran en esta división. Existen algunos paquetes que pueden incrementar la protección en computadoras personales, pero no por esta razón podrán subir a la siguiente división.

C-Protección de seguridad discreta (Discretionary protection)

C1-Protección discreta en seguridad (Discretionary security protection) (Separación entre usuarios y datos)

En este nivel se permite que cada usuario tome el control de los objetos que le pertenecen. El usuario puede no tener permisos para borrar o modificar archivos que no le pertenecen, pero generalmente puede tener permisos de lectura, lo que le permite hacer copias de la información y cambiar los atributos de la misma. Los privilegios están basados en tres categorías, usuario (user), grupo (group) y otros (other). El usuario debe identificarse con el sistema a través de una cuenta (login name) y una contaseña (password). El "login name" es usado para identificar las acciones del sujeto.

C2-Protección de acceso controlado (Controlled access protection)

Este nivel agrega revisiones (auditoría) e incrementa el proceso de identificación (más estricto que el nivel C1). Registra y examina los eventos relativos a seguridad, por ejemplo, las actividades que desarrolla el administrador.

La auditoría puede se configurada para incrementar el número de eventos que serán registrados. La auditoría trae consigo la necesidad de incrementar la verificación de autenticidad de cada usuario, ya que ésta sólo registra el evento más significativo, y no considera la persona que lo efectúa. La auditoría tiene el efecto secundario de utilizar recursos del sistema, tales como el procesador y subsistemas de disco. El encriptamiento de contraseñas (passwords) permite que los usuarios sin privilegios pueden usarlas. En C1, las contraseñas encriptadas se encuentra un archivo de la base de datos de los usuarios. Utilizar la técnica de encriptamiento ayuda a prevenir intentos de ataques a los sistemas.

En resumen, un usuario debe ser capaz de proteger los datos de modo que este disponible sólo a un usuario específico. Debe mantener un auditoría que lleve la cuenta de los accesos e intentos de acceso a objetos, tales como archivos.

B-Protección obligatoria (Mandatory protection)

B1-Protección de seguridad etiquetada (Labeled security protection)

Los sistemas en el nivel de seguridad B1 deben de tener capacidades de control de acceso obligatorias. En particular, los sujetos y objetos que son controlados deben ser etiquetados individualmente con un nivel de seguridad.

B1 es el primer nivel que soporta "multiniveles" de seguridad, por ejemplo, Secret y Top Secret. B1, incluye controles de acceso obligatorio; un objeto que

se encuentre bajo control obligatorio su propietario no puede cambiarle los permisos o atributos. La certificación en el nivel B1 requiere de una prueba informal del modelo de seguridad.

Las etiquetas agregan descripciones más completas de los niveles de seguridad y categorías a los sujetos y objetos. Las etiquetas no reemplazan los permisos y dominios de la seguridad discrecional pero son usadas adicionales a las ya existentes. En este nivel hay comandos adicionales que pueden ser vistos y manipulados por objetos etiquetados.

B2-Protección estructurada (Structured protection)

El nivel B2 requiere que cada objeto sea etiquetado o marcado. Dispositivos, tales como discos, cintas o terminales, pueden tener un sólo nivel de seguridad o múltiples niveles, por ejemplo, una unidad de cinta puede ser capaz de preservar las etiquetas de los objetos almacenados en ella. B2 requiere de una prueba formal de la exactitud del modelo formal de seguridad.

B2 incluye el principio de las restricciones de los canales de conversión (convert channels). Los canales de conversión son formas indirectas para establecer la comunicación entre un sujeto operando a nivel alto con otro proceso operando a un nivel mas bajo. Los canales de conversión constantemente implican un uso no apropiado de las capacidades de los sistemas.

B3-Dominios de seguridad (Security domains)

La seguridad de los sistemas en el nivel B3 deben basarse en un modelo conceptualmente sencillo, pero completo. Debe haber un argumento conviencente, pero no una prueba formal, de que el sistema implemente el diseño. Debe incluirse la capacidad de especificar protección de acceso para cada objeto y de especificar sujetos permitidos, junto con el acceso permitido a cada uno de ellos, y sujetos no permitidos. Un monitor de referencia que tome las peticiones de acceso de los usuarios y aprueba o desapruebe el acceso basado en políticas de control de acceso debe ser implementado.

El sistema debe ser altamente resistente a la penetración y la seguridad debe ser a toda prueba. Debe proporcionarse una facilidad de auditoría que pueda detectar violaciones de seguridad potencial.

El nivel B3 obliga a la separación de los dominios de seguridad con el hardware. El dominio de seguridad puede ser parte de Trusted Computing Base, por ejemplo, un monitor de referencias. El hardware manejador de la memoria protege la seguridad del domino contra accesos o modificaciones por operaciones de software en otros dominios.

Los sistemas en B3 deben de proveer de una ruta confiable. Las rutas confiables garantizan a los usuarios que la terminal usada esté conectada directamente a un software confiable. La ruta confiable previene de entradas a

través de login name y password a programas enmascarados como el proceso de login.

A-Protección verificada (Verified protection)

A1-Diseño verificado (Verified diseing)

Para certificaciones en este nivel se requiere de una prueba formal matemática de la exactitud del modelo de seguridad. La certificación también requiere de un análisis formal de los canales de conversión, una especificación formal de top-level y distribución confiable. La distribución confiable permite que el software y hardware estén protegidos durante la expedición o embarque para prevenirse de falsificaciones con los sistemas de seguridad.

Muy pocos sistemas cumplen con la certificación del nivel A1, En la actualidad sólo el sistema SCOMP de Honeywell está certificado en este nivel.

"ULTRIX System Management II: Security and Performance Management"
Educational Services, Digital Equipment Corporation

"UNIX System Security"
Rik Farrow
Addison Wesley, 1991

"UNIX System V Release 4: An Introduction"
Kenneth H. Rosen
Richard R. Rosinsky
James M. Farber
Osborne Mc Graw-Hill

Clasificación 5

Programas nocivos

Dentro de las diferentes formas de amenaza a la seguridad de los sistemas de cómputo se encuentran los programas nocivos. Un programa nocivo tiene como objetivo alterar la información o el funcionamiento normal de operación en los sistemas de cómputo. Se ha establecido una clasificación de los programas nocivos tomando como característica principal su modo de operación.

Virus

El término Virus Informáticos (Computer Virus) se ha utilizado ya que existe una analogía con el funcionamiento de los virus biológicos. Desde el punto de vista médico se conoce como virus a los más pequeños agentes infecciosos, que no son capaces de vivir independientemente por largos períodos, ni reproducirse autónomamente, sino que requieren de vivir en organismos más complejos que les provean de energía para la reproducción de sus componentes virales. Estos parásitos, al reproducirse, determinan notables anomalías metabólicas y la mayoría de ellos no dejan señales externas de su presencia, al menos en sus primeras etapas de gestación. Similarmente, un virus informático es un programa pequeño que se copia en otros programas que pueden ser parte del sistema operativo o aplicaciones desarrolladas por los usuarios; cuando estos programas infectados son ejecutados, también se ejecuta el código del virus y trata de autocopiarse. Generalmente, los virus informáticos causan deterioros en los sistemas de cómputo.

Un virus de computadora puede propagarse cuando los usuarios comparten cintas y discos infectados o intercambian programas infectados a través de ligas de comunicación entre computadoras como modems y redes. Los virus informáticos no pueden multiplicarse a través de archivos de datos, ya que los archivos de datos no son ejecutables. Sin embargo algunos archivos de datos que sirven de entrada para otros programas, pueden ser interpretados como código. Por ejemplo, los archivos de texto pueden contener secuencias especiales de caracteres que pueden ser ejecutadas como comandos de un editor. Bajo estas circunstancias, los datos son "ejecutados" y pueden activar el código de un virus. Los archivos de datos también pueden contener código oculto que es ejecutado cuando los datos son utilizados por otras aplicaciones, y éstas también pueden ser infectadas. Desde el punto de vista técnico, los archivos que contienen exclusivamente datos no pueden ser infectados.

Los virus informáticos son una amenaza a la seguridad e integridad de los sistemas de cómputo. Un virus puede causar la pérdida o alteración de programa o datos, y puede hacerlo sin intervención humana directa.

Tradicionalmente cuando alguien desarrolla una aplicación le fija un nombre. En el caso de los virus, normalmente la persona que lo descubre es quien le asigna un nombre, ya que su autor por lo general permanece en el anonimato. El nombre de un virus puede estar basado en el lugar en que se descubrió o donde ocurrió su mayor infección, tal es el caso de los virus: Alabama, Argentina, Australia, Alameda, Barcelona, Japan, Jerusalem, Korea, Lehigh, Ohio, Ontario, Paris, Taiwan, Tokyo, URSS, Vienna, etc. En otras ocasiones, el nombre de un virus se asocia al mensaje que despliega, por ejemplo los virus: Brain, Cookie, Den Zuk y Stoned. Otra característica que se toman en cuenta para asignar nombre a los virus es el efecto que se visualiza

Un virus de computadora puede propagarse cuando los usuarios comparten cintas y discos infectados o intercambian programas infectados a través de ligas de comunicación entre computadoras como modems y redes. Los virus informáticos no pueden multiplicarse a través de archivos de datos, ya que los archivos de datos no son ejecutables. Sin embargo algunos archivos de datos que sirven de entrada para otros programas, pueden ser interpretados como código. Por ejemplo, los archivos de texto pueden contener secuencias especiales de caracteres que pueden ser ejecutadas como comandos de un editor. Bajo estas circunstancias, los datos son "ejecutados" y pueden activar el código de un virus. Los archivos de datos también pueden contener código oculto que es ejecutado cuando los datos son utilizados por otras aplicaciones, y éstas también pueden ser infectadas. Desde el punto de vista técnico, los archivos que contienen exclusivamente datos no pueden ser infectados.

Los virus informáticos son una amenaza a la seguridad e integridad de los sistemas de cómputo. Un virus puede causar la pérdida o alteración de programa o datos, y puede hacerlo sin intervención humana directa.

Tradicionalmente cuando alguien desarrolla una aplicación le fija un nombre. En el caso de los virus, normalmente la persona que lo descubre es quien le asigna un nombre, ya que su autor por lo general permanece en el anonimato. El nombre de un virus puede estar basado en el lugar en que se descubrió o donde ocurrió su mayor infección, tal es el caso de los virus: Alabama, Argentina, Australia, Alameda, Barcelona, Japan, Jerusalem, Korea, Lehigh, Ohio, Ontario, Paris, Taiwan, Tokyo, URSS, Vienna, etc. En otras ocasiones, el nombre de un virus se asocia al mensaje que despliega, por ejemplo los virus: Brain, Cookie, Den Zuk y Stoned. Otra característica que se toman en cuenta para asignar nombre a los virus es el efecto que se visualiza

en los monitores como los virus Ping-Pong (simula una pelotita rebotando) y Cascade (da la impresión que el texto se cae a la parte más baja del monitor como una cascada). Algunas veces los virus tienen asociado como nombre el número de bytes que agregan a los programas que infectan, tal es el caso de los virus 1008, 191, 1704, 2930, 7808, 1280, etc. Se han dado nombres a los virus por la fecha en que causan estragos, por ejemplo los virus: August 16, Friday 13th, Saturday 14th y Sunday-2. En otras ocasiones el nombre de un virus se asigna de acuerdo al tipo de archivo que modifica o destruye asociado a algún software: dBase (archivos con extensión DBF), Anti-Pascal (archivos con extensión PAS), Wordperfect (archivos generados por Wordperfect). Otro tipo de nombre para los virus se asigna por el equipo al que infectan como el virus Mac, que se hizo para correr en equipos Macintosh de Apple.

Caballos de Troya

Un Caballo de Troya (Trojan Horse) es un programa similar a un Virus, con la diferencia que no contiene código de autoreproducción. Un Caballo de Troya se introduce a los sistemas bajo la apariencia de un programa útil, que al ser ejecutado realiza una acción inesperada, la cual la mayoría de las veces está dedicada a causar daños a los datos o al sistema.

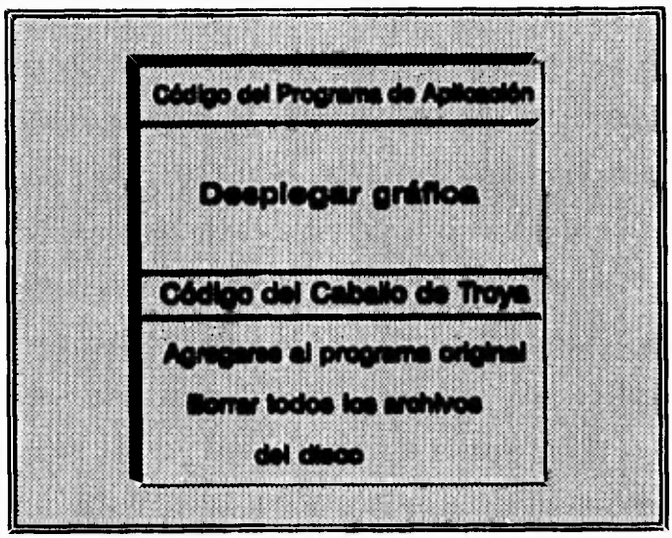
Un Caballo de Troya, agrega su código a un programa legítimo de aplicación, de tal forma que primero se ejecuta el programa de aplicación (puede ser todo o parte del mismo), dando la apariencia de un buen funcionamiento y posteriormente se ejecuta el código del Caballo de Troya. Esto se muestra en la siguiente figura:

en los monitores como los virus Ping-Pong (simula una pelotita rebotando) y Cascade (da la impresión que el texto se cae a la parte más baja del monitor como una cascada). Algunas veces los virus tienen asociado como nombre el número de bytes que agregan a los programas que infectan, tal es el caso de los virus 1008, 191, 1704, 2930, 7808, 1280, etc. Se han dado nombres a los virus por la fecha en que causan estragos, por ejemplo los virus: August 16, Friday 13th, Saturday 14th y Sunday-2. En otras ocasiones el nombre de un virus se asigna de acuerdo al tipo de archivo que modifica o destruye asociado a algún software: dBase (archivos con extensión DBF), Anti-Pascal (archivos con extensión PAS), Wordperfect (archivos generados por Wordperfect). Otro tipo de nombre para los virus se asigna por el equipo al que infectan como el virus Mac, que se hizo para correr en equipos Macintosh de Apple.

Caballos de Troya

Un Caballo de Troya (Trojan Horse) es un programa similar a un Virus, con la diferencia que no contiene código de autoreproducción. Un Caballo de Troya se introduce a los sistemas bajo la apariencia de un programa útil, que al ser ejecutado realiza una acción inesperada, la cual la mayoría de las veces está dedicada a causar daños a los datos o al sistema.

Un Caballo de Troya, agrega su código a un programa legítimo de aplicación, de tal forma que primero se ejecuta el programa de aplicación (puede ser todo o parte del mismo), dando la apariencia de un buen funcionamiento y posteriormente se ejecuta el código del Caballo de Troya. Esto se muestra en la siguiente figura:



En los sistemas multiusuario, un Caballo de Troya puede ser una emulación de la secuencia de entrada al sistema; cuando el usuario intenta entrar al sistema dando su clave de acceso (login name) y contraseña (password) éstos serán registrados en un archivo, que posteriormente el creador del Caballo de Troya utilizará, enseguida se ejecutará realmente el proceso de entrada al sistema dando la apariencia de que la primera vez se equivocó el usuario.

El siguiente programa ejemplo, escrito en Korn Shell (interprete de comandos), muestra un caballo de Troya para emular la entrada a un sistema UNIX y atrapar el login y password del usuario:

```
# !/bin/ksh
# Shell script diseñado para atrapar la clave (login) y
# contraseña (password) de los usuarios
#
set INTRO='Mensaje de bienvenida'
set FILE=$HOME/atrapa
clear
echo ' '
echo ' '
echo $INTRO
echo ' '
echo -n 'login: '
read login
stty -echo
echo -n 'Password:'
read password
stty echo
echo ' '
echo $login $password >> $FILE
echo 'Login incorrect'
login
```

El término Caballo de Troya se le dio a estos programas nocivos, debido a la similitud que existe con el relato de la Guerra de Troya. En la Iliada, epopeya griega atribuida a Homero, Troya fue sitiada por los griegos durante 10 años. Tras el último año de sitio e infructuosas batallas Ulises, guerrero griego, propone la creación de un Caballo (con soldados en su interior), como trofeo a los Troyanos, para hacerlos creer que se daban por vencidos. Fue así como pudieron penetrar y tomar la ciudad. De esta narración, se llega a la conclusión que no todo es lo que aparenta.

Otro ejemplo interesante de este tipo de programas en ambiente UNIX es el siguiente:

```
/*      Ejemplo de un caballo de Troya en Ansi-C para UNIX        */
#include <stdio.h>
#include <sys/file.h>
#include <limits.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/types.h>
void main ()
{
/*      seccion de inicialización          */
int fd1;                                   /* archivo /bin/sh          */
int fd2;                                   /* la copia de SUID         */
int n;                                     /* contador                 */
char buf[100000], *c = buf;                /* copia del buffer         */
fd1 = open("/bin/sh", O_RDONLY);           /* se abre el archivo destino */
fd2 = create("/tmp/shell", 4755);          /* creacion del nuevo shell   */
/*  se copia /bin/sh en /tmp/shell en incrementos de 100000 bytes      */
while ((n = read(fd1, c, 100000)) == 100000)
        write (fd2, c, n);
if (n != 0)
        write (fd2, c, n);
close (fd1);
close (fd2);
chmod("/tmp/shell", 04755);
}
```

El propósito de este programa es crear un shell que al ser ejecutado por cualquier usuario adquiera los privilegios de quien lo generó; esto tiene implicaciones importantes si se considera que el administrador puede ejecutarlo, y entonces cualquier otro usario puede utilizar los privilegios de super usuario sin necesidad de conocer la contraseña de éste.

Gusanos

Otra clasificación de los programas nocivos se le conoce como Gusanos (Worms). Un Gusano es un programa muy parecido a un virus, con la característica de que no utiliza otros programas para ejecutarse; éste es un programa completamente autónomo que puede sobrevivir por sí mismo.

El creador del Gusano lo deja corriendo en una computadora y a partir de ese momento comienza a realizar su misión destructiva, que se enfoca a dañar las aplicaciones que se encuentran en memoria. Esta misión se lleva a cabo mediante el desplazamiento del código del Gusano en las diferentes localidades sobreescribiéndose en las instrucciones y datos de otros programas.

El término Gusano se le dio a estos programas por la similitud que existe con los gusanos en los frutos, los cuales para subsistir se desplazan en el mismo consumiéndolo. Dado que este tipo de programa nocivo está diseñado para "viajar", es común encontrarlo en sistemas de redes de computadoras en los que se desplaza de nodo a nodo.

Los Gusanos son más comunes que los virus en los sistemas multiusuarios en red, tal es el caso del Gusano (y no virus) ejecutado por Robert Morris el 2 de noviembre de 1988, que ha sido el de mayor contagio y de mayor impacto ocasionado hasta ahora. Enseguida se incluye una cronología de este incidente en la que se puede observar la rapidez con que se desplazan este tipo de programas.

El símbolo ~ se usa para representar "aproximadamente".

2 de noviembre de 1988

~ 17:00	El Gusano es ejecutado en una máquina de la Universidad de Cornell. Tal vez esta fue una última prueba o la ejecución inicial del Gusano.
~ 18:00	Máquinas del dominio público del Instituto Tecnológico de Massachusetts fueron infectadas.
18:30	Máquinas de la Universidad de Pittsburg fueron contagiadas.
21:00	El Gusano se descubre en máquinas de Stanford.
21:30	La primera máquina en la Universidad de Minnesota es invadida.
22:04	Una máquina gateway de la Universidad de California, Berkeley es afectada.
22:34	Un nodo gateway de la Universidad de Princeton es infectado.
~ 22:40	El Gusano infecta Computadoras de la Universidad de North Carolina e intenta contaminar otras máquinas este plantel.

22:52	El Gusano intenta invadir máquinas en la Universidad de Carnegie-Mellon.
22:59	Se registra un intento de entrar a una máquina de la Universidad de Pennsylvania. En las siguientes 12 horas se registraron otros 210 intentos.
~ 23:00	Son afectadas máquinas de un laboratorio de Inteligencia Artificial del MIT.
23:28	En la Universidad de Maryland las computadoras son contaminadas vía correo electrónico (mail).
23:45	Máquinas de Dartmounth y los laboratorios de Investigaciones Balísticas de la Armada de los Estados Unidos son atacadas e infectadas.
23:49	Una máquina gateway de la Universidad de Utha fue infectada.

En la siguiente hora se producen más de 100 infecciones.

El 3 de noviembre de 1988, se reportaron treinta infecciones más, el 4 se detectaron otras ocho y se descubrió a Robert Morris como el autor del Gusano. El funcionamiento normal de la red se pudo establecer hasta el jueves 8 de noviembre. Este evento nos permite ver lo peligrosos que pueden ser este tipo de programas.

Bombas

Una Bomba, es un programa generalmente pequeño, que se agrega a un programa que ya existe, o en algunos casos realiza modificaciones a los programas ya existentes.

Las Bombas pueden ser de varios tipos: las Carta-Bomba (letter bomb) y las Bombas Lógicas (logic bomb).

Una Carta-Bomba es una pieza de información que se envía por correo electrónico (electronic mail) que contiene caracteres especiales de control con los que en algunos sistemas, con sólo ver el contenido de la carta, disparan la ejecución de comandos tal como si el usuario lo hubiera hecho por su propia mano.

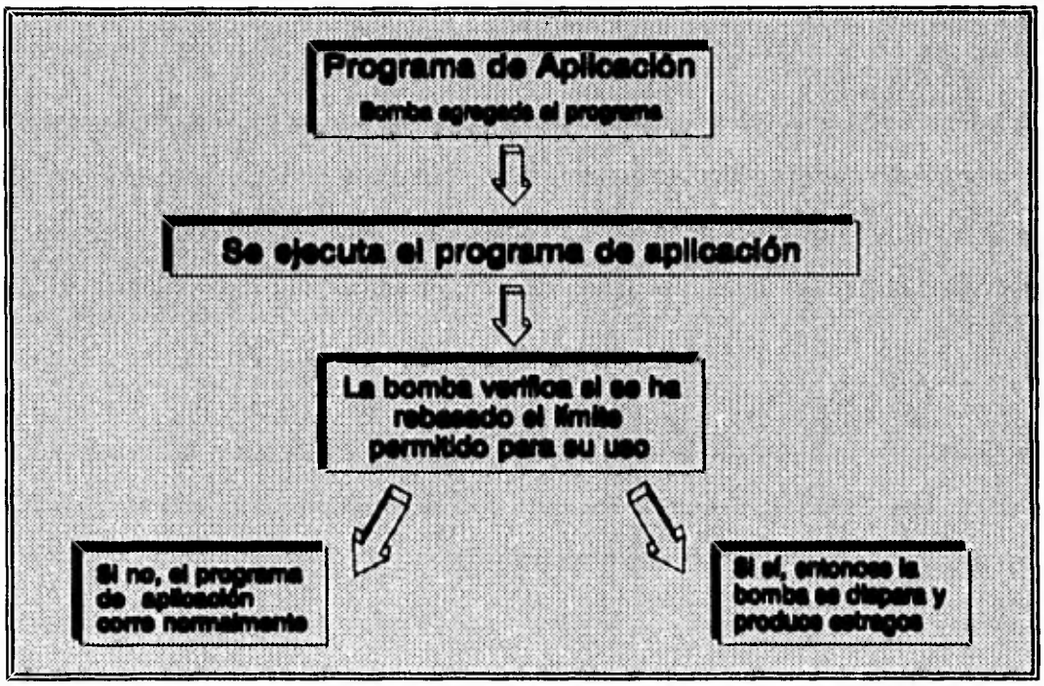
Las Bombas Lógicas son piezas de código agregadas a los programas de aplicación por personas que conocen el sistema en el que se ejecutará y tienen acceso al mismo. Por esta razón muchas bombas lógicas las generan programadores que son despedidos y tienen resentimientos en contra de la organización a la que pertenecían.

Se le ha llamado Bomba Lógica porque su función es "explotar" cuando se cumplen ciertas condiciones, por ejemplo que se cumpla una fecha.

Las Bombas Lógicas no afectan a los sistemas de cómputo degradándolos, como ocurre con los Caballos de Troya, Virus o gusanos. Estos programas son escritos para funcionar dentro de un software específico que está corriendo en un hardware específico. Su objetivo es permitir

que se ejecute la aplicación mientras no se cumpla el evento que active el mecanismo de disparo. Una vez que este evento se da, se ejecuta el código de la Bomba que produce deterioro.

Un ejemplo de este tipo de programas sería un paquete de software distribuido gratuitamente para su evaluación, el cual incluye un límite en el número de veces que se puede ejecutar. Mientras no se llegue a este límite el programa funcionará de una forma normal, cuando se sobrepasa este límite la Bomba se activa y produce su efecto nocivo, que puede ser destruir el paquete y/o la información generada por el mismo. Esto se muestra gráficamente a continuación:



Otras situaciones de riesgo

Los sistemas de cómputo no sólo son sensibles al ataque de programas, sino también a individuos que intentan penetrar en ellos a través de violar o evadir las medidas de seguridad establecidas.

Hackers

El término hacker tiene dos connotaciones:

La connotación positiva se refiere a un usuario que husmea dentro de los sistemas sin ningún otro propósito que el educacional o de investigación para satisfacer su curiosidad personal, afectando la privacidad de otros usuarios. La connotación negativa se refiere a aquellos que sí tienen propósitos criminales o de espionaje. De esta manera se pueden distinguir dos tipos de hackers:

El estudiante, que lo hace por diversión, reto o por el acceso a un mayor número de computadoras. Los PranKsters entran en este tipo de hackers. Típicamente actúan sólos, desorganizados y usualmente con fines no maliciosos.

El organizado, que actúa como parte de un grupo, con objetivos claros y definidos:

- Para espionaje industrial o periodístico
- Para espionaje internacional
- Con fines terroristas

Los hackers, en general, hacen uso de diferentes medios para evadir o violar los sistemas de seguridad, ya sea a través de aparatos electrónicos, programas de detección de claves de acceso o por programas nocivos (tales como los Virus, Gusanos, Bombas o Caballos de Troya).

Puertas de acceso

Las Puertas de acceso (trap doors), son rutinas especializadas de acceso insertadas en los sistemas y desarrolladas por programadores profesionales que trabajan para empresas de software o instituciones de investigación, con el objeto de que la gente de mantenimiento de dichas empresas o instituciones pueda tener acceso a los sistemas de sus clientes cuando no esté presente el administrador de esos sistemas. El caso más famoso de este tipo de código es el creado por Ken Thompson, el padre del sistema operativo UNIX.

Aun cuando muchas compañías desarrollaron sus propias versiones de UNIX, Thompson y sus amigos podían entrar con privilegios de administrador, a cualquiera de esos sistemas. Expertos programadores examinaron minuciosamente el código fuente de UNIX en busca de alguna palabra clave de acceso; particularmente en el programa Login que es el que permite

entrar a los usuarios al sistema, pero no encontraron nada. Para estar seguros, volvieron a compilar el código de UNIX. Perplejos se dieron cuenta que Thompson todavía tenía acceso, por lo que llegaron a la conclusión de que el problema estaba en el compilador. Se compiló nuevamente el código del compilador de C y con él todo el sistema operativo UNIX y atónitos observaron como Thompson seguía teniendo acceso.

Thompson instaló, junto con Dennis Ritchie (el creador del lenguaje de programación C), una puerta de acceso en la primera definición del compilador de C, de tal manera que, se insertaría como parte del código si el programa que compilaba era el programa **Login**. Esto lo hacía identificando una secuencia única de llamadas al sistema, por lo que el código sólo se insertaría en este programa. Además, reconocía si el programa que compilaba era un compilador de C, e insertaba nuevamente el código de la Puerta de Acceso (esto técnicamente hablando era un virus). Así, la única forma de quitar esta puerta de acceso era remover al virus de la definición original.

Esta Puerta de Acceso no fue descubierta por más de 10 años, hasta que Thompson hizo un discurso en ocasión de recibir un premio de la Asociación para la Maquinaria de la Computación (Association for Computing Machinery) en 1983, donde explicaba el funcionamiento de este código.

Actualmente, muchas compañías de software trabajan con puertas de acceso en los sistemas que desarrollan. Esto representa una amenaza potencial, si se considera el hecho de que alguno de sus programadores puedan ser despedidos.

"ULTRIX System Management II: Security and Performance Management"
Educational Services, Digital Equipment Corporation

"Rogue Programs: Viruses, Worms, and Trojan Horses"
Lance J. Hofman
Van Nostrand Reinhold

"Computer Viruses"
Jonathan L. Mayo
Windcrest

"UNIX System Security"
Rik Farrow
Addison Wesley

Electronic Information System Group (EISG)
"Introduction to Personal Computer Security"
Digital Equipment Corporation

Análisis y diagnóstico 6

Existen procesadores de textos, hojas de cálculo, shells, y en general programas de aplicación que están hechos para realizar una función específica, sin embargo el mecanismo de funcionamiento puede variar entre programas del mismo tipo. De la misma manera los diferentes tipos de virus de computadoras son diseñados para reproducirse, activarse y ocultarse de diversas maneras.

Existen virus que funcionan almacenándose en el área de arranque del sistema, algunos virus infectan los programas ejecutables, unos más utilizan los manejadores de dispositivos (drivers, demons), otros funcionan estando residentes en memoria y algunos permanecen almacenados en la memorias CMOS de configuración de los sistemas.

Cualquier técnica de infección que utilicen los programas nocivos tiene ventajas y desventajas. Algunos métodos de contaminación son seleccionados porque es más difícil detectarlos, sin embargo esto provoca que el código del virus se incremente. Otros procedimientos pueden ser enfocados para desarrollar virus de código compacto, pero esto limita su capacidad para producir deterioro.

Estructura de un programa nocivo

Los virus de computadora son a fin de cuentas programas. Para que surtan algún efecto tienen que estar activos, es decir, deben ejecutarse y esta ejecución debe pasar más o menos inadvertida para el usuario.

Para que sea exitoso un virus debe incluir, al menos, las siguientes partes:

- Uno o varios mecanismos disparadores
- Uno o varios sistemas de ocultamiento
- Uno o varios sistemas de reproducción y contagio
- Una o varias misiones que cumplir
- Un sistema de control interno

Dado que son varias las actividades que el virus debe llevar a cabo, como reproducirse, cumplir su misión y protegerse; puede haber más de una forma de funcionamiento y éstas pueden originarse en un mismo evento o tener cada una uno o varios eventos que la originen.

Llamaremos mecanismo disparador, al que liga al evento externo con alguna actividad del virus. Los mecanismos disparadores pueden tomar diversas formas, desde algunas muy simples, como utilizar el nombre de algún programa o procedimiento catalogado de uso frecuente, hasta algunos casos en que los virus interceptan funciones internas de los sistemas operativos para detectar el momento y la forma en que deben actuar. El estudio de estos mecanismos puede valer la pena, ya que a menudo contienen ideas útiles para propósitos constructivos. Los mecanismos de disparo pueden incluir una gran variedad de tiempos, usos y condiciones lógicas, quizá generados pseudo-aleatoriamente desde una gran variedad de algoritmos.

Los sistemas de ocultamiento son muy interesantes y presentan también gran variedad de enfoques. Existen los que aprovechan las prioridades o tipos de archivo para no aparecer en las listas de programas, otros utilizan regiones marcadas como dañadas en los discos magnéticos, en

los sistemas multiusuario algunos se ocultan bajo la clave de un usuario inexistente o inactivo, otros se incorporan dentro de algún programa perfectamente legítimo, y algunos más imitan mecanismos encontrados en la naturaleza reproduciéndose más aprisa de lo que sus predadores los pueden destruir y hay aún virus que se reconstruyen cuando son dañados. Otros más utilizan métodos de encriptamiento y desecriptamiento para disminuir el riesgo de ser identificados.

Los sistemas de reproducción y contagio caen en general en dos grandes categorías: los que utilizan las herramientas del sistema para reproducirse (en forma similar a los virus biológicos, que son capaces de aprovechar los mecanismos de reproducción de las células), y los que incorporan sus propios medios.

Ambos enfoques presentan características diferentes y llevan también a maneras diferentes de combatirlos. En ambos casos hay, sin embargo, un aspecto importante, y éste es la tasa de reproducción que se puede lograr y la tasa de supervivencia de las copias generadas. Una reproducción demasiado frecuente puede ser detectada con mayor facilidad que una de carácter esporádico; sin embargo, dada la naturaleza geométrica del crecimiento del número de copias de virus, la primera tiene generalmente mayores perspectivas de ser exitosa, especialmente si la supervivencia del virus es baja. Una manera de incrementar la tasa de reproducción es hacer programas "portadores" del virus. Estos programas son en el fondo, una clase de Caballos de Troya, ya que bajo la apariencia de un programa útil sirven en realidad de vehículo para inocular más máquinas. Los contagios pueden incluir variaciones tales como "colocarse al principio del archivo", "colocarse al final del archivo", "colocarse en el sector de boot", "colocar 23 instrucciones al principio" "colocar 25 instrucciones antes de la mitad del programa", etc.

En cuanto a las misiones que cumplir, estas son tan variadas como la imaginación. Algunas no pasan de ser una broma en la que no se producen deterioros apreciables a los sistemas y simplemente despliegan imágenes o mensajes en el monitor, en tanto que otras lo que buscan es más el deterioro que la diversión, como borrar archivos, degradar el rendimiento del sistema, cambiar el quinto byte por el séptimo del programa, cambiar el byte 25 al 27 por el séptimo programa que encuentre, ver los datos de los archivos con extensión 123 y modificar la primera celda de la hoja electrónica, etc.

Consideremos como ejemplo el siguiente caso:

Un editor de texto es infectado por un virus de computadora. Cada vez que el editor es usado, desarrolla su funcionamiento normal pero también busca programas ejecutables y los infecta. Cuando se corran estos nuevos programas infectados, desarrollarán su tarea habitual y posteriormente intentarán infectar otros programas. Este proceso continua y los programas infectados pueden incluso pasar a otras computadoras al hacer respaldos de ellos y copiarlos en otro sistema. Finalmente el 1º de enero de todos los años, el programa infectado que se encuentra activo se dedica a borrar archivos aleatoriamente.

En el ejemplo anterior podemos identificar cada una de las partes que conforman un virus.

El mecanismo disparador sería la fecha del sistema cuando coincida con el 1º de enero. El sistema de ocultamiento se hace al añadirse al código de los programas ejecutables, de esta manera no se visualizan en el directorio archivos extras. El sistema de reproducción se logra al hacer copias del virus en otros programas ejecutables. La misión que cumple este virus es borrar

archivos aleatoriamente. Y el mecanismo de control es el encargado de coordinar cada uno de los elementos anteriores.

Un virus de computadora puede ser fácilmente descrito en un pseudocódigo. Fred Cohen planteó algo similar por primera vez en una conferencia que dictó en la Universidad del Sur de California en enero de 1986.

La notación simbólica que utilizó Cohen para explicar el funcionamiento del virus es muy simple, el símbolo := significa "por definición", el símbolo : etiqueta un enunciado, el símbolo ; separa enunciados, el signo = es utilizado para asignación o comparación, el carácter ~ significa negación, los símbolos { y } agrupan una secuencia de enunciados.

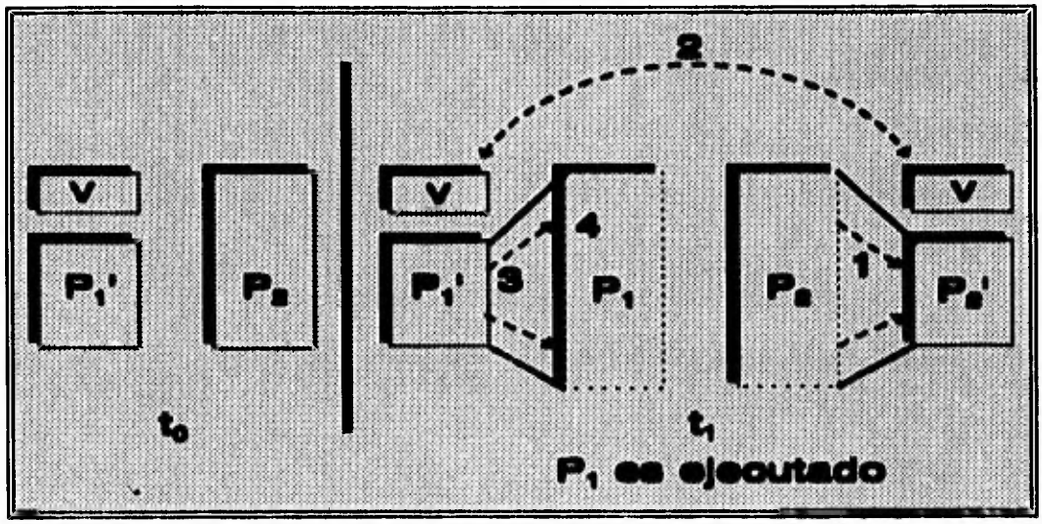
```
program virus :=
        { 1234567;
        subrutina infecta-ejecutable :=
                { loop:
                        archivo = seleccionar-archivo-ejecutable-aleatoriamente;
                        if primera-linea-del-archivo = 1234567 then
                                goto loop;
                        agregar este código al archivo;
                }
```

```
subrutina hacer-deterioro :=
        { while archivos > 0
                archivo = traer-archivo-aleatorio;
                borrar archivo;
                exit;
        }

subrutina disparo :=
        { if fecha = 1º de enero then
                return true;
        else
                return false;
        }

main-program :=
        { infecta-ejecutable;
        if disparo then
                hacer-deterioro;
                goto next;
        }
        next:
        }
```

Al hacer pequeñas modificaciones al esquema anterior, una gran variedad de virus pueden ser creados. Por ejemplo la siguiente variante del virus compresión de Cohen:

```
program virus-compresion :=
        {01234567;
        subroutina infecta-ejecutable :=
                { loop:
                archivo = seleccionar-archivo-ejecutable-aleatoriamente;
                if primera-linea-del-archivo = 01234567 then
                        goto loop;
(1)             comprimir archivo;
(2)             agregar virus-compresion al archivo;
                }

main-program :=
        { if tiene-permiso-de-ejecucion then
                infecta-ejecutable
(3)     descomprimir el resto de archivo en archivo-temporal;
(4)     ejecutar archivo-temporal;
        }
}
```

El proceso de funcionamiento de este virus podemos dividirlo en cuatro pasos, que están marcados en el código con los digitos 1, 2, 3 y 4.



Virus compresión.

Al tiempo t_0 tenemos al programa P_1', que es la versión infectada del programa P_1, y un programa P_2 que nunca ha sido infectado con algún virus. Si al tiempo t_1, el programa P_1 es ejecutado, los siguientes pasos serán ejecutados:

1. El programa P_2 se comprime en P_2' usando una rutina de compresión de archivos estándar.

2. V ataca a P_2' y se coloca al principio del mismo.

3. P_1' se descomprime en el original programa P_1.

4. El programa original P_1, es ejecutado normalmente.

Existe un par de aspectos que se deben considerar en la compresión de virus. Uno es que el tamaño de P_2' es aproximadamente 1/2 del tamaño del original de P_2. Esto es, las rutinas de compresión de archivos salvan al rededor del 50% del espacio que los archivos ocupan normalmente.

El efecto real es que si diseminados el virus en el sistema, podemos desocupar la mitad del espacio de nuestro disco. El costo que se tiene por salvar el espacio es que cuando se ejecutan los programas, estos tienen que ser descomprimidos, lo que tomará tiempo extra. Así la implementación para la compresión de virus esta dada por la relación tiempo/espacio. Los deterioros que causa este virus compresión es afectar el rendimiento del sistema, pues se consumen recursos al hacer la compresión y descompresión de archivos, y el usuario no lo notará hasta después de cierto período cuando observe que al correr sus programas consumen más tiempo de lo normal. Este esquema es similar a un virus biológico que cuando infecta otro organismo altera su funcionamiento normal e inicialmente no se detectan las anomalías que produce.

Ahora, si tenemos que defendernos contra un ataque de virus, podríamos verificar el cambio en el tamaño de archivos, sin embargo, agregando los apropiados bytes nulos en el programa infectado podemos hacer que el programa original e infectado tengan exactamente el mismo tamaño. Como se ve esta defensa tiende a fallar.

Podemos utilizar como defensa una simple suma de comprobación (Checksum) para indicar los cambios en un archivo. Una simple suma de comprobación esta formada por la suma de todos los bytes del archivo o también se puede implementar con la suma de bytes de algunos módulos seleccionados del archivo. Podemos efectuar un checksum y compararlo con otro en una fecha posterior y comprobar si hubo algún cambio. Si con anterioridad vimos que podemos colocar bytes nulos que mantengan el tamaño del archivo, para este caso podemos colocar los bytes apropiados para hacer que el checksum del archivo infectado coincida con el checksum del archivo original. Un simple checksum es fácil de eludir.

Lo mismo sucede para los códigos CRC (Cycle Redundance Code). Un código CRC es tipicamnte usado para identificar cambios sobre un disco debidos a errores aleatorios.

En la 7ª Conferencia sobre Seguridad en Computadoras realizada en septiembre de 1984, Fred Cohen participó como conferencista en una ponencia titulada "Virus de Computadoras, Teoría y Experimentos", en su participación decía: *Un Virus puede ser definido como una secuencia de símbolos que, son interpretados en un ambiente dado y que provocan otra secuencia de símbolos que pueden estar contenidos en el ambiente a ser modificado (posiblemente evolucionado).*

Hay muchos aspectos en esta definición que son válidos ahora. Primero, un virus es como cualquier otro programa que está formado por una secuencia de símbolos. Una instrucción de máquina que es parte del programa de un virus, generalmente no es distinguible de la instrucción de máquina que contiene un programa. Segundo, un virus puede causar otra secuencia de símbolos que se convertirá posiblemente en un virus evolucionado (por ejemplo los virus son

capaces de auto-reproducirse). Tercero esta auto-reproducción ocurre cuando la secuencia de símbolos está siendo interpretada (un virus debe estar activo en memoria principal para reproducirse).

Así mismo, es posible definir a un virus de computadora en un amplio campo de tipos de ataques a sistemas de cómputo; se puede ver como una pieza de código con dos características:

1. *Al menos parcialmente una capacidad automática para reproducirse.*

2. *Un método de transferirse con sus dependencias y habilidades para atacar él mismo otras entidades de computo (programas, sectores de disco, archivos de datos, etc.), que lo mueven entre estos sistemas.*

Por otro lado, otras definiciones plantean que para que un programa se pueda considerar como virus debe tener las siguientes características:

1. *La modificación del software que afecta no debe ser para el programa virus y su estructura de encadenamiento debe estar dentro de estos otros programas.*

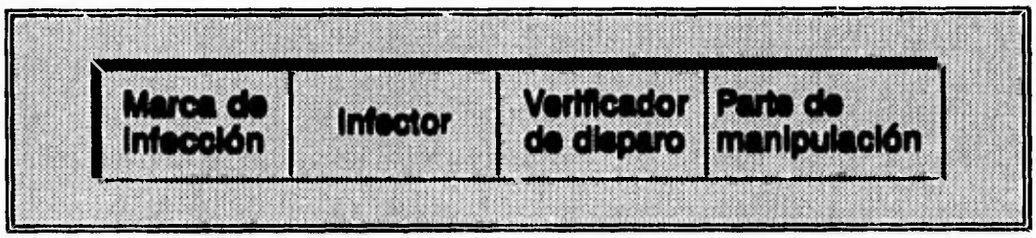
2. *Capacidad para ejecutar las modificaciones de los programas.*

3. *Capacidad para reconocer una modificación desarrollada sobre un programa.*

4. *Habilidad para prevenir nuevas modificaciones a un programa que ya alteró.*

5. *Modificar software asumiendo las características 1 a la 4.*

De esta definición se puede plantear una estructura general de un virus, que es mostrada en la siguiente figura:



Un virus no necesita estar "modularmente estructurado" como se indica en la figura, sus partes no deben estar en un orden específico. Sin embargo, al dividir un virus en estas áreas funcionales es más fácil explicarlo y manejarlo. La *marca de infección* es una parte opcional que sirve para identificar programas que ya han sido infectados. En el caso de virus que incrementan el tamaño del programa que afectan, esta marca previene múltiples reinfecciones que podrían resultar en un incremento notable del tamaño del archivo. La parte denominada *infector* es la pieza de código que identifica el archivo a dañar y lo "infecta" con una posible copia modificada del virus. El *verificador* de disparo es opcional y esencialmente es utilizado para controlar algunos aspectos del comportamiento del virus (normalmente la *parte de manipulación*) basados en alguna condición interna o externa como la fecha, hora, número de veces que el virus se

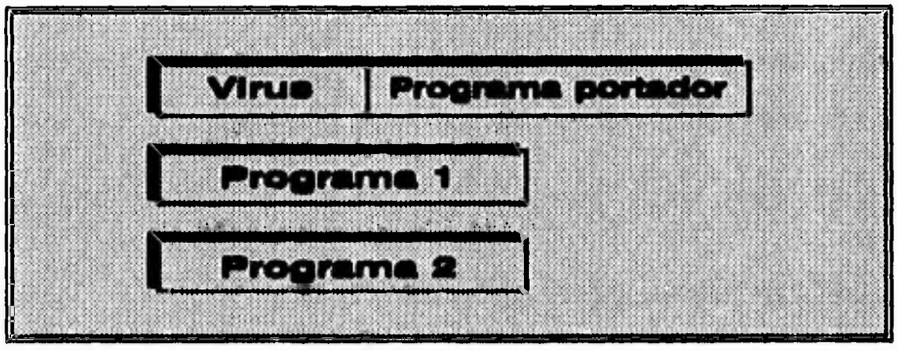
reproduzca, etc. La *parte de manipulación* es también opcional, es la parte del virus que produce el deterioro, está diseñada para realizar las tareas que el creador del virus desee, que van desde escribir mensajes en la pantalla hasta hacer que se caiga el sistema.

Mecanismo de ataque de los virus

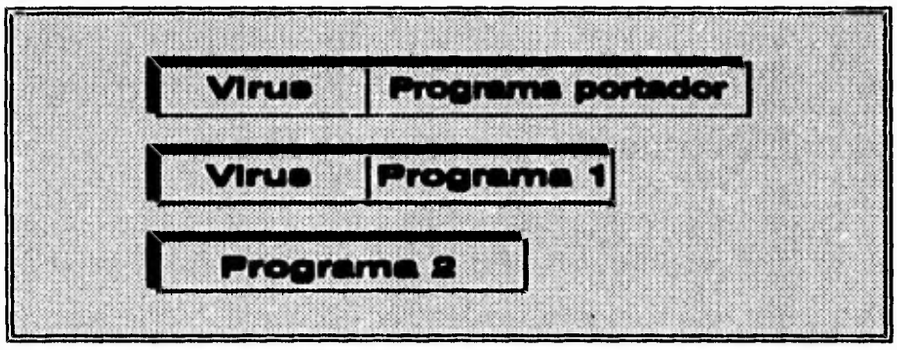
Aunque existe indudablemente un gran número de variantes, los virus conocidos pueden ser categorizados en términos de dos propiedades fundamentales: *¿cómo infectan a otros programas?*, y *¿donde se almacenan?*. En términos de cómo infectan a otros programas los virus conocidos pueden ser agrupados en tres clases generales: de **sobreescritura**, de **reemplazo** y de **adición** y otro que puede tener alguna de las tres anteriores pero con la característica adicional de ser **mutante**.

Virus de sobreescritura

Este tipo de virus sencillamente sobreescribe en los primero N bytes de un archivo ejecutable el código del virus y es relativamente fácil de crear. Consideremos un programa portador (por ejemplo un Caballo de Troya) que contiene un virus y es introducido en un sistema de cómputo, existen además dos programas sin infección, este estado se muestra en la figura siguiente:

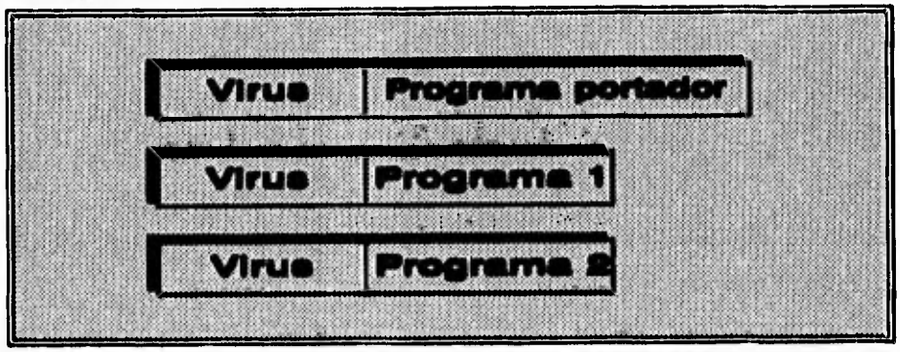


Cuando el programa portador es ejecutado, la parte correspondiente al virus es ejecutada primero e infecta al programa 1, entonces el resto de programa portador es ejecutado dando la apariencia de un funcionamiento normal. El estado de estos programas se vería así:



La primera parte del programa 1 ha sido reemplazada por el código del virus. Cuando el programa 1 es ejecutado, se efectuarán las instrucciones del virus, el cual intentará infectar otros programas por ejemplo el programa 2, a continuación se intentará ejecutar seguramente sin éxito

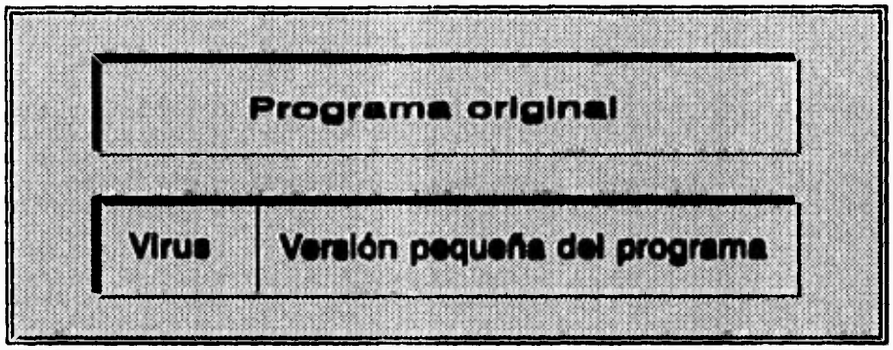
la parte restante del programa 1 dando la apariencia de que está dañado.



Ahora el programa 1 y el programa 2 son incapaces de desarrollar sus funciones originales pero sí son aptos para producir otras infecciones. Nótese que el programa portador también contiene el virus y puede infectar otros programas siempre que sea ejecutado. Si el usuario sospecha la presencia de un virus, pensará que el problema existe en el programa 1 ó programa 2 y no en el portador, por lo que el problema se seguirá dando mientras se ejecute este último.

Virus de reemplazo

Estos virus operan al reemplazar un programa con un equivalente funcional que contiene al virus. El archivo de reemplazo no debe ser mayor que el original. Por ejemplo un programa original escrito en un lenguaje de alto nivel también podría ser escrito en ensamblador, así como un programa virus. El código objeto de la nueva versión junto con el código del virus, podría ser más pequeño que el programa original. La figura siguiente ilustra este tipo de ataque.

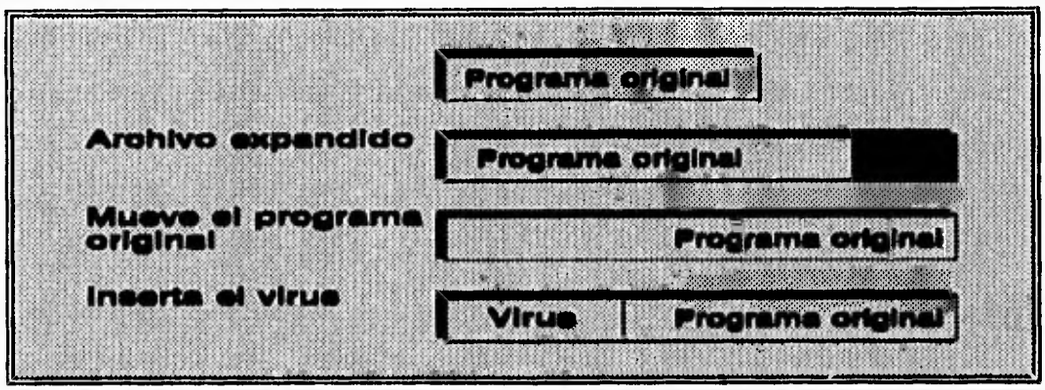


La limitante de estos virus es que únicamente pueden infectar programas específicos, pero tienen la característica de que su detección es muy difícil.

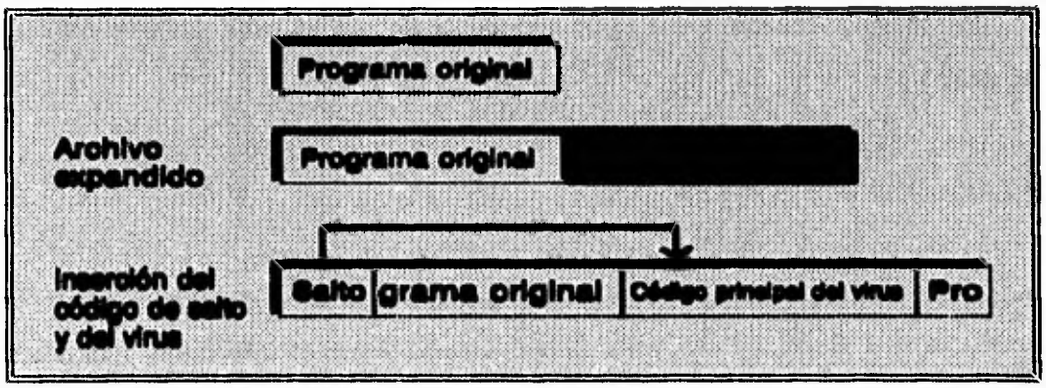
Virus de adición

Estos virus son los más peligrosos porque pueden infectar muchos programas sin afectar su funcionalidad. Ellos operan agregando su código al programa que infectan ya sea incrementando el tamaño del archivo o escribiéndose en un área no usada. Russell Davis identifica dos formas para implementar este tipo de virus.

En el primer esquema, el virus primero incrementa el tamaño del archivo a infectar y después mueve el código original dejando un espacio al principio para insertar el código del virus.



En el segundo esquema, el virus salva primero algunas instrucciones del programa a infectar, las reemplaza con una instrucción de salto (tal vez también con una marca de su presencia) y se agrega al final del archivo. Cuando se corre un programa infectado, la instrucción de salto ejecuta el código del virus. Una vez que termina de realizar su misión, el virus procede a ejecutar al programa original.



Una variación para este esquema es reemplazar algunos bytes en la parte media del archivo a infectar, lo cual es más difícil porque es necesario identificar un punto en el que el programa no pierda su funcionalidad.

Virus mutantes

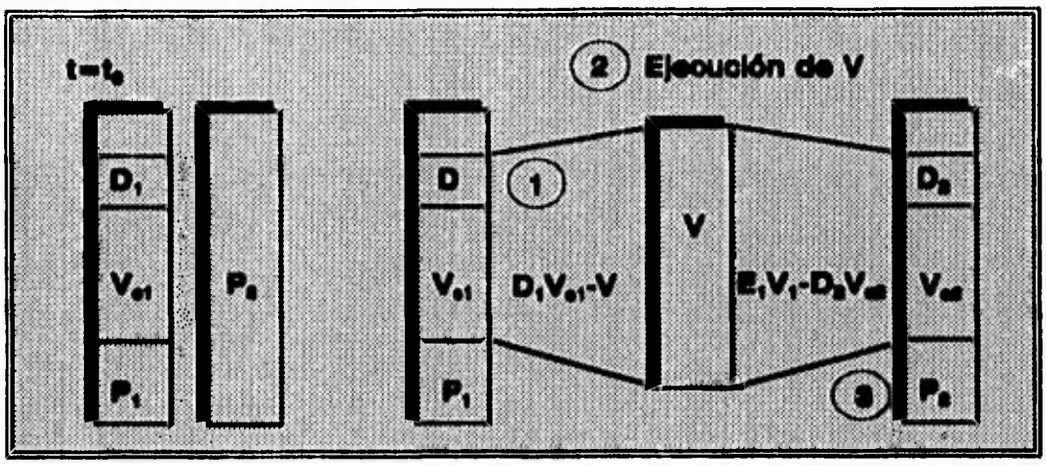
Se basan en un planteamiento diferente para ser desarrollados, su característica principal es que las nuevas generaciones producto de una versión original no son iguales, por los que los procesos de detección y erradicación son aún más complicados. De un mismo código origen del virus, dependiendo de la implementación, se pueden obtener N-número de variantes. A continuación un ejemplo del funcionamiento de un virus mutante:

Al tiempo $t=t_0$, se tiene un programa P_1 que está infectado con un virus y un programa limpio P_2. Si al tiempo $t=t_1$ se ejecuta P_1, los siguientes pasos se llevarán a cabo:

1. D_1, un algoritmo desencriptador, desencripta a V_{c1}, la versión encriptada de V, en el original V.

2. V es ejecutado.

3. V selecciona un algoritmo diferente de encriptamiento E, se reencripta a si mismo con E_2 y da V_{c2} y coloca el algoritmo de desencriptamiento D_2 (diseñado para desencriptar sistemas encriptados con E_2) en el programa P_2 junto con V_{c2}.



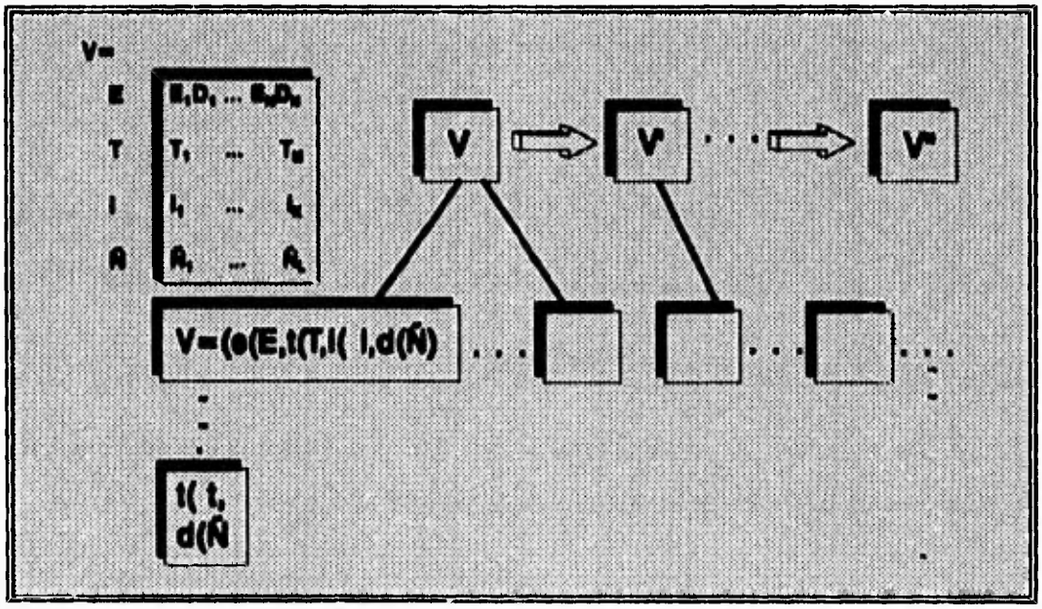
Virus mutante

Sin tomar en cuenta la probabilidad de que las secuencias aleatorias coincidan, las secuencias comunes de bytes no existen entre el par P_1 y P_2 para este virus.

A continuación se presenta un ejemplo mas específico. Suponiendo que V_{c1} fue encriptado con un algoritmo DES (Data Encryption Standard). Cuando se ejecuta P_1, D_1 desencripta a V_{c1} usando un algoritmo DES para obtener un virus original V. Entonces el virus V cambia al sistema de encriptamiento RSA y se encripta con algoritmo RSA así mismo, dejando a V_{c2}. Después coloca un algoritmo de desencriptamiento D_2 empleando la técnica RSA junto con V_{c2} en el programa P_2. Debido a este punto es muy difícil escribir un programa que determine si un programa es evolución de algún otro.

A continuación se tiene una nueva versión de un virus mutante, donde V tiene un conjunto completo de algoritmos de encriptamiento ($E_1...E_m$), algoritmos de desencriptamiento

($D_1...D_m$), mecanismos de disparo ($T_1...T_m$), mecanismos de infección ($I_1...I_m$) y mecanismos de las misiones que cumplir ($\tilde{N}_1..\tilde{N}_m$).



Esquema de un virus mutante complejo

Se han mencionado dos algoritmos de encriptamiento, pero existen muchos más que pueden ser automáticamente generados, además de los generados por los propios programadores.

Un virus mutante con una gran variedad de algoritmos de encriptamiento y desencriptamiento y también con una gran variedad de mecanismos de disparo, deterioro e infección se extenderá entre los programas en un sistema como lo hacen los virus mutantes sencillos, con la característica que ésto se reemplazará así mismo con un subconjunto de

algoritmos de encriptamiento y desencriptamiento, condiciones de disparo, mecanismos de infección y mecanismos de las misiones que cumplir. Esto se repite todo el proceso, diseminándose y evolucionando por siempre, y reemplazándose con los diferentes subconjuntos, etc. Por lo que en un periodo se tendrá una gran cantidad de diferentes variaciones, todas muy diferentes unas de las otras, en términos de la secuencia de bytes que aparezcan en el sistema, y se habrán diseminado a través de los sistema de cómputo y de respaldo.

Diagnóstico de programas nocivos

Si se tiene un virus de computadora al que no se puede detener su entrada al sistema y está diseñado para infiltrar los sistemas de manera que los paquetes antivirus y los métodos de seguridad para la computadora no lo detectan. Los síntomas que indican la presencia de algún virus pueden incluir cualquier conducta que la computadora y el software son capaces de realizar bajo cualquier otra circunstancia. Un virus puede causar diferentes síntomas sobre las computadoras donde el hardware y el software son idénticos, aún con diferente configuración sobre cada máquina. Se mencionan a continuación algunos ejemplos:

Una computadora Macintosh aparenta trabajar normalmente, e imprime normalmente cuando utiliza papel de formas continuas, pero no puede funcionar igual cuando utiliza hojas individuales. El problema se resolvió después de dos días cuando se quitó el virus WDEF del sistema.

Una misma versión del virus de Jerusalem que contamina una red Novell podría establecerse en un nodo de otra red Novell con diferentes versiones, o provocar un error de comunicaciones.

El virus Stoned (que se aloja en el BOOT de DOS y la tabla de particiones) no causa problemas sobre los discos duros con versiones de DOS 3.3 o mayores, pero puede dañar la Tabla de Particiones cuando infecta máquinas con versiones de DOS menores, o cuando la máquina no ha sido actualizada con la actual versión de FDISK.

Estos son sólo algunos ejemplos que indican la presencia de un virus. Sin embargo, son muchas las ocasiones en donde no es posible distinguir si se trata de una falla de software o hardware. Se ha vuelto muy común que usuarios de computadoras encuentren bugs (mal funcionamiento) en los programas comerciales. Esto se debe a que las exigencias del mercado orillan a las empresas a liberar productos que no han sido probados totalmente. Dadas las circunstancias, actualmente es considerado imposible eliminar todos los bugs en programas que en su mayoría se componen de decenas o centenas de miles de líneas.

Algunos bugs de los programas reflejan los síntomas que en algunas ocasiones son señales de una infección de virus:

- Los programas desarrollan parte de su función incorrectamente.

- Los programas pueden causar que la computadora se bloqueé.

- Los archivos son dañados o no son salvados correctamente.

Existen dos maneras de descubrir a un virus: Al utilizar un programa de protección o diagnosticar los síntomas de la presencia de este. Los síntomas con los que se puede sospechar la presencia de un virus son:

- La memoria RAM disminuye sin haber cargado algún programa.
- El disco realiza accesos o se enciende el indicador de la unidad sin existir alguna causa.
- El sistema se vuelve muy lento al estar trabajando.
- Los programas cuando se ejecutan despliegan mensajes de error extraños.
- El sistema operativo despliega mensajes de error inesperados
- Los tamaños de los archivos cambian sin motivo.
- El número de archivos en el directorio del disco cambia sin razón.
- El borrar, renombrar o copiar archivos toma mucho tiempo (aunque de hecho, esto no se aplica en sistemas multiusuario ya que esta situación es común bajo carga excesiva de trabajo).

- El teclado envía caracteres extraños al monitor o de repente no trabaja.

- El sistema se pierde sin motivo.

Aunque estos mismos síntomas pueden deberse a una falla de los circuitos o programas del sistema. No todos los problemas que presenta un equipo de cómputo se deben a infecciones de virus, algunos son productos de malas conexiones, discos defectuosos, archivos defectuosos, etc. La mejor manera de verificarlo es mediante la ejecución inmediata de programas de diagnóstico del sistema, si estos indican que no existe ningún problema, entonces hay que proceder a utilizar los detectores y erradicadores de virus.

Es por esto que cuando se presenta algún síntoma de la posible existencia de un virus no se debe atribuir inmediatamente la culpa a ellos. Es recomendable primero agotar todas las posibilidades que involucren una falla en el funcionamiento de los programas, en el hardware o en las instalaciones.

A continuación se presentan algunos síntomas que demuestran la presencia de algún virus en sistemas con DOS.

Síntomas comunes de los virus del sistema operativo DOS

- La actividad del sistema se vuelve lenta.

- Los caracteres de la pantalla caen a la parte baja del monitor.

- Se efectúa actividad persistente sobre el drive de disco flexible aún cuando el disco esté protegido contra escritura.

- Los programas ejecutables cambian de longitud.

- El espacio disponible en disco disminuye rápidamente.

- El número de sectores dañados se incrementa.

- El monto de RAM disponible disminuye (En el ambiente de DOS se utilizan los comandos CHKDSK o MEM para monitorear la RAM).

- El mapa de memoria muestra programas residentes de origen desconocido (comando MEM para DOS).

- Los programas que trabajaban bien funcionan anormalmente o fallan sin razón.

- Los programas producen errores cuando antes no lo hacian.

- Los programas generan mensajes no documentados.

- Los archivos desaparecen misteriosamente.

- Los archivos son reemplazados por objetos de origen desconocido o son reemplazados con datos raros.

- Los nombres, extensiones, fechas, atributos y datos de los archivos o directorios se modifican sin la intervención directa de los usuarios.

- Aparecen archivos o directorios de origen desconocido.

- Se detectan cambios a los objetos estáticos (archivos) del sistema. Los cambios detectados en los objetos dinámicos no son necesariamente indicaciones de infecciones virales.

- Cambiar el nombre del volumen del disco.

- Marcar sectores dañados en áreas no usadas del disco disminuyendo paulatinamente su capacidad.

- Interferir con la operación de programas residentes en memoria RAM.

- Eventualmente cancelar "BOOT", "FAT" y sectores del directorio.

- Provocar imágenes molestas en el monitor o enviar mensajes.

- Borrar archivos.

- Bloquear buffers de manera no se permita la entrada y salida datos.

- Llenar de basura la memoria de la computadora.

- Destruir directorios de discos.

- Dar formato a los diskettes y discos duros.

- Reinicialización de la computadora.

- Redifinir teclas.

- Bloquear el teclado.

- Modificar la información en programas o archivos.

Detección de un virus de computadora

Es posible, y esto ocurre en muchas situaciones, que los síntomas que se presentan no sean suficientemente claros. Es por esta razón que lo más sencillo es usar los programas de diagnóstico. Sin embargo, en este punto es importante considerar sobre la capacidad de los diagnosticadores. ¿Es posible detectar un virus? Un programa de defensa ideal debe evitar resultados tales como los siguientes:

- Siempre corra sin proporcionar algún resultado para cualquier caso.

- Tenga un número infinito de falsos positivos (que identifique como virus a un programa cuando no este no lo es).

- Tenga un número infinito de falsos negativos (que no identifique a un programa como virus cuando este lo es) o lo que es peor, que tenga una combinación de los tres problemas anteriores.

Es posible asegurar que ningún ativirus, programa de diagnóstico o de defensa desarrollado a la fecha, puede evitar alguno de los tres casos anteriores para una situación en particualar.

Ahora bien, ¿podemos encontrar el orígen de las infecciones ocasionadas por algún virus? la respuesta es sí, siempre y cuando el programa nocivo funcione con un patrón constante de comportamiento. Por otro lado si se enfoca esta pregunta a virus que evolucionan en su código, no es posible determinar en forma general, todas la evoluciones de los virus conocidos, al menos no sistemáticamente con un programa. Llegará el día en que exista un deterioro pequeño en el sistema, un archivo será borrado o quizá algunos bytes serán cambiados en un archivo de datos seleccionado al azar. Suponiendo que se identifican esas anomalías en forma correcta y se llega a la conclusión de que es un caballo de troya el que esta causando el deterioro. Se busca en todo el sistema y no se encuentran otras copias del programa nocivo y así se piensa que todo está a salvo. Pero en pocos minutos dos cosas empiezan a estar dañadas, después tres, después cuatro, después cinco. Después de algunos días transcurridos cientos de miles de esas cosas estarán

sucediendo sin ningún orden.

¿Entonces que se puede hacer?, ¿ir por un respaldo?, ¿restaurar lo de una semana anterior, buscar los patrones diferentes y tal vez no encontrar ningún programa nocivo?, o ¿regresar a restaurar dos semanas anteriores, o tres o cuatro o tan lejos como se quiera? No hay forma de estar seguro de lo que está sucediendo. Este punto indica los problemas con virus mutantes, con la búsqueda hacia atras de los virus a través del sistema. Varias interrogantes usualmente se presentan al respecto. Por ejemplo: ¿Cómo se puede estar a salvo de esas reinfecciones en el mismo programa repetidamente?. No hay forma de estar a salvo pero existen una gran variedad de formas para limitar las infecciones.

Recuperación de un ataque de virus

La recuperación de un ataque de virus es una de las labores más difíciles de desempeñar, muchas veces no es posible determinar cuanto daño se ha causado en el sistema y si el respaldo que se tiene es confiable. Existen cuatro etapas en el proceso de infección y contagio de un virus:

VIRUS

MEMORIA PRINCIPAL	
Daño y recuperación	
- Afecta las aplicaciones que se están efectuando	* Simple de remover * Daño limitado si se contiene

ALMACENAMIENTO LOCAL FIJO	
Daño y recuperación	
- Puede infectar todas las aplicaciones almacenadas - Daño potencial a la información	* Requiere de 1 a 5 horas/hombre para ser eliminado * Daños moderados si se atrapa a tiempo

SISTEMA DE ARCHIVOS COMPARTIDOS	
Daño y recuperación	
- Programas de servicio - Compiladores - Editores - Herramientas - Aplicaciones compartidas - Comunicaciones - Archivos de sistemas	* Infección diseminada por el sistema * Puede causar un daño sustancial * Recuperación compleja

MEDIOS DE ALMACENAMIENTO REMOVIBLES	
Daño y recuperación	
- Discos flexibles - Discos de una sola grabación y varias lecturas - Respaldo en cintas - Comunicaciones - Archivo de sistemas	* Se dispersa ampliamente * Es difícil de localizar * Fácil de pasar por alto * Archivado por períodos considerables * Puede reintroducir una vieja infección
Extremadamente difícil de recuperar Los medios de almacenamiento pueden no ser controlados Probabilidades de reinfección muy alta	

Una vez que se ha detectado la presencia de un virus, es necesario detener toda actividad sobre el sistema o los sistemas que se presumen infectados y proceder a su restauración.

1. Es necesario apagar completamente la computadora. En algunos casos, los mecanismos de "reinicialización del sistema" provocan el disparo en el ataque del virus o simplemente no limpian al virus de memoria (tal como el Ctrl–Alt–Del de DOS en las computadoras personales). Por esta razón, use únicamente el botón de apagado de la computadora.

2. Encender de nuevo la computadora utilizando los medios de distribución originales del sistema operativo (diskette, cinta, cartucho, etc.), con esto se elimina la posibilidad de que algún virus esté activo en memoria.

3. Ejecutar programas de diagnóstico, búsqueda y erradicación de virus. Estos deben ser también originales. **Por ningún motivo se debe ejecutar aplicación alguna del sistema infectado.**

4. En los medios de almacenamiento fijo, si no tenía un respaldo reciente y tiene necesidad de recuperar la información, se deben respaldar todos los archivos que exclusivamente sean de datos en medios de almacenamiento no "booteables". Si se tiene necesidad y/o tiempo, revisar todos los programas ejecutables haciendo comparaciones con los originales (cheksums, tamaño del archivo, etc.).

5. La recomendación es reinicializar (o formatear) todos los medios de almacenamiento fijo, desde el nivel más bajo si es posible. Sin embargo, si se siente satisfecho con sus respaldos y chequeos del sistema, puede omitir este paso.

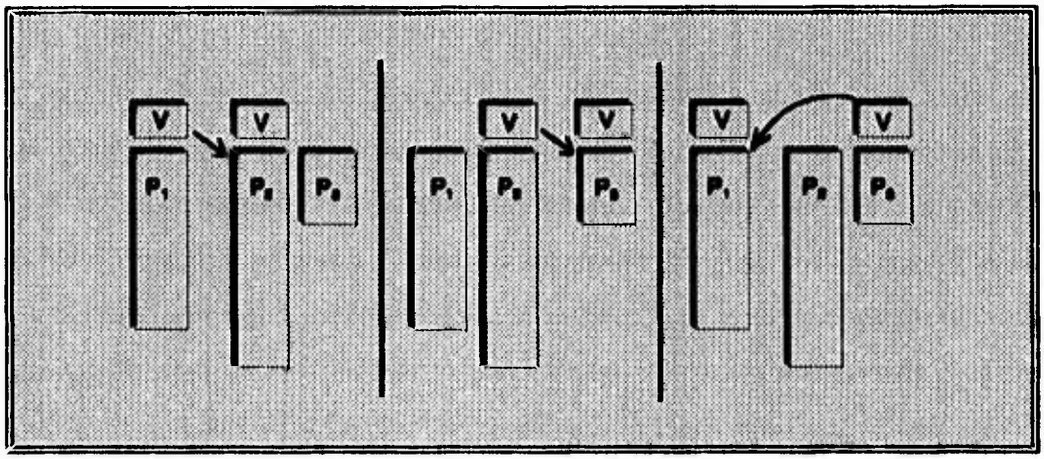
6. Restablecer los archivos de datos en el disco.

7. Arrancar la computadora y mantenerla en operación. Tratar de reproducir las circunstancias por las cuales se detectó al virus y observar resultados. Si se concluye que se ha herradicado al virus, repita estos pasos para cada una de las computadoras que conforman el sistema.

Si cuenta con un buen respaldo del sistema infectado, se recomienda tratar de "capturar al virus" para ser identificado o desensamblado después. Buenas ideas (sobre todo benéficas) surgen del estudio de estos programas. Cuando se trate de una red de computadoras será necesario aislar primeramente al nodo servidor (o nodos servidores), realizar los pasos anteriores y así proceder, también de manera aislada, con cada uno del resto de los nodos de la red. La reconexión es importante una vez que se han revisado todos los nodos de la red; se deben conectar uno a uno probándolos hasta quedar satisfechos. Nunca conectar todos simultáneamente, pudiera ser motivo de reinfección masiva.

Otra situación que se presenta con mucha frecuencia es el tratar de eliminar un virus cuando el sistema está en operación. En la mayoría de los casos no se tiene éxito al intentar esta acción. A continuación la explicación a este problema a través de un ejemplo:

Se tienen dos programas P_1 y P_2 que están infectados, detectamos que P_1 está infectado y erradicamos el virus de éste, pero mientras tanto, P_2 es ejecutado y se infecta el programa P_3, No hay problema, encontramos que P_2 esta infectado y eliminamos el virus, pero mientras tanto alguien ejecuta el programa P_3, el cual reinfecta al programa P_1. Ahora estamos como al principio.



La reinfección o la erradicación de programas nocivos dependen del tiempo y la inversión que se le dedique a este problema. Esto sigue la teoría de las infecciones biológicas; básicamente el número de programas infectados se mide por la frecuencia de uso de los mismos dado el número de nuevas infecciones por unidad de tiempo. Si esto excede el promedio de cura, entonces lo más seguro es que siempre los sistemas se encuentren infectados. Pero si se presenta el caso contrario, es muy probable que el sistema quede protegido contra posibles ataques de programas nocivos (siempre que estos sean conocidos). Sin embargo, la realidad es que el

promedio de curas, salvo en casos excepcionales, siempre será menor que el de la del promedio de infecciones; esto se debe a que el costo de inversión para erradicar programas nocivos es muy alto.

Problemas en una red LAN y algunas soluciones

Los virus han afectado gravemente a las redes a lo largo de los últimos años y continuarán haciéndolo en el futuro. Sin embargo, este riesgo puede desaparecer si se toman ciertas medidas preventivas. En este aspecto, el punto de partida consiste en la formulación de una sólida estrategia global que posteriormente se aplicará con unos márgenes razonables de flexibilidad. No obstante, a pesar de reconocer la necesidad de disponer de protecciones frente a esta amenaza, todavía son pocos los usuarios que se han movilizado para salvaguardar con cierta eficacia sus redes. Los especialistas en seguridad afirman que no es fácil definir exactamente las razones por las que muchos usuarios no han comenzado a protegerse y, desde luego, la falta de conocimientos no constituye una excusa válida. Al respecto, muchos administradores admiten que los planes que tienen implantados en su empresa como la verificación una vez al año de los puestos de trabajo de las redes de área local para localizar posibles infecciones de virus, son totalmente inadecuados. Y entre aquellos que disponen realmente de estrategias antivirus, la mayoría reconoce que únicamente se pusieron en marcha después de haber sufrido un desastre de este tipo.

Todavía resulta más curiosa la carencia de una respuesta coherente por parte de los administradores de redes si se tiene en cuenta la inmensa cantidad de usuarios afectados por distintos virus en los últimos años.

Además, todo indica que la situación irá a peor. Los desarrolladores de estas plagas, cada vez más prolíficos generaron en promedio un virus nuevo cada dos días en 1993. Una tasa de crecimiento que se calculaba alcanzar una media de seis nuevos virus diarios para 1994.

Por otra parte, estos peculiares desarrolladores están consiguiendo dotar a sus criaturas de una progresiva sofisticación. Ultimamente se han introducido virus de enorme potencia que ocultan su existencia enmascarando los cambios producidos en las longitudes de los archivos que se listan en los directorios o desinfectandose a si mismos cuando se lee la memoria. Esta última técnica impide que la mayoría de paquetes antivirus puedan detectar los cambios ocurridos en un programa, impidiendo así detectar su presencia. En este panorama un tanto desolador, ¿qué pueden hacer los usuarios para proteger sus redes? Los especialistas en seguridad se basan en tres conceptos básicos: el valor de los datos, el presupuesto financiero y las soluciones.

El valor de los datos

El primero de ellos consiste en tomar verdadera conciencia del creciente valor de los datos almacenados en las redes de área local. Y esto es necesario porque, hasta hace poco, no se depositaba en las LAN's una confianza plena respecto al almacenamiento en ellas de los datos de verdadero valor para la empresa. Sin embargo, esta situación está cambiando drásticamente a medida que se generalizan los procesos de "downsizing".

Presupuesto financiero

El segundo aspecto esencial a considerar, consistirá en la consecución de presupuesto financiero para la adquisición de equipos antivirus. En este sentido, debe desarrollarse en plan que permita justificar, ante la alta dirección de la empresa, los gastos dedicados a la protección frente a los virus.

Finalmente, tras los pasos anteriores, se procederá a adquirir y poner en marcha una amplia variedad de herramienta de protección. Llegados a este punto, se hace necesario observar que ningún producto por sí solo puede ofrecer una salvaguarda total frente a todos los posibles virus que puedan atacar la red.

Para combatir el problema con eficacia, se requiere una combinación de varios productos, desde paquetes de muestreo y localización de encontrar archivos infectados por virus conocidos, a equipamiento software, o combinación software, hardware, que corre en los servidores de la red para impedir la ejecución de paquetes no autorizados.

Las medidas de seguridad que pueden conseguir aislar las redes locales de la posible contaminación de virus son similares a las que se toman en las redes de minis y grandes computadoras.

Entre ellas, destaca la necesidad de utilizar palabras clave de acceso, así como asignación a los usuarios de diferentes niveles de acceso a los recursos disponibles en función de sus necesidades afirman que entre ambas categorías se mantiene una diferencia fundamental: en los

centros tradicionales de sistemas de información no se permite el acceso de extraños que podrían montar una cinta y cargar programas en la computadora, mientras que el caso de las redes locales los usuarios pueden llevar diskettes y cargarlos, sin responsabilidad alguna, en las estaciones de trabajo.

Aunque este tipo de comportamiento se ignora con frecuencia, cuando queda afectada una red por la introducción de un virus cargado a través de un programa infectado, la perspectiva suele cambiar radicalmente. En realidad los diskettes compartidos y utilización de software de contrabando constituyen las principales fuentes de infecciones por virus en las redes locales.

El método más utilizado por los directivos de comunicaciones cuando buscan la aprobación de inversiones por parte de la alta dirección para medidas de seguridad antivirus, consiste en la utilización de ciertos recursos publicitarios: se suelen apelar al conocimiento de la infección de la red de alguna otra compañía para plantear la posibilidad de que ocurra lo mismo en la propia. A pesar de ello, por desgracia, frecuentemente no se consigue el soporte adecuado hasta que la propia red es atacada por el virus.

En cuanto a costos, ¿cómo puede justificarse la incorporación de medidas de seguridad en las redes? Al respecto, conviene recordar la cantidad de tiempo que puede estar fuera de servicio una red y lo que esto representa en costos debido a una circunstancia de este tipo. Por ejemplo, en el ataque del gusano "worm" a Internet en 1988, llegaron a quedar infectados el 7.2 por ciento de las 85 mil 200 máquinas de la red, es decir, 6 mil 200 estaciones de trabajo. Prácticamente toda la red tuvo que ser parada para comprobar el alcance de la contaminación.

Una vez conseguido el soporte financiero necesario, no habrá quedado todavía suficientemente claro, el camino hacia las redes libres de virus. Puesto que las redes pueden quedar infectadas de múltiples formas, se requiere una cierta combinación de métodos. Por ejemplo, además de los diskettes y los programas no autorizados, otro medio de introducción de virus consiste en el acceso vía red conmutada.

Un medio de conseguir disminuir el riesgo de infección por este camino consiste en disponer el servidor de comunicaciones de forma que se transfieran los archivos a una estación de trabajo no conectado a la red. De este modo, se pueden explorar los archivos antes de descargarlos a la red para comprobar si incluyen virus.

El método de aislamiento funciona para salida con red conmutada aunque, de permitirse que los usuarios se conecten por este medio, el problema será más difícil de resolver. En este caso, como medio de protección suele utilizarse un servidor de seguridad que requiere autentificación por palabra clave antes de que la persona que llama pueda introducirse en la red.

Soluciones

Existen en el mercado servidores de comunicaciones UNIX que restringen el acceso a las computadoras de la red con datos sensibles. Los usuarios que llaman al servidor deben introducir una palabra clave que pueda asignarse específicamente para cada nodo controlado para el producto, lo que permite conseguir protección para los diferentes segmentos que componen la red.

Existen otros que controlan el acceso a la red vía nodos de comunicación, acceso por líneas conmutada (vía modem) o conexiones directas, mediante la utilización de un equipo del tamaño de una tarjeta de crédito que presenta un código que va modificandose cada minuto. El servidor mantiene códigos para todas las tarjetas enviadas a los usuarios. Si alguno de ellos intenta acceder a los recursos de una red introduce su número de código presentado en el momento que se intenta el acceso. Si el número introducido no es igual al código almacenado en el sistema de administración, la entrada es denegada. Los dos servidores de seguridad comentados constituyen ejemplos de herramientas que impiden el acceso a usuarios no autorizados. Puesto que, en su mayoría, los virus únicamente se extienden cuando se procesa un programa, el simple bloqueo a programas no autorizados impide su difusión.

Existe una variedad de productos en el mercado que no hacen posible el proceso de ejecución de programas. Así, algunas compañías ofrecen la combinación de software/hardware y otras ofrecen soluciones basadas en software. Estas últimas, permiten que el administrador construya una lista de programas transferibles desde un disco local a la red. De este modo, no pueden cargarse otros programas no autorizados, como pueden ser los que traen los usuarios en diskettes. Si se intenta ejecutar un programa desde un nodo local de la red, se compara al archivo ejecutable con el de la lista de archivos autorizados residentes en el servidor. Esta tipo de producto compara el nombre del archivo, su tamaño en bytes, nombres del creador y fecha de creación de cada archivo antes de su ejecución. Normalmente, cualquier paquete infectado, incluso si está autorizado, presentará características diferentes a las de la copia limpia, y si la versión de la unidad local no es igual al archivo limpio, el programa no podrá ejecutarse en la red.

Otros fabricantes dan un enfoque diferente. Utilizan conjuntamente una tarjeta y software para controlar la ejecución de programas no autorizados. La tarjeta trabaja en colaboración con la correspondiente tarjeta para interfase de red (NIC por sus siglas en inglés de Network Interfase Card) de la estación de trabajo. Los usuarios deben introducir una palabra clave que se verifica frente a las válidas para cada dirección NIC en particular. Una ventaja de este enfoque es su disponibilidad para todas las posibles computadoras que vayan a conectarse a la red, con lo que las que acceden a través de red conmutada poseeen el mismo nivel de seguridad que las conectados directamente al cable de la red. Pero incluso con este tipo de protecciones, los administradores deben llevar cuidado con los virus. En muchos casos, no es que haya que añadir ningún otro equipo especial, sino simplemente ciertos conocimientos en lo que a labores de administración se refiere.

Uno de los métodos más satisfactorios consiste en no esperar a observar que a alguien le falte espacio en disco. Puesto que los virus se propagan, suelen consumir mucha cantidad de espacio en disco, por lo que controlando las estructuras de forma regular es posible su pronta detección.

Entonces, ¿qué es lo que debe controlarse? Se deben buscar nuevos directorios que aparezcan repentinamente. También puede ocurrir que un disco pierda 20 MB de capacidad de almacenamiento en una hora o 200 en un día . Si sucede algo así lo más seguro es que exista un virus o un problema serio de hardware.

Asimismo, puede servir de gran ayuda el disponer de paquetes de software de exploración de virus. Muchos productos verifican los cambios ocurridos en el tamaño de los archivos o en

las estructuras de programas, como puede ser un código de redundancia cíclica diferente. Existen en el mercado muchos productos de este tipo. todos ellos ofrecen un método relativamente poco costoso para controlar la posible aparición de virus. Su mayor problema, sin embargo, reside en que no suelen considerar los virus más nuevos.

No obstante, la adquisición de paquetes de exploración y detección de virus y otros programas y equipos que impiden la introducción de éstos en las redes, son tan sólo una parte de la estrategia global defensiva a seguir.

Los especialistas del sector afirman que los administradores de redes deben desarrollar un plan concreto para tratar esta cuestión. Su puesta en marcha requerirá una amplia variedad de equipos, así como una amplia difusión de los peligros para que los usuarios puedan apreciar la magnitud del problema.

De cualquier modo, todos los datos disponibles indican que los virus van a estar ahí por mucho tiempo, indudablemente, los usuarios que adopten una adecuada estrategia preventiva se posicionarán por delante de la competencia.

"Rogue Programs: Viruses, Worms, and Trojan Horses"
Lance J. Hoffman
Van Nostrand Reinhold

"The Computer Virus Handbook"
Richard B. Levin
Osborne Mc Graw-Hill

F. Cohen
"A Short Course on Computer Viruses"
1990

"On Computable Numbers, With an Application to the Entscheidungs problem"
A. M. Turing

ComputerWorld/México
Julio de 1993

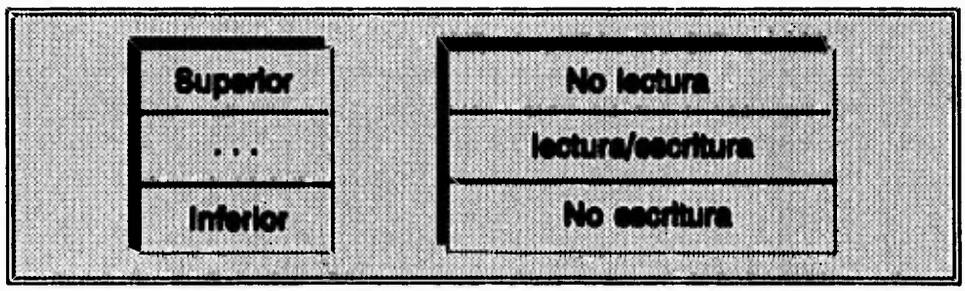
Defensas y protección en sistemas de cómputo 7

Técnicas de defensa puras

Existen técnicas de defensa que pueden ayudar a prevenir un ataque de virus en una computadora o en una red. Algunas de estas son la distribución limitada (limit sharing), transitividad limitada (limit transitivity) y funcionalidad limitada (limit funtion).

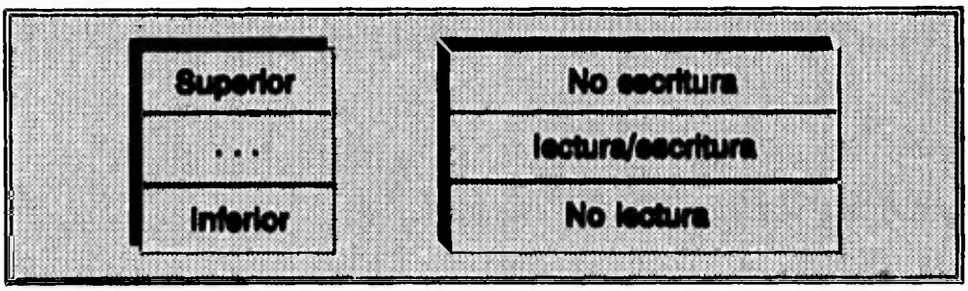
Distribución limitada (limit sharing).

A continuación se presenta un ejemplo de distribución limitada (limit sharing). En un sistema basado en modelo de seguridad de Bell-LaPadula, a los usuarios se les asigna un nivel secreto, donde pueden leer y modificar la información, en el caso de la información que se encuentra un nivel más superior únicamente se les permite el acceso de sólo escritura y en el caso de que la información se encuentre un nivel inferior sólo se les permite el acceso de lectura.



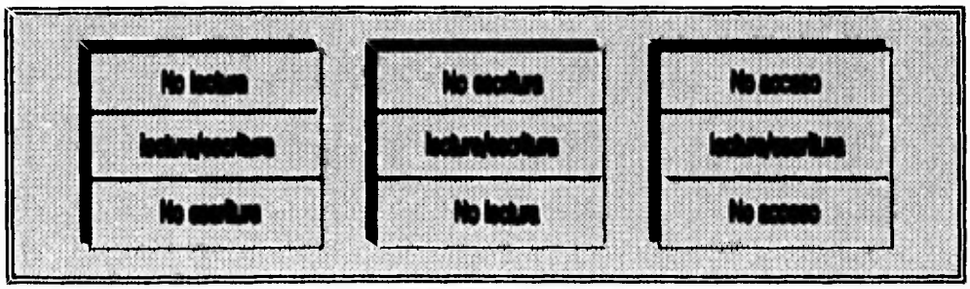
El modelo básico de confidencialidad Bell-LaPadula

Similarmente si tenemos un sistema de seguridad basado en el modelo de integridad Biba. Estando en el nivel de integridad asignado, no se podrá leer información de un nivel más bajo y tampoco se podrá modificar o escribir información en un nivel de integridad más alto.



El modelo de integridad Biba

Si se desea realizar una combinación de ambos, se tiene que eliminar la distribución, esto se logra haciendo coincidir los campos de un sistema con el otro para que no se pueda leer y escribir en niveles altos o niveles bajos.



Combinación de confidencialidad con integridad.

Lo anterior es sólo un caso específico de distribución limitada (limit sharing). A continuación se mencionará un caso más general de ésta técnica. Una mejor implementación es la política basada en la estructura llamada POset o Conjunto Ordenado Parcialmente (Partially Ordered Set).

Matemáticamente, una estructura POset está basado en las siguientes reglas:

1. $A <= B$ y $B <= C$ --> $A <= C$

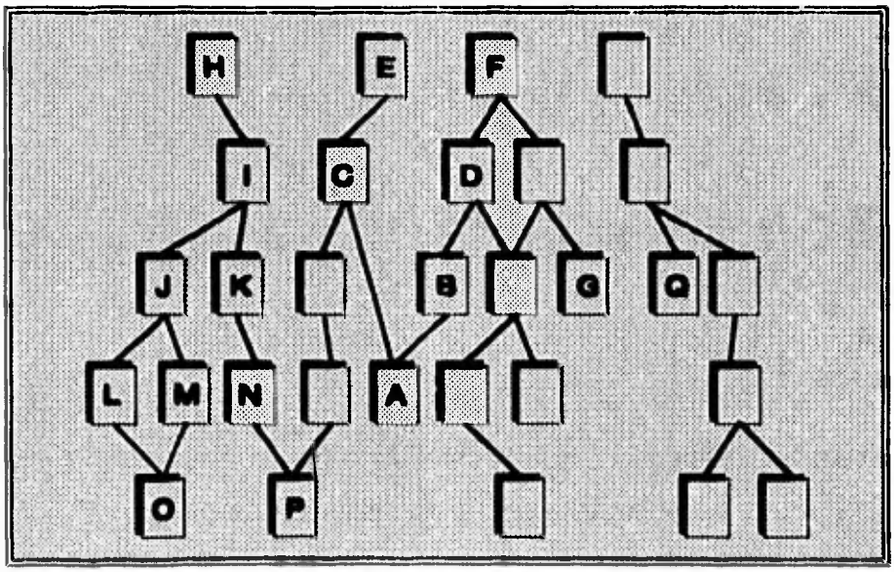
2. $A <= B$ y $B <= A$ --> $A <= B$

3. $A <= A$

Básicamente, un POset puede ser representado como un conjunto de cajas unidas entre ellas por líneas, y una de las reglas dice que la información sólo puede fluir hacia arriba (a la derecha o izquierda si es necesario cambiar de dirección) a continuación se describe el funcionamiento.

Suponiendo que alguien, estando en A, desarrolla un virus, este puede propagarse a B, C, D, E y F, pero no puede propagarse a G, porque no hay ruta para que la información de A pase a G. Si no hay información en A que puede llegar a G, entonces no hay un virus que pueda llegar a G.

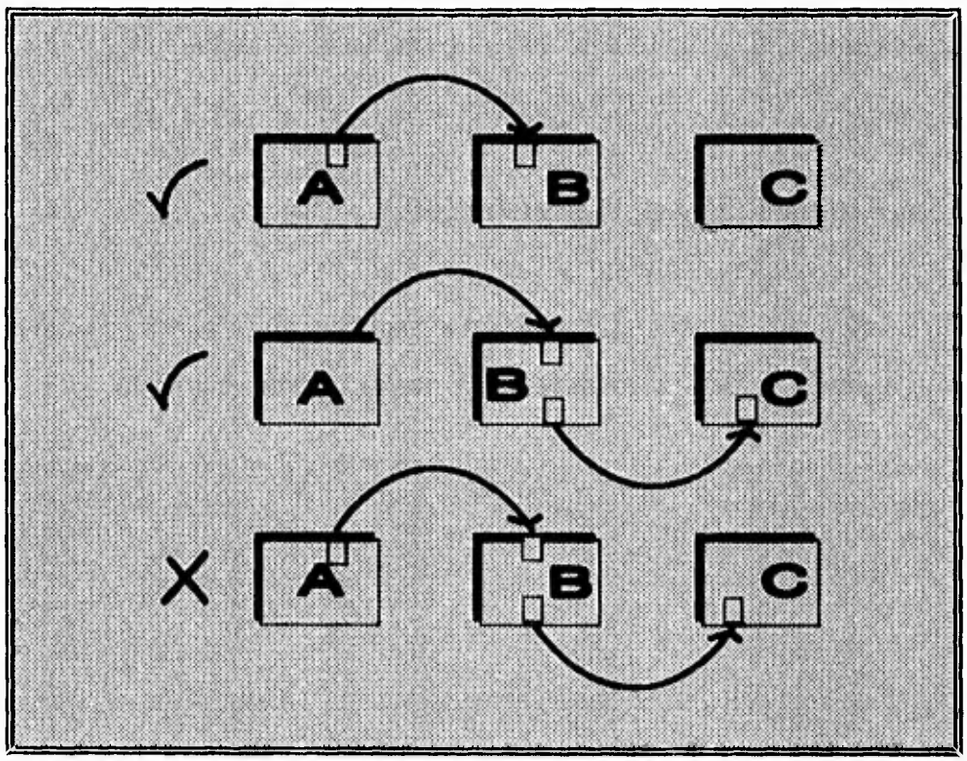
Similarmente, si alguien estando en H trata de infiltrar información, sólo lo hará para I, J, K, L, M, N, O y P, pero no logrará infiltrar información hacia Q, porque no hay ruta para la información en Q que puede llegar a H. En otras palabras, se puede limitar el flujo de información, y la estructura mas general para efectuar esto es el POset.



Conjunto ordenado parcialmente, POset

Transitividad limitada (Limit transitivity).

La transitividad consiste en transferir información de un dominio a otro. La segunda de las tres posibilidades de prevención es limitar la transitividad.



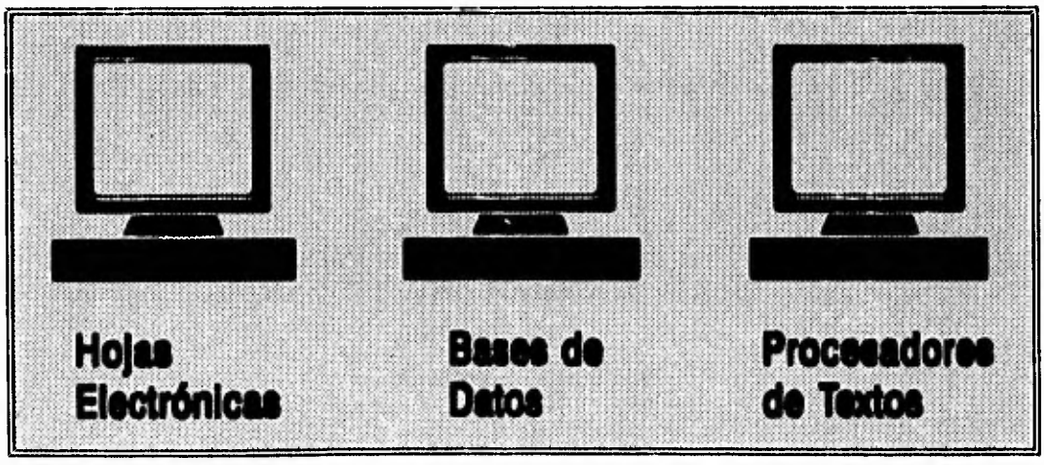
Transitividad limitada

A es capaz de dar información a B, y B puede dar información a C. Pero la regla dice que la información que sea pasada de A a B sólo B la podrá pasar a C.

Si se implementa el concepto de transitividad limitada (Limit transitivity), se puede prevenir que un virus se extienda rápidamente a partir de su origen, el único problema es que no es factible usar éste método eficientemente en sistemas reales de cómputo. Puede implementarse la transitividad limitada, pero cuando se trata de usar, se tiene el problema de que los virus son parecidos a cualquier otra clase de información, entonces de cualquier manera que se restrinja el acceso a los virus se aplicará la restricción a otro tipo de información.

Funcionalidad limitada (Limit funtion).

La tercera de la tres posibilidades para la prevención de los virus en una computadora o en redes es la Función Limitada. A continuación se indica su funcionamiento.



Funcionalidad limitada

Se puede tener un sistema para introducir información a una base de datos, seleccionar alguna de esta información para ser analizada en una hoja electrónica, elaborar una gráfica con los resultados obtenidos y después exportar la gráfica a un procesador de textos e imprimir el reporte final en un formato de dos columnas en una impresora láser. Se pueden efectuar todas la operaciones anteriores sin dar a los usuarios funciones de propósito general.

Desafortunadamente, hoy en día, la mayoría de los actuales programas manejadores de bases de datos tienen macro-instrucciones orientadas a funciones de propósito general. Se puede desarrollar una macro-instrucción que pueda enviar o recibir información de una base de datos a otra. De hecho, se puede desarrollar una macro-instrucción que pueda infectar una base de datos o una hoja electrónica. Similarmente, se puede adquirir una hoja electrónica sin que cuente con macro-instrucciones para la exportación e importación de información entre hojas electrónicas o entre bases de datos o incluso a algunos procesadores. De igual forma los procesadores de texto cuentan con macro-instrucciones de propósito general para la importación o exportación de datos entre diferentes procesadores, hojas electrónicas o bases de datos.

Existen algunas situaciones donde se puede implementar la Función Limitada (Limit Funtion). Por ejemplo, muchas redes EFT o Transferencia Electrónica de Fondos (Electronic Funds Transfer), son limitadas en su función. La forma en que las Redes EFT normalmente operan es que cualquier información sobre la red reune los siguientes características:

1. Desde una cuenta para otra cuenta

2. Con una cantidad de dinero

3. Una cadena de caracteres para verificación.

Con esto, se logra que no cualquier clase de información sea enviada por la red, se puede tratar de enviar un programa sencillo, y la red tratará a los primeros bytes como el origen y destino, característica número 1, los siguientes bytes serán el monto del dinero, característica número 2, y los últimos corresponderán a la cadena caracteres de verificación, característica número 3. Si la verificación no tiene éxito, la red ignora la transacción y reporta a ésta como un error.

La razón por la que una de estas redes no puede ser infectada es porque la información que se envía no puede contener un virus. La limitación de funciones no se da por el tipo de información, se da por la forma de cómo la información puede ser interpretada, en otras palabras, la información sólo tiene una forma de ser interpretada.

Clasificación de defensas

Existen diferentes formas de defensa que se pueden implementar para la prevención de un virus y limitar su habilidad para diseminarse en un sistema de cómputo o una red. Si las defensas son inadecuadas para un caso en particular, se tendrá que pensar en algunas otras, en particular, en la detección y cura.

Todas las defensas están catalogadas como imperfectas, ya que llevan consigo algunas interrogantes respecto a que si se pueden o no implementar con un alto grado de confiabilidad.

Algunas de estas interrogantes son las siguientes:

- ¿Se puede desarrollar un programa que pueda verificar a otros programas e indicarnos si tienen o no virus?

- Si se tiene un secuencia de instrucciones que identifiquen a un virus en particular. ¿Se puede desarrollar un programa que encuentre todas las infecciones que este virus haya causado?

- ¿Existe un método estadístico para la detección de virus? Quizá se pueda poner alguna clase de mecanismos estadísticos en el sistema que identifique virus cuando estos llevan a cabo su infección.

- ¿Si se encuentra un virus, se puede remover este del sistema sin que la información quede alterada o incompleta?

- ¿Se puede implementar un mecanismo realmente eficiente para mantener la integridad del software? Es decir, programas que se auto-defienden de ataques.

Se podrá hablar acerca de N-versiones de programas, usuarios experimentados, vacunas, etc. Pero muchas de esas defensas nos dejarán en una posición de supervivencia contra los

ataques, ya que nos podemos encontar en un punto donde los ataques son más efectivos que las defensas.

La Asociación para la Industria de Virus de Computadora (CVIA, por sus siglas en inglés), clasifica las técnicas de defensa en:

Defensas por hardware

Control de Pre-boot

Es un circuito electónico (tarjeta o hardware lock) que se agrega al hardware de la computadora para examinar y verificar la integridad de los elementos del sistema operativo antes de proceder a levantarlo. Estos productos sólo existe para computadoras personales, en donde verifican la integridad del "master boot record", los archivos de configuración y el intérprete de comandos.

Dispositivos del tipo WORM (write-once-read-many)

Discos ópticos que pueden ser escritos una sola vez y que garantizan la integridad de los archivos y programas que ahí están almacenados. Este método, aunque aún es caro, puede ser una buena medida para evitar un contagio por virus. Sin embargo, existe el riesgo que lo que se almecene ahí esté contaminado.

Defensas de software

Productos de prevención de infecciones.

Normalmente son programas residentes que monitorean las llamadas al sistema que tienen que ver con escritura o modificación de datos. No son 100% efectivos y en ocasiones confunden programas ordinarios con virus.

Productos de detección de infecciones.

Son programas que se checan si ha habido cambios en programas ejecutables y archivos de datos, normalmente mediante técnicas de cheksum o encriptamiento, esta última ha dado los mejores resultados hasta ahora.

Rastreadores de virus (Virus Scanners)

Un rastreador de virus es un programa capaz de examinar los sistemas para encontrar las ocurrencias o señales concretas de los virus para las que ellos han sido programados que detecten. Los problemas que se pueden presentar con este tipo de utilerías se mencionan a continuación:

- Sólo pueden detectar un número limitado de virus y patrones de ataque conocidos. Esto significa que virus nuevos o modificados pueden estar activos, propagándose y son completamente indetectables por los rastreadores.

- En una organización donde la información es de vital importancia y el tiempo muy valioso el ejecutar el scanner implicará un pérdida de dinero.

- Generalmente éste tipo de programas requieren frecuentes y a veces costosas actualizaciones cuando se descubren nuevos virus o cuando antiguos virus se actualizan. Los nuevos patrones o señales de los virus tienen que ser agregados al código fuente del rastreador. Los usuarios que no utilicen versiones recientes estarán en peligro de un ataque o infección.

- Los rastreadores de virus no son muy eficientes contra virus con la capacidad de evolucionar que empiezan a desarrollarse en la actualidad.

- Son mas lentos cada vez que se actualizan. Los virus actuales también atacan a hojas electrónicas o bases de datos por lo que los lugares a buscar y patrones se incrementan, incrementando así el tiempo.

- Los rastreadores de virus (virus scanners) son superados fácilmente por los virus modernos que emplean las técnicas de codificación de datos para enmascarar sus patrones. Tales virus codifican sus signos reveladores bien encriptándolos o cambiándolo como sus patrones cambian con cada nueva infección, no queda nada tangible para que los scanners de virus busquen o identifiquen.

- Producen falsos positivos para patrones cortos. Ya que puede presentarse el patrón en un archivo por casualidad sin que este sea virus.

Productos de identificación de infecciones.

Son programas que identifican virus, regenerando programas y archivos infectados. Este método no proporciona una protección contra toda clase de virus (sobre todo contra los llamados mutantes) y frecuentemente se requiere la actualización de éstos productos.

Vacunas

La llamadas vacunas fueron de los primeros productos anti-virus que aparecieron cuando se presentó el problema de los virus informáticos. Estos consisten en eliminar el virus del sistema y en algunos se contempla la capacidad de inmunizar contra el atacante. Sin embargo, a continuación se mencionarán algunos problemas con esta clase de producto para la erradicación de virus.

- Las vacunas pueden falsear la información, es decir, indicar la presencia de un virus cuando en realidad no existe (falsos positivos).

- La base de conocimientos actual (sobre virus) en el programa vacuna, sólo sirve para virus conocidos.

- No todos los programas pueden ser vacunados, ya que existen virus que dañan el archivo donde se alojan y al ser eliminados dejarán el archivo en malas condiciones.

- Al vacunar contra un virus esto puede permitir que otro virus entre, como se explica en el capítulo anterior, en el tema de recuperación de un ataque de virus.

En el caso de las vacunas que inmunizan al sistema, la técnica empleada para este fin es agregar pequeños programas y datos de suma total a los archivos ejecutables. Los archivos ejecutables son modificados para que cuando se ejecuten, se pase primero el control de los programas anti-virus agregado. Los programas anti-virus compararan las sumas totales activadas con sus datos de suma total agregada. Cuando las sumas corresponden el control se devuelve al archivo ejecutable y las operaciones continúan en forma normal, en caso contrario se alerta al usuario para que tome las medidas pertinentes. Los problemas que se presentan al emplear la técnica anterior pueden ser los siguientes:

- Los programas vacunados tardan más tiempo en cargarse ya que el código y datos del anti-virus aumentan su tamaño, además de que el proceso previo de verificacion de suma total a la ejecución requiere más tiempo.

- No hay garantía de que las modificaciones de los archivos ejecutables no afecten negativamente a sus operaciones.

- Puede presentarse el caso de que la vacuna empleada para inmunizar esté contaminada con un virus que ésta no contemple en su base de conocimiento.

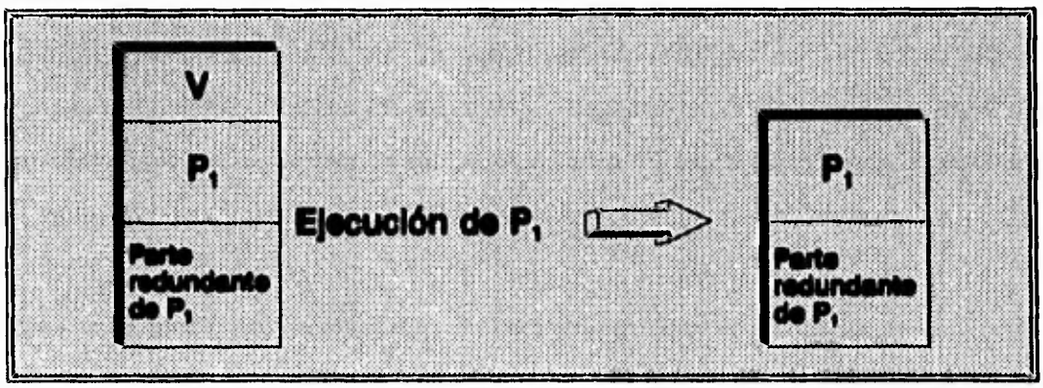
- La conducta de las vacunas, parecidas a la de los virus, puede causar conflictos con otros sistemas de defensa viral.

- Los virus pueden detectar la presencia del código del programa vacuna en los archivos ejecutables destino y borrar o modificar los datos relacionados con la vacuna, o pueden corregir a los ejecutables inmunizados para saltarse el proceso de verificación.

Software con auto-defensa

La idea del software con auto-defensa es de algunos años atrás. Básicamente consiste en que el programa se auto-verifica y automáticamente elimina la corrupción si es que la había.

La clave está en construir el programa con alguna clase de redundancia en él. Cuando se ejecute el programa, éste se verificará a si mismo, usando la misma clase de redundancia, y si no se encontró normal, detectará el error y corregirá éste, usando de igual forma la misma clase de redundancia para la corrección.



Software con auto-defensa

Algunos de problemas que se pueden presentar con este tipo de software se mencionan a continuación:

El primer problema que se puede presentar es que el virus puede modificar la parte redundante del programa P_1 para alojarse o propagarse. A continuación un ejemplo de lo antes mencionado:

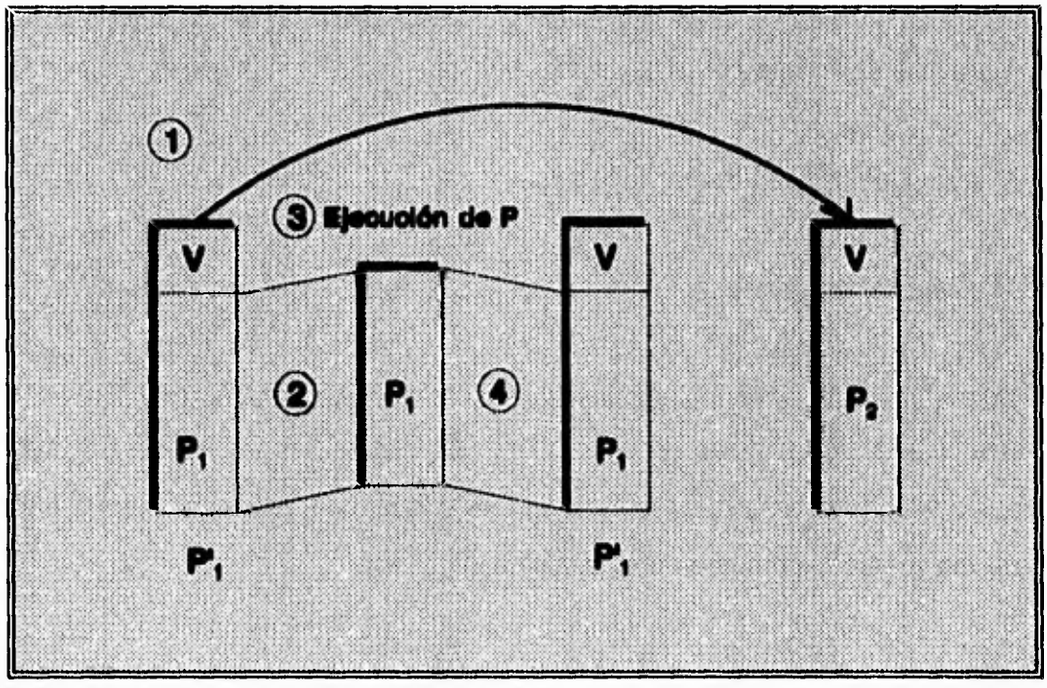
Tenemos un programa "ESPEJO" que se ejecuta en algún sistema. ESPEJO estará diseñado para elaborar una copia exacta del programa que se desee, con la característica fundamental de que todos los unos serán cambiados por ceros y los ceros por unos, además de almacenar la imagen espejo (mirror image) del programa en cuestión en el almacenamiento secundario del sistema. Cuando se ejecute el programa, previamente se le elaboró una imagen espejo con ESPEJO, el sistema operativo comparará el original con la imagen espejo. Si no son exactamente opuestos, el sistema elaborará un programa limpio a partir de la imagen espejo.

En el caso de que se desarrollara un virus para infectar a la imagen espejo (mirror image) con una imagen espejo del virus, al ejecutar el programa, el sistema comparará el original con la imagen espejo. Debido a que habrá diferencias entre el programa y su imagen espejo, el sistema elaborará a partir de una ella un nueva versión limpia del programa infectado.

El segundo problema que se puede presentar es que la detección puede fallar si se utiliza un simple método de checksum, ya se mostró con anterioridad como también puede fallar con la compresión de virus.

Existe una gran cantidad de técnicas diseñadas para aumentar la calidad del software que se autoverifica, una de ella es la suma de comprobación criptográfica (cryptographic checksum). Esta técnica puede variar en su implementación, es decir, puede comprobar sólo un sector al azar del archivo o el primero o el último. Otros seleccionan cinco o seis bloques cualesquiera. Estos esquemas pueden ser muy rápidos ya que no verifican toda la información pero tienden a fallar.

El tercer problema es el tiempo invertido que el programa toma para mantenerse limpio. Esta es una situación de riesgo que hace que la auto-defensa falle. A continuación se explica:



Esquema general de ataque contra la técnica de auto-defensa.

Presentamos un ataque genérico contra cualquier mecanismo de autodefensa, el ataque se llevará a cabo sin conocimiemto de la técnica de auto-defensa empleada por el defensor, sólo se conoce que se está empleando cualquiera de ellas, sólo así se garantiza que este ataque podrá trabajar.

Inicialmente tenemos a P'_1, que es la versión infectada del programa P_1, que fue infectada con el virus V y una versión limpia del programa P_2.

Si al tiempo t_1, ejecutamos a P'_1 la siguiente secuencia de eventos ocurrirán:

1. V infecta al programa P_2.

2. V reemplaza P'_1 con el original P_1.

3. V ejecuta el original P_1.

4. Después de que finaliza la ejecución de P_1, V reemplaza P_1 con P'_1.

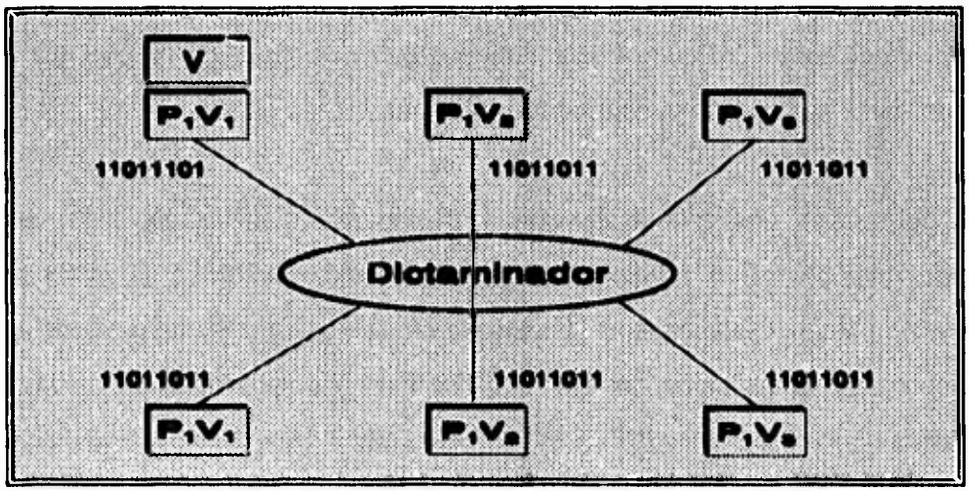
Durante el paso 3, P_1 se verifica a si mismo y determina que se encuentra limpio. La misma situación se puede presentar en el mundo real.

Técnicas de defensa sólidas

Denominaremos técnicas de defensa sólidas, aquellas que nos proporciona un grado muy alto de protección, pero esto no quiere decir que no existan virus o situaciones de riesgo que las pueden evadir. Algunas de las técnicas de defensa son las siguientes:

Software tolerante a fallas

El software tolerante a fallas consiste en la ejecución de copias redundantes de programas. A continuación se presentará un ejemplo de este tipo de defensa: En la figura siguiente tenemos representado un sistema con tres programas:



Programación de N-versiones

Estos son P_1V_1, P_1V_2 y P_1V_3. P_1V_1 está infectado con un virus, por lo que produce una salida con un resultado erróneo. Las salidas de los programas pasan a través de un programa dictaminador, que elimina el resultado erróneo y hace que las tres salidas tengan resultados idénticos para que el sistema continúe funcionando sin ninguna alteración. Así los programas P_2V_1, P_2V_2, P_2V_3 tendrán entradas correctas, aun cuando P_1V_1 haya producido una salida errónea. El programa dictaminador es también redundante para autoprotegerse de sus errores.

A continuación se presentan algunos problemas con las N-versiones de programas que deben ser creados en este tipo de defensa.

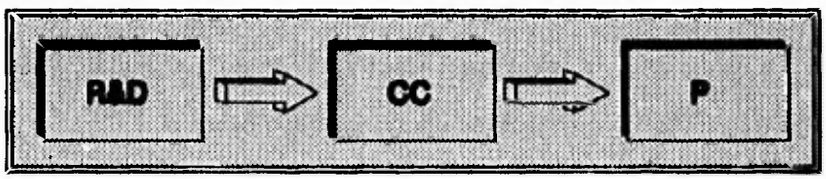
- El costo paran la implementación es muy elevado. En el caso de este ejemplo, debe de haber tres diferentes programas, lo que implica tres diferentes programadores, también se necesita tener tres veces la capacidad de cómputo para tener la misma eficiencia.

- Difícil de elaborar. Ya que se tienen que implementar varios programas que realicen lo mismo, pero que sean totalmente diferentes entre sí.

- Esta defensa no está garantizada contra ataques intencionales. Es decir, esta defensa trabaja muy bien si el problema que se presenta es totalmente aleatorio pero no es aplicable a ataques intencionales. Un ataque intencional, por ejemplo, puede consistir en la elaboración de un mecanismo de modificación legítimo para modificar los resultados de salida de los programas P_1V_1 y P_1V_2, y entonces el programa dictaminador cambiaría el resultado correcto de la tercera versión por lo que el uso de esta defensa no garantiza su funcionamiento contra un ataque intencional.

Control total a cambios

El control total a cambios es un proceso de seguridad de calidad que verifica cualquier tipo de corrupción a través de un "guardia". Normalmente se tienen dos ambientes, uno el "de investigación y desarrollo" (Reserarch y Develoment, R&D), y otro el ambiente de "producción" (P). El ambiente R&D es donde se hacen y se examinan los cambios. El ambiente P es donde se actualiza el uso de esos cambios día con día. Para que se tenga un control total a cambios existe un área entre los ambientes R&D y P llamada "control de cambio", que contiene un conjunto de reglas acerca de como trabaja el sistema.



Control total a cambios

Estas reglas son típicamente soportadas por controles administrativos y especialistas en seguridad. Las reglas para el control total a cambio usualmente vienen siendo muy similares a las que se presentan a continuación:

1. El control total a cambios sólo puede aprobar o rechazar un cambio propuesto; este no puede hacer un cambio como si fuera propio. En otras palabras los cambios se hacen en el R&D y después se pasan ellos al control de cambios. Todos los controles de cambios sólo pueden indicar dos cosas: es aceptable o no lo es.

2. El control total a cambios sólo puede pasar fuentes desde el R&D a P. No puede pasar librerías en código ordinario, ejecutables o cosas similares, porque no se puede realmente determinar qué tipo son los no fuentes.

3. El control de cambios es hecho vía humanos y por métodos de verificación automática.

- Se verifica que el cambio sea necesario. Esto es, que el cambio no se llevará a cabo sin motivo alguno.

- Se verifica que el cambio sea el apropiado para un fin específico. En otras palabras, se expresa la necesidad de un cambio por un motivo en particular, y se debe tener la seguridad de que el cambio está relacionado con ese motivo.

4. El cambio tiene que ser probado sobre un dato muestra desde el ambiente P para verificar que trabaja apropiadamente.

5. La operación del cambio debe ser evidente y obvia. Software que no lo es usualmente puede ser fácilmente modificado para incluir rutinas de ataque ingeniosas.

6. El cambio no debe tener cualquier código innecesario o datos complicados. Cualquier código innecesario probablemente puede contener virus u otro tipo de amenazas.

Shell de integridad

Actualmente existen muy pocas soluciones prácticas para el problema de virus en computadoras, una de ellas que funciona tanto en procesos grandes como en procesos pequeños es la denominada Shell de integridad (Integrity shell).

Un Shell de integridad utiliza la redundancia para detectar los cambios, y así adquirir en cierta forma un grado de automatización de las técnicas de tolerancia a fallas y control a cambios. Muchas veces conocemos el momento justo de cuando se debe comprobar, y qué comprobar y para ciertos casos, realizar automáticamente las correcciones. A continuación se explica este funcionamiento.

Comenzaremos con replantear los problemas del software con auto-defensa, ya que los shells de integridad dan una alternativa a la solución de los problemas presentados por las técnicas de autodefensa. A continuación se mencionan los tres problemas:

1. Un virus puede modificar la parte redundante del programa.

2. La detección puede fallar.

3. En el momento que el programa toma el control, éste puede aparentemente encontrarse limpio.

Suma de comprobación criptográfica (Cryptographic checksum)

Los tipos de problemas abarcados por los puntos uno y dos se relacionan con el hecho de que necesitamos una vía confiable para detectar la corrupción en presencia de un ataque intencional. El único camino que conocemos para hacer esto es mediante la ejecución de suma de comprobación criptográfica, cryptographic checksum, (por brevedad, a partir de ahora utilizaremos la notación $\sqrt{\Sigma}$ para representarla). Pero, ¿qué es la suma de comprobación criptográfica?

Se empezará explicando el concepto de la suma de comprobación (checksum). Esta es como una huella digital. Si se toma una huella digital de un archivo y a continuación se modifica el archivo, su huella digital también cambiará. Por lo que la siguiente ocasión en que se observe el archivo, este tendrá una huella digital diferente y por lo se notará el cambio. Pero, todavía se tendrá que expresar la huella digital de una manera confiable y no redundante.

En el caso de que se realize una $\sqrt{\Sigma}$ por encriptamiento de un archivo utilizando una clave secreta, y luego se efectúa una suma de comprobación del archivo guardando ésta en el mismo archivo, con un sistema criptográfico suficientemente bueno, y la clave se mantiene en secreto, se tendrá un huella digital difícil de crear. Para ser más precisos, se contará con un conjunto de tres elementos $[(F_1,k,S_1,)...(F_N,k,S_N)]$, en donde cada F es un archivo, k es la clave secreta especificada por el usuario, y cada S es una $\sqrt{\Sigma}$. Sólo el usuario conoce la clave, en tanto que el atacante puede ver el archivo y la $\sqrt{\Sigma}$.

Si el sistema criptográfico es suficientemente bueno, no podrá modificar F con el fin de que bajo la misma k tenga la misma √Σ, no podrá modificar F y √Σ para que bajo la misma k ambas coincidan, y no podrá acceder a k por medio de la observación de F y √Σ. Esto es lo que hace una técnica √Σ para este propósito en particular.

A continuación se listan algunas de las otras propiedades de las técnicas de √Σ, para que podamos comprender mejor sus virtudes.

- La √Σ puede ser independiente del sistema operativo. Esto es, podemos tener una √Σ del propio sistema operativo por si es modificado por un atacante, o también por si un administrador del sistemas, un operador del sistemas, y/o un programador de sistemas, se unieran para hacer trabajar en forma diferente al sistema sin hacerlo saber a nadie, una √Σ puede aún detectar la modificación.

- Una √Σ es también independiente de otros usuarios, así que cada usuario puede independientemente verificar el control del cambio. Cada usuario puede contar con una clave diferente, de tal forma que si un atacante lleva a cabo una falsificación esta sólo trabajara adecuadamente bajo esa misma clave, de tal forma que esto resulte poco probable su funcionamiento bajo el resto de la totalidad de las claves en el sistema.

- Son posibles revisiones múltiples, independientes, con una √Σ. Esto significa que no sólo podemos realizar revisiones o comprobaciones internas, sino que un auditor externo puede

acceder y realizar la auditoría del control a cambios y detectar qué es lo que ha cambiado, si se presenta el caso de que no conocemos el hecho de algo ha cambiado. Ellos pueden acceder y utilizar su propia √Σ con absoluta independencia.

- Una √Σ puede trabajar sobre la totalidad de la información, no precisamente sólo sobre archivos binarios ejecutables. Puede trabajar también sobre archivos sobrepuestos, archivos de bases de datos, archivos de datos, o cualquier otro tipo de información en un sistema de computadora o en una red.

- Una √Σ puede trabajar sobre redes. Puede realizar una √Σ en una PC y verificar su efectividad sobre un marco principal. Puede verificar la integridad de la información durante la transmisión, el almacenamiento, y la recuperación en la red, y puede utilizar la misma técnica de √Σ, independientemente del tipo de sistema en el cual se encuentre trabajando.

- Existe una dependencia entre funcionamiento y protección. Podemos usualmente lograr una mejor protección si sacrificamos un poco el funcionamiento. Ello no significa que un funcionamiento más bajo siempre proporcione una mayor protección. Por ejemplo, puede tener un sistema criptográfico lento que particularmente no sea bueno. No obstante, podrá mejorar el nivel de protección mediante el uso de una clave de mayor tamaño, almacenando una √Σ más grande, etc.

Funcionamiento del shell de integridad

El problema final del software con auto-defensa, en el momento en que un programa asume el control, es que éste puede aparentemente estar limpio. Recordemos que existe un ataque genérico contra cualquier mecanismo de auto-defensa.

¿Cómo se puede solucionar este problema? Una solución, es revisar el programa antes que éste asuma control. Se puede lograr esto mediante el uso de un producto que se denomina Shell de integridad. Aquí se explica la forma en que trabaja un shell de integridad:

El usuario dice ''Interpreta X'':

If X no se ha modificado AND cualquier X dependendiente no se ha modificado then

interpreta X;

else

Acepta; OR

Confía; OR

Olvida; OR

Recupera;

Se empezará por describir lo que se debe de entender por la interpretación de X. No significa de ninguna manera que X sea un programa binario ejecutable el cual corre en el hardware, en otras palabras, este sólo es un ejemplo de interpretación. Puede ser que sólo se

cuente con una hoja electrónica que sea interpreta por un programa de hojas electrónicas. Se interpreta la hoja electrónica, pero ésta depende para su propia interpretación del programa de hojas electrónicas. El programa de hojas electrónicas en turno, puede depender de una diversidad de otro tipo de información. Lo mismo se aplica a un programa fuente que sea traducido por un compilador o descifrado por un interprete, una base de datos que sea interpretada por un programa manejador de bases de datos, etc.

Cuando el usuario indica "interpreta X", si X no ha variado (es decir, si la $\sqrt{\Sigma}$ para X no ha cambiado) y si cada una de las X dependientes para su propia interpretación en el programa no han variado (esto es, sus $\sqrt{\Sigma}$s no han cambiado), entonces sabemos que tenemos un ambiente no cambiado. En otras palabras el ambiente no ha sido contaminado desde la última vez que fue utilizada una $\sqrt{\Sigma}$. Por lo que simplemente se puede interpretar X en un grado razonable de seguridad de que éste es lo que se supone que es. Si tanto la información que está siendo interpretada como la información de la que se depende para su interpretación de haber sido modificadas, no se puede confiar que la interpretación funcionará adecuadamente. En este caso, se tiene que seleccionar algunas de las opciones que a continuación se presentan:

- Una opción es aceptarlo en su nueva forma. Esto quiere decir, utilícelo, aún cuando se tenga el conocimiento de que se halla contaminado. En algunas casos el daño que se espera por utilizar información contaminada es menor que el que se puede presentar al detener la ejecución del mismo.

- Otra opción es "confiar en el cambio". Por ejemplo, Si se realiza un cambio legítimo, se debe de informar al sistema que el cambio es apropiado, y que debe de identificarlo dentro de los cambios adicionales. Este es a menudo el caso cuando se trata del desarrollo de programas y su mantenimiento.

- Otra opción es "olvidarlo".

- La última opción es "recuperarlo". Si se cuenta con un respaldo y se conoce donde se encuentra, se pude ir y tomar ese respaldo y automáticamente restaurar la información contaminada en su estado previo. En este caso, el sistema determina la contaminación y se procede normalmente con un ligero retraso de tiempo.

Ejemplo del funcionamiento de un shell de integridad

Se intentará dar un ejemplo del funcionamiento normal de un shell de integridad operando con cierto número de tipos diferentes de contaminación. Comenzaremos por desactivar la protección de integridad para que podamos presentar un virus tal como si este operase sin la presencia de alguna integridad. Este virus se dispersa de programa a programa, mostrando su avance a lo largo del camino. Entonces activamos la protección de integridad de nuevo y repetimos la demostración, sólo que en esta ocasión la contaminación se ha detectado inmediatamente, y se ha evitado una dispersión adicional.

Ahora la mayoría de los ejemplos son contra virus, pero esto revela que los shell de integridad no sólo trabajan contra los virus en computadoras, ya que estos tienen implantados mecanismos de usos muy generales. A continuación un ejemplo. Estando dentro del directorio de DOS, en donde todos los programas de DOS se hallan almacenados, encontramos un programa denominado "FORMAT.COM", el cual, es un programa del sistema operativo. Ejecutaremos FORMAT para observar los mensajes que se presentan en la pantalla, y después ejecutaremos un "Ctrl C" para terminar sin que el comando tenga algún efecto:

```
C:\>FORMAT A:
Insert new diskette for drive A:
and strike ENTER when ready  ^C
```

El siguiente comando que ejecutaremos será:

```
C:\>ECHO Que tal Hugo > FORMAT.COM
```

Esto reemplazará el contenido de FORMAT.COM. Entonces volveremos a ejecutar FORMAT.COM para observar que éste realmente ha sido contaminado, y ejecutamos el comando "DIR" para que muestre que FORMAT.COM es sólo de 17 bytes de longitud.

```
C:\>TYPE FORMAT.COM
Que tal Hugo
```

```
C:\>DIR FORMAT.COM
FORMAT    COM        17     09-21-93     12:01p
C:\>
```

En el ambiente de DOS, si se ejecuta un programa de este tipo (no se recomienda que lo haga sin un shell de integridad instalado), el sistema normalmente se dañaría. Esto es porque el sistema operativo simplemente carga el archivo en la memoria y empieza ejecutando instrucciones en el primer byte del archivo. Ahora ejecutaremos FORMAT.

```
C:\>FORMAT A:
Insert new diskette for drive a:
and strike ENTER when ready  ^C
```

El Shell de integridad detectó la contaminación, reemplazó el FORMAT.COM utilizando un respaldo en línea, y corrió el programa correcto de FORMAT, tomando únicamente unos cuantos segundos extras.

Si desea restaurar desde un respaldo en línea, requiere tener el respaldo en línea. Puede hacer lo mismo con respaldos fuera de línea, pero ello tomará mucho tiempo más. Entonces, tiene una relación espacio/tiempo. Los respaldos en línea cubren más espacio, en tanto que los respaldos fuera de línea toman más tiempo.

Limitaciones del shell de integridad

A continuación se mencionan algunas limitaciones y características de un shell de integridad. Los cuales son muy buenos contra los virus, pero tienen también sus limitaciones.

La primera limitación es que el sistema criptográfico debe ser bastante seguro. ¿Pero qué significa "bastante seguro"? Bien, lo que realmente significa es que ningún atacante pueda aproximársele, pero es difícil determinar el momento justo en que el atacante se está aproximando dentro de la misma ejecución. Pero, existen otras cuestiones diversas que deben considerarse. Por ejemplo, el intercambio del grado de la operación con respecto al grado de integridad, como ejemplo tomaremos un shell que posee diferentes sistemas criptográficos dependiendo del grado de intercambio que se desea en la relación. Entonces, tiene que ser capaz de decidir si desea que éste tome 10 segundos, 1 segundo, 0.1 segundos ó 0.01 segundos en promedio para revisar un programa. Si selecciona 0.01 segundos, probablemente no logrará mucho en él con respecto al grado de integridad. Si toma 100 segundos, probablemente será demasiado lento para justificar su uso. Por lo que normalmente, se fija un grado de operación, para desarrollarlo efectivamente.

Necesita una base firme para el grado de confiabilidad de cualquier sistema criptográfico, y en términos de criptografía, existen sistemas de uso general los cuales son confiables en este momento.

Uno de ellos que es relativamente seguro, es el sistema criptográfico DES (Data Encription Standard). Este sistema se resiste a una buena cantidad de atacantes, gente especialista en la materia puede romper el código DES en cuestión de unos días (y en algunos casos en cuestión de horas), el atacante típico que desarrolla algún virus tiene poca probabilidad para romper códigos de DES, y si en forma personal lo intentaran, les tomaría un largo tiempo el crear una infección.

Otro sistema es el RSA. Se puede romper cualquier código en este sistema, pero la cantidad de tiempo que se requiere para romper el código puede variarse mediante el cambio del tamaño de la clave. Por ejemplo, un RSA de cientos de dígitos tomaría cientos de años para romperse utilizando los algoritmos más rápidos en las computadoras más rápidas. Una rúbrica o signatura RSA de doscientos dígitos tomaría cientos de trillones de lapsos de vida del universo para romper con los mejores sistemas disponibles hoy en día, pero también toma mucho tiempo crear las rúbricas.

¿Cómo determinamos si un sistema es seguro? Es muy difícil determinar si un sistema criptográfico es seguro, pero existen algunas indicaciones de sistemas no seguros. Por ejemplo, si se tiene un sistema criptográfico que no toma demasiado tiempo para revisar elementos que son de tamaño considerable, es posible que éste no se encuentre revisando todo, porque toma más tiempo encriptar más información. Existen un par de productos que revisan archivos de 50 Kbytes en una décima de segundo en una PC-XT. Esto ciertamente no es seguro, porque toma más de un segundo para leer 50 Kbytes en una PC-XT típica. Otro ejemplo de práctica criptográfica no

segura es el uso de una clave intercambiable del sistema criptográfico. Si todos tenemos la misma llave, el atacante también la tiene, y puede efectuar modificaciones sin ningún problema.

La segunda limitación es que el mecanismo en sí mismo debe ser inalterable. Si no puede asegurar el mecanismo, un atacante puede modificarlo para permitir contaminaciones que pasen inadvertidas. Se puede usar una $\sqrt{\Sigma}$ para revisar el shell de integridad, pero si podemos alterar el mecanismo, podemos rebasar cualquier defensa.

Virtualmente en todo sistema, existe una manera de utilizar protección del hardware para defender el mecanismo de integridad. Así mismo una PC, podrá tener discos flexibles con protección de hardware contra escritura los cuales proporcionan un grado confiable de protección contra la contaminación del mecanismo de integridad. Si se corre una revisión de la integridad desde el disco flexible para revisar el mecanismo de integridad del disco duro, y todas las otras partes críticas del sistema operativo, entonces tendrá un grado alto de seguridad de que estos se encuentran en buen estado. Se puede empezar utilizándolos como base para correr otros programas en el sistema.

A continuación se hará una observación acerca de las etiquetas de protección de escritura en discos flexibles para PC. Si protege un disco flexible con una etiqueta de plástico negra para protección contra escritura, ésta puede no funcionar. Esto revela el hecho de que muchas PC's utilizan rayos infrarrojos para detectar la presencia de una etiqueta de protección contra escritura, y esas pequeñas barritas negras no bloquean la luz infrarroja. Por lo que tendrá que al utilizar

la etiqueta de protección contra escritura deberá de ser de metal. Los discos flexibles de 3 1/2", no presentan estos problemas.

El tercer elemento que necesitamos es un vía confiable para la clave. Si el atacante puede observar la clave cuando la escribimos, puede utilizar la clave para forzar una √Σ. Normalmente, una vía confiable se genera mediante el forzamiento del shell de integridad para ejecutarse antes que otros programas en el ambiente, para que nada pueda insertarse por sí mismo entre el arranque del sistema y la operación del shell de protección.

La cuarta limitación es realmente más una negociación que cualquier otra cosa. Si estamos tratando de cubrir una base de datos de 500 megabytes, podremos verificar la totalidad de la base cada vez que leemos, y la resumimos cada vez que efectuamos un cambio, esto resulta poco práctico.

Para cosas como bases de datos grandes, probablemente resulta mucho más apropiado realizar la revisión a un nivel de granularidad más bajo que a nivel del archivo. En tal caso, se podrá tener el programa de base de datos verificando la √Σ asociada con cada registro cuando éste es leído, y actualizar la √Σ cada vez que se escribe un registro.

La limitación final de los shells de integridad es un tema fundamental, en el que la legitimidad del cambio es una función de intentar. Es decir, no podemos decir que un cambio sea o no normal, a menos que podamos de alguna manera determinar cual es la intención del usuario, y comparar dicha intención con el cambio actual.

En la mayoría de los sistemas actuales, la mayor parte de los usuarios no saben cual parte del sistema deberá cambiar cuando realizan un cambio de algo sobre su pantalla. Ellos no tienen noción de esto. Por lo que finalmente, no tenemos modo de asegurar la legitimidad del cambio en cualquier sistema de este tipo. Los shells de integridad pueden detectar cualquier cambio y actuar como si hubiesen sido avisados, pero no pueden determinar la intención del usuario y mapearla dentro de las acciones del sistema.

Características de los shell de integridad

Ahora que hemos analizado las limitaciones de los shells de integridad, también debemos analizar sus tres características principales. Son óptimos para defensas contra los virus en un ambiente de computadoras no confiable, son eficientes en términos del sobrecarga (overhead) requerido para su operación, y son altamente confiables tanto contra atacantes conocidos como contra atacantes desconocidos.

Protección óptima

La característica básica de un shell de integridad y la razón por la que el shell de integridad fue creado así, es que éste es óptimo. Esto es, en un sistema de computadora no confiable, por definición, no se puede evitar la infección. Dado esto, lo mejor que puede es esperar, detectarlo y limitar una propagación. Esto es lo que un shell de integridad realiza.

- Un shell de integridad detecta todas las infecciones primarias (infecciones por un programa confiable), y evita todas las infecciones secundarias (infecciones adicionales por un programa infectado al través de infección primaria).

Eficiencia

La segunda característica de los shells de integridad es que son muy eficientes. Cuando se dice eficiente, significa que para una protección óptima, los shells de integridad utilizan el mínimo de sobrecarga (overhead):

- Los shells de integridad no realizan ninguna revisión que no sea con miras a evitar la infección secundaria, y sólo revisan las cosas que tienen que revisarse cuando éstas deban ser revisadas. Por lo que, son eficientes en el sentido de una mínima sobrecarga.

- También son eficientes en el sentido de que pueden automatizar los procesos de reparación. Sin automatización, los costos de la eliminación de un virus una vez detectado son potencialmente enormes. Por ejemplo, si se tienen 5000 computadoras y un virus se dispersa por una organización, se tendrá una tarea gigantesca para limpiarlos. Con una solución automática, no se requiere la intervención de un usuario en el proceso. Este es totalmente automático y transparente.

- Finalmente, son eficientes porque son fáciles de usar. Desde el punto de vista de un usuario, los shells de integridad pueden ser transparentes si se automatizan todas las decisiones basadas en su política de seguridad.

Algunos utilizan una √Σ sin un shell de integridad. Por ejemplo, pueden realizar una √Σ para detectar modificaciones cada mañana. El problema con este esquema, y una de las razones por las que los shells de integridad son óptimos, es que un virus podría estar diseñado para infectar sólo programas modificados durante el último día.

En este caso, sólo cambios legítimos podrán ser detectados por la √Σ, y el virus se propagaría sin ser detectado. Esta es la razón por la cual el shell de integridad debe revisar la información justo antes de utilizarla, y resumirla automáticamente después de usarla sólo si existen modificaciones legítimas.

Confiabilidad

Finalmente, los shells de integridad son confiables si se utilizan en una organización.

- Los shells de integridad son capaces de cubrir toda la información, lo mismo en hojas electrónicas, bases de datos, etc. Ya sea que la información que se desea revisar en busca de algún cambio pueda ser protegida con un shell de integridad porque este la podrá proteger con una √Σ.

- Podemos tener revisiones múltiples independientes para que si alguna de las copias del shell de integridad se encuentra contaminada, la revisión independiente pueda detectarla. Será muy difícil, si no es que imposible, forzar una modificación que sea invariable bajo la totalidad de las claves posibles de un sistema √Σ razonablemente bueno.

- Los shells de integridad trabajan sobre redes ya que la √Σ trabaja sobre redes. Podemos desarrollar un shell de integridad para un servidor de archivo y pasar la información de integridad a través de la red transparentemente hacia los sistemas que están siendo servidos.

- Pueden trabajar a través de diferentes máquinas, debido a que la √Σ es una transformación matemática que puede ser implementada en cualquier tipo de máquina. Puede contar con capacidades para mover una hoja electrónica desde una máquina a otra,

y si ambas cuentan con shells de integridad, aún cuando esta transferencia sean entre sistemas de computadoras diferentes con sistemas operativos diferentes, puede garantizar la integridad sobre la totalidad del proceso.

Aspectos a considerar en el desarrollo de los shells de integridad

Existen tres aspectos principales a tomar en cuenta en el diseño de shells de integridad.

Capacidades básicas

Algunos sistemas cuentan con capacidades de respaldos en línea, otros no. La integridad de la $\sqrt{\Sigma}$ es vital para una apropiada operación de un shell de integridad. Otras características se requieren en distintos ambientes, y su existencia y calidad son críticas para el buen diseño del producto.

Interfases de usuarios

Probablemente necesite una interfase diferente para el administrador del sistema que para el de una secretaria típica. La característica de los shells de integridad debe de caer dentro de las interfaces existentes para que sean lo mas fácil. Por ejemplo, una interfase que no se integre con un ambiente de ventanas resulta no apropiada si este es el ambiente dominante.

Automatización de la toma de decisiones

Debido a que existen bastantes caminos diferentes para usar los shells de integridad en diferentes ambientes, y debido a que distintas decisiones resultan no apropiadas bajo diferentes condiciones, es conveniente que las decisiones sean configuradas para el ambiente.

Desarrollos futuros

Las principales cosas de las cuales carecemos hoy en día son herramientas más generales para refinar la automatización del mantenimiento de la integridad. Nos gustaría contar con un lenguaje para conductas esperadas para que, por ejemplo, si tenemos una hoja electrónica que se supone modifica a otras, con los tres primeros caracteres del primer nombre, el proceso de revisión y de suma podría ser automatizado.

Un lenguaje que especifica intenciones no es tan simple como pudiera creerse. Entre más cerca podamos hallarnos para especificar nuestra intención, podremos hacer un mejor trabajo al determinar si es que nuestra intención ha sido cumplida o no, pero que también implica que debemos de conocer mucho acerca del modo en que trabaja el sistema y ser capaces de transmitir los conocimientos a la computadora. Otro problema al tratar de describir la intención es que ésta es diferente para ambientes diferentes, usos, y gente. No podemos determinar automáticamente qué se está intentando. No obstante podemos lograr acercarnos, y a mayor acercamiento que

logramos, menor será la diferencia que habrá de encontrarse entre la intención real y la intención que debería de ser la especificada, y menor será el grado de vulnerabilidad con que contaremos.

F. Cohen
"A Short Course on Computer Virus"
1990

F. Cohen
"Computer Viruses Theory and Experiments, Computer and Security"
1987

F. Cohen
"A Criptographic Cheksum for Integrity Protection in Untrusted Cumputer and Securtiry"
1987

Charles M. Preston
"Computer Virus Protection"
1993

F. Cohen
"A DOS based POset implementation, Computers and Security"
Vol 10, 1991

Plan corporativo contra virus 8

Plan corporativo contra virus

La solución al problema de los virus y programas nocivos involucra mucho más que el uso de antivirus. Significa la integración de un plan que incluya políticas, reglamentos, una campaña de educación, planes de contingencias, etc. Se trata de reunir todas las soluciones parciales para conjuntar un verdadero esquema antivirus. Sin embargo, un Plan corporativo contra virus puede representar un costo muy alto. ¿Por qué invertir entonces en tiempo, recursos humanos, dinero y esfuerzo?

Hay diversos factores que obligan a incorporar, ante la amenaza de los virus, diferentes políticas corporativas para evitar riesgos:

- Muchas instituciones ahora se comunican a través de redes de computadora. Esto significa que los virus pueden fácilmente infectar computadoras remotas en cuestión de segundos. Además, la mayoría de las infecciones son realizadas accidental e inconscientemente por los mismos administradores de sistemas, operadores y usuarios.

- Muchas de las computadoras usan el mismo sistema operativo y software de aplicación, y más aún, la tendencia es hacia la compatibilidad en el hardware. Esto hace que muchas de estas computadoras sean vulnerables al mismo virus.

- No existen programas de respaldo que puedan evitar respaldar programas y archivos infectados por virus, de tal forma que ningún respaldo de información puede ser seguro,

lo que hace que las labores de restauración y recuperación sean mucho más difíciles actualmente.

La conveniencia de la implantación de un Plan Corporativo contra virus deberá ser analizada según el escenario o panorama de las empresas o instituciones. Así por ejemplo, el panorama de una institución educativa es completamente diferente al de una institución bancaria. Deberán tomarse en cuenta diferentes factores, tales como el valor de la información, el tiempo invertido en obtenerla, el costo de la pérdida de la misma, el tiempo de restauración, etc.; comparados con el costo de la inversión en el desarrollo de dicho plan.

Existen tres partes que conforman un plan corporativo contra virus informáticos:

- Política administrativa, que incluye planes específicos de acción en respuesta.
- Información y educación hacia los empleados.
- Defensas técnicas consistentes en software y hardware.

Política administrativa

Una política administrativa deberá cubrir al menos los siguientes puntos:

Establecimiento de áreas de responsabilidad

- Todo el personal de todos los niveles de la organización deberán tener claramente definidas las áreas de responsabilidad.

- Definir quién estará a cargo de los diferente recursos de cómputo.

- Definir quién hará uso de esos recursos. Muchos sistemas (incluso las computadoras personales actuales) cuentan con mecanismos de passwords que facilitan esta tarea.

- Fijar cómo se hará el uso de los mismos (horarios,etc).

Valoración objetiva de la seguridad de la información

El análisis de riesgo ha variado en dos aspectos básicos desde el advenimiento de los virus de computadoras. El primer cambio es que el riesgo de contaminación ahora está siendo considerado. El segundo cambio, es que ahora se tiene que considerar el flujo de información en los análisis.

Hay diferentes métodos para la valoración de los riesgos informáticos, desde la evaluación cuantitativa hasta el apoyo de expertos consultores.

Como un ejemplo del método de evaluación cuantitativa, la evaluación del riesgo y costo se puede resumir con la siguiente fórmula:

$$R = P \times L$$

donde:

R es el riesgo

P es la probabilidad numérica de la ocurrencia de un ataque

L es la pérdida de información producida por una simple ocurrencia (que se puede expresar en pesos).

Mientras que este método es ampliamente usado, algunos expertos consideran que el cálculo empleado no es suficiente para reflejar el riesgo real de un ataque de virus. Sin embargo, lo que sí se puede establecer es que el costo de protección contra un posible ataque de virus no debería exceder el daño que pudiera producirse.

Desafortunadamente, el análisis de riesgo estándar realiza la consideración de que podemos asociar probabilidades con sucesos. Sin embargo, dado que no se puede tener ninguna certeza sobre los factores que contribuyen al ataque viral, es difícil valorar probabilidades.

Otra alternativa sobre el análisis de riesgos es el análisis de exposición. Las exposiciones son esencialmente los casos de mayores posibilidades de pérdida bajo un escenario dado (universidades, centros de investigación, empresas, etc.).

Supóngase que se tienen la siguiente matriz "sujeto/objeto" comúnmente utilizada para representar el control de acceso en los sistemas de cómputo.

	DB1	FILE1	BUP	FILE2	FILE3	DB2
Héctor	rw	r			r	r
Olimpia		rw	rw			r
Ramón			r	rw		rw
Hugo		r	w	w	rw	

En este modelo, existen sujetos (es decir, Héctor, Olimpia, Ramón y Hugo), objetos (o sea, DB1, FILE1, BUP, FILE2, FILE3 y DB2), y una gama de derechos de acceso (es decir, leer, y escribir). La matriz está supuesta para describir lo que se permite y lo que no se permite. Sin embargo, ésta no lo hace completamente.

La matriz sujeto/objeto dice, por ejemplo, que Héctor puede leer o escribir DB1. Por lo tanto, si Héctor puede escribir DB1 y Héctor puede leer DB1, entonces Héctor puede hacer fluir la información hacia sí mismo. Este concepto se denomina **control de flujo de información**, y se utiliza para describir el flujo de información en un sistema de cómputo.

	Héctor	Olimpia	Ramón	Hugo
Héctor	f			f
Olimpia	f	f	f	f
Ramón	f	f	f	f
Hugo	f	f	f	f

En esta matriz, en vez de tratar con sujetos y objetos y derechos de sujetos sobre objetos, se trata con dominios de información y el flujo de información entre ellos. Así, Héctor puede hacer fluir información hacia Héctor, por lo que colocamos una f en la intersección de Héctor con Héctor en la matriz de flujo. Luego, Olimpia puede escribir algo que Olimpia puede leer, por lo que Olimpia puede hacer fluir información hacia Olimpia. Olimpia puede escribir algo que Héctor puede leer, por lo que Olimpia puede hacer fluir información hacia Héctor. Olimpia también puede hacer fluir información hacia Ramón y Hugo.

En vez de utilizar estas oraciones, se puede abreviar la disposición de las palabras mediante la escritura de ecuaciones como éstas: **Ramón f Héctor** significará "Ramón puede hacer fluir información hacia Héctor"; **Ramón w Héctor** significará "Ramón puede escribir algo que Héctor puede leer"; y **Ramón r Héctor** significará "Ramón puede leer algo que Héctor puede escribir". En conclusión:

Ramón	***w***	***Héctor***	=>	***Ramón***	***f***	***Héctor***
Ramón	***w***	***Olimpia***	=>	***Ramón***	***f***	***Olimpia***
Ramón	***w***	***Ramón***	=>	***Ramón***	***f***	***Ramón***

por lo que si

Ramón f Olimpia y Olimpia f Hugo => Ramón f Hugo

Usando lógica común, si Ramón puede escribir información que Olimpia puede leer, y Olimpia puede escribir información que Hugo puede leer, entonces Ramón puede escribir información que Hugo puede leer indirectamente vía Olimpia. Esto muestra la propiedad de la **transitividad** del flujo de información, que no fue ampliamente reconocida hasta que los virus lo demostraron tan claramente (debido a la dispersión de usuario a usuario). Finalmente:

Hugo	***w***	***Héctor***	**=>**	***Hugo***	***f***	***Héctor***
Hugo	***w***	***Olimpia***	**=>**	***Hugo***	***f***	***Olimpia***
Hugo	***w***	***Ramón***	**=>**	***Hugo***	***f***	***Ramón***
Hugo	***w***	***Hugo***	**=>**	***Hugo***	***f***	***Hugo***

En conclusión, si se observa la matriz de acceso original, se podría pensar que Olimpia no puede leer o escribir FILE2, pero ello no es verdad. La situación real es que todo el mundo excepto Héctor puede enviar información hacia cualquiera de los demás. Entonces lo que parece especificar la matriz de acceso, no es lo que realmente significa. Las probabilidades, cualquiera que sea el sistema que se tenga hoy en día, se dan como en el caso anterior. Se cuenta con controles de acceso de lectura y escritura sobre la información, pero si realmente se observa lo que ello implica, las posibilidades son que prácticamente cualquiera puede enviar información hacia el resto de los usuarios, es decir, cualquiera en el sistema puede dañar o filtrar parte o la totalidad de la información.

por lo que si

Ramón f Olimpia y Olimpia f Hugo => Ramón f Hugo

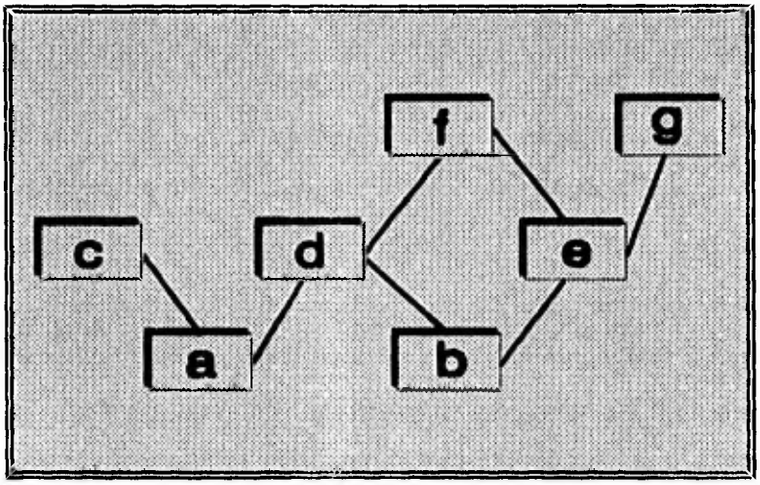
Usando lógica común, si Ramón puede escribir información que Olimpia puede leer, y Olimpia puede escribir información que Hugo puede leer, entonces Ramón puede escribir información que Hugo puede leer indirectamente vía Olimpia. Esto muestra la propiedad de la **transitividad** del flujo de información, que no fue ampliamente reconocida hasta que los virus lo demostraron tan claramente (debido a la dispersión de usuario a usuario). Finalmente:

Hugo	***w***	***Héctor***	***=>***	***Hugo***	***f***	***Héctor***
Hugo	***w***	***Olimpia***	***=>***	***Hugo***	***f***	***Olimpia***
Hugo	***w***	***Ramón***	***=>***	***Hugo***	***f***	***Ramón***
Hugo	***w***	***Hugo***	***=>***	***Hugo***	***f***	***Hugo***

En conclusión, si se observa la matriz de acceso original, se podría pensar que Olimpia no puede leer o escribir FILE2, pero ello no es verdad. La situación real es que todo el mundo excepto Héctor puede enviar información hacia cualquiera de los demás. Entonces lo que parece especificar la matriz de acceso, no es lo que realmente significa. Las probabilidades, cualquiera que sea el sistema que se tenga hoy en día, se dan como en el caso anterior. Se cuenta con controles de acceso de lectura y escritura sobre la información, pero si realmente se observa lo que ello implica, las posibilidades son que prácticamente cualquiera puede enviar información hacia el resto de los usuarios, es decir, cualquiera en el sistema puede dañar o filtrar parte o la totalidad de la información.

Ahora se puede entender la forma en que los sistemas pueden ser modelados y analizados utilizando flujos de información, es decir, **el análisis de exposición.**

Esto nos proporciona un camino eficiente y fácil de usar para prever los efectos de una infiltración. La técnica es simple: primero, se tiene que considerar la transitividad mediante la obtención de todos los flujos de información directa; esto se hace estableciendo un diagrama POset para después vaciarlo en la matriz de flujo de información. Posteriormente, se añaden los flujos de información indirecta derivados de las relaciones anteriores.



POset

El siguiente paso es analizar los efectos de la contaminación y filtrado de información en la organización; esto se realiza siguiendo la matriz de control de flujo a partir de un punto determinado y observar hasta dónde puede llegar el daño. Por ejemplo, en la siguiente matriz:

	a	b	c	d	e	f	g
a	f		f	f		f	
b		f		f	f	f	f
c			f				
d				f		f	
e					f	f	f
f						f	
g							f

si "a" lanza un virus, potencialmente puede infectar a "a", "c", "d" y "f"; mientras que si es "b" quien lanza el virus, entonces el riesgo es para "b", "d", "e", "f" y "g" (esto también se puede observar directamente en el POset). En este caso (y en el de muchos otros), con estas dos únicas infecciones, se puede contaminar todo el sistema.

Una vez realizado lo anterior, lo siguiente es valorar cada una de las infiltraciones y contaminaciones posibles, ponderando de acuerdo al daño financiero que pudiera provocar la corrupción de algún elemento de la tabla. Aquí, es recomendable establecer una diferencia entre el daño global y el daño local. Por ejemplo:

	Filtración Local	Corrupción Local	Filtración Local	Corrupción Local
a	25	75	25	1850
b	250	10	250	1190
c	1000	25	1025	1000
d	750	250	1025	825
e	90	200	340	190
f	75	90	1190	75
g	25	1000	365	1000

El daño local (efecto directo) es el valor asociado al daño que, en el mejor de los casos, se produzca sobre de ese único elemento. El daño global (efecto indirecto) es el valor asociado al daño de un elemento que, en el peor de los casos, se produzca como producto de una contaminación general.

En el ejemplo anterior, una corrupción en b podría afectar b, d, e, f y g (las columnas señaladas con f en el renglón de b del flujo de la matriz de control). Si se suman los valores locales de corrupción del renglón b para esos dominios, el resultado será la corrupción global en la tabla. El análisis de filtrado es esencialmente el mismo, excepto que se suman las columnas en lugar de los renglones.

De lo anterior, se deduce que una de las cosas que debe ser considerada en cualquier análisis de riesgo, es el nivel de importancia de los sistemas de cómputo, redes o sistemas

individuales y la información que almacenan. Existen al menos una serie de puntos que deben ser considerados:

- El valor en tiempo de la información almacenada. Hay información, producto del trabajo de días, meses o años, que difícilmente podría recuperarse si es alterada por un ataque de virus.

- El costo de reemplazo o restauración del software o datos, consecuencia de la pérdida de su integridad.

- El grado de conectividad entre sistemas de cómputo. La forma en que se encuentran conectados los diferentes equipos de una empresa puede facilitar la proliferación de virus.

A partir de estas consideraciones se puede establecer una tabla de prioridades e importancia:

Sistemas	Tipo de información
De alto riesgo	Contienen información corporativa, contable, etc
De administración	Contienen información estadística y ejecutiva para la toma de decisiones.
De producción	Contienen los programas y herramientas que puedan obtener la información de la empresa y procesan los resultados para los niveles superiores.
De desarrollo	En donde se realizan los programas y herramientas para obtener la información de la empresa.
De bajo riesgo	Archivos y programas de usuario, sino reemplazables, fáciles de recuperar.

El propósito de uso del equipo o uso de la computadora

Este punto deberá definir:

- El tipo de tareas que desempeñarán los sistemas: administrativas, operativas, de producción, de desarrollo, etc.

- El número de usuarios que harán uso de los sistemas. El número de personas que tienen acceso al sistema de cómputo, así como los privilegios que tienen para realizar ciertas tareas.

- La información que deberán almacenar los sistemas.

Manejo de la información y recursos de software

Una medida que algunas Instituciones han empleado es la centralización de la administración de la información, es decir, el manejo de la misma está asignada a un sólo departamento o grupo de gente. Esto cumple con el principio de "entre menos manos toquen la información, estará más segura"

Siguiendo ésta idea, la manera más eficaz de combatir un posible contagio por virus es centralizar la adquisición y la administración del software que se utiliza, por lo que se deberá definir:

- El software que se utilizará para cualquier fin (software institucional). Algo que se debe tomar en cuenta, es que dicho software deberá cumplir con los requerimientos del usuario y ser "compatible" entre una aplicación y otra. La razón del fracaso de los esfuerzos por combatir la proliferación de virus se debe a que en muchas ocasiones se toma a la ligera este punto.

- Una serie de políticas sobre el software no institucional, estableciendo mecanismos para adquirir y probar nuevo software.

- Un esquema de procedimientos para el control del intercambio de información (ya sea de manera interna o con otras empresas o instituciones), así como autorizar únicamente medios de respaldo revisados, certificados y etiquetados.

Cumplimiento de las políticas

Se deben establecer sistemáticamente auditorías internas, de manera que se obligue al usuario a depurar su información constantemente (sobre todo aquel que hace uso de software no institucional).

Un punto que hay que cuidar mucho es que las políticas corporativas contra virus no deben hacer sentir al empleado que será despedido si reporta un problema de virus en su máquina.

En lugar de multas o sanciones lo recomendable es asociar promociones o ascensos a quienes observen un conducta acorde con las políticas. Esto hará sentir al usuario que su desarrollo personal no sólo depende de su trabajo, sino de su comportamiento.

Planes de acción en respuesta

Parte importante de una política corporativa contra virus son las acciones y procedimientos, entre otros:

- Dado que el uso de la computadora se extiende frecuentemente a todos los niveles, todo el personal necesita procedimientos específicos sobre qué hacer cuando se presume la presencia de virus.

- La defensa exitosa contra el ataque de un virus depende muchas veces de las apropiadas prácticas de seguridad en computación dentro de la organización. Deberán existir manuales de procedimientos sobre el uso de los recursos (encendido y apagado de los sistemas, uso del software de diagnóstico, etc.) Algunas malas costumbres aumentan el riesgo de contagio, por ejemplo, empezar a trabajar con una computadora que ya se encuentra en sesión o encendida.

- Realizar guías donde se describen los síntomas más comunes para detectar la presencia de un programa nocivo. Como sugerencia, lo primero que se debe especificar es que muchos problemas en los sistemas son realmente fallas de hardware o software y no de virus; en primera instancia, y por razones de salud mental, debe considerarse que no existe la presencia de un programa nocivo.

Dependiendo del análisis de los puntos anteriores, se puede requerir establecer una serie de medidas de protección, como por ejemplo:

- Aislamiento de otros sistemas
- Restringir la función del sistema
- Control de acceso físico
- Control de acceso por hardware o software y/o encriptamiento
- Control de integridad de hardware o software con checksums y búsqueda de virus
- Uso de diagnosticadores
- Medidas específicas que restrinjan la operación del equipo

Educación e información al usuario

Una forma de unir esfuerzos para hacer de las políticas corporativas un éxito, es la educación en este tema a todos los niveles; es muy importante hacer sentir al usuario que la información que maneja es vital para la empresa o institución y que debe responsabilizarse de los recursos de cómputo que utiliza.

La proliferación de virus se debe a que el usuario no es capaz de determinar si hay virus o no en su sistema (la idea que se tiene, es que mientras funcione bien no hay virus, si falla invariablemente se debe a uno de ellos).

Es necesario establecer un esquema de capacitación, a todos los niveles, que permita unificar conocimientos y criterios respecto del fenómeno y aplicar regularmente encuestas sobre el comportamiento de los sistemas.

Es importante establecer un sistema de información que permita avisar a los usuarios de ocurrencias de proliferación que los mantenga alertas, así como de nuevos productos y herramientas para detectar y restaurar su información.

Defensas técnicas consistentes en software y hardware

Para complementar un plan corporativo contra virus, independientemente de qué tan bien se implanten las políticas administrativas y los esquemas de capacitación, es necesario tener una serie de productos de software o hardware como defensas optativas.

La mejor forma de seleccionar es analizar los costos de las defensas con el fin de determinar cuál es la que mejor inversión para las necesidades específicas de los diferentes tipos de escenarios. Los principales productos que hay en el mercado para la defensa contra virus hoy en día son los rastreadores (scanners) de virus, los monitores de virus (programas que exploran en busca de virus conocidos justo antes de la ejecución), sumas de comprobación criptográficas (cheksums) y shells de integridad, por lo que el análisis estará dedicado estos productos.

A continuación se muestra una lista de los parámetros que se deben de considerar en un análisis de costos generados al utilizar diferentes herramientas de defensa:

T_s	Costo total por adquisición de programas scanners
s	Número de sistemas
T_c	Costo total por adquisición de programas de Cheksum criptográfico
c	Chequeos del sistema por año
T_m	Costo total por programas de monitoreo
e	Costo del pago al empleado asignado por minuto
T_i	Costo total por adquisición de shells de integridad
u	Costo de distribución por actualizaciones (dist-cost*update-count)

t_s Minutos empleados por uso de scanners

t_c Minutos empleados en chequeo

l_m Costo de la licencia por uso del scanner

l_i Costo de la licencia por uso de cheksum criptográfico

l_m Costo de la licencia por uso de programas monitores

l_i Costo de la licencia por uso de shells de integridad

a_n Nuevos ataques

a_o Viejos ataques

r_s Costo de limpieza del sistema

r_f Costo de limpieza de archivos

d Costo de distribución

o_i Tasa de dispersión del virus [K/c]

La mayoría de estos términos se explican por sí mismos, pero vale la vena aclarar algunos otros. Los costos de licencia de uso de cheksums criptográficos y de shells de integridad son normalmente costos de una sola vez, en tanto que para los scanners y monitores, las actualizaciones regulares hacen necesario que las licencias sean pagadas sobre el tiempo. Para compensar, se utilizará el 10% por año del costo total de la licencia para shells de integridad y cheksums criptográficos como equivalente del costo de licencia anual para scanners y monitores. o_i es un término que describe la tasa de dispersión de un virus, y experimentalmente han sido de cerca de 2 para una PC en un ambiente típico y cerca de 10 para una PC típica en un ambiente LAN.

Según Fred Cohen, las ecuaciones para evaluar el costo total por adquisición y uso de las defensas antes mencionadas son las siguientes:

$$T_s = s[cet_s + l_s + u] + a_n r_s s + a_o r_s o_i + d$$

$$T_c = s[(cet_c) + l_c] + [a_n + a_o] r_s o_i + d$$

$$T_m = s[l_m + u] + a_n r_s s + a_o r_f + d$$

$$T_i = s l_i + [a_n + a_o] r_f + d$$

Los resultados de éstas ecuaciones deberán ser avaluados para definir cuál o cuáles son los que mejor convienen según la importancia de la información y los recursos con que cuente la Empresa e Institución.

F. Cohen
"A Short Course on Compueter Virus"
ASP Press
1990

Juan José Nombela
"Virus Informáticos"
Paraninfo
Segunda edición, 1991

F. Cohen
"A DOS Based POset Implemetation Computers and Security"
Vol. 10, 1991

Charles M. Preston
"Computer Virus Protection"
Digital Equipment Corp.
1993

Estudio de casos reales 9

Como se ha mencionado, el advenimiento de los virus informáticos no sólo ha provocado situaciones desagradables, sino que ha traído consigo toda una nueva forma de ver a los sistemas de cómputo y, quizá lo más importante, una nueva forma de elaborar programas. Es muy interesante el estudio del código de programas nocivos, ya que proporciona ideas de cómo plantear sistemas de cómputo más seguros, independientemente del conocimiento que se aporta para erradicar dichos programas.

Este capítulo está dedicado al estudio de casos reales de programas nocivos, tanto en sistemas multiusuario como en sistemas personales (PCs). De hecho, no se pretende mostrar la descripción absoluta de todas las instrucciones de cada uno de estos programas, sino establecer los puntos más importantes (forma de ejecución, forma de infección, manejo de memoria, disparadores, etc.) que aporten ideas que lleven a soluciones factibles y, sobre todo, económicas en la elaboración de sistemas de cómputo más seguros.

Programas nocivos en sistemas multiusuario

Los programas nocivos en sistemas multiusuario no son tan comunes como en las computadoras personales, de hecho se puede hablar de un número limitado de casos. Sin embargo, estos son los que más han causado pérdidas, ya que se han presentado bajo esquemas de red a nivel internacional y casi siempre como "gusanos".

En esta sección se presentarán los casos del "gusano de la red Internet" que atacó sistemas UNIX y el gusano "WANK" basado en lenguaje de comandos DCL para el Sistema Operativo VMS.

El gusano de la red Internet

El 2 de noviembre de 1988, dos de las más importantes redes estadounidenses de computadoras conectadas a Internet fueron .víctimas de un gusano que se introdujo en ellas. El culpable del contagio informático más extenso ocurrido hasta hoy fue Robert Morris Jr., un estudiante de 23 de la Universidad de Cornell, a quién se le encontraron códigos a los que no tenía derecho de acceso similares a los utilizados para propagar el gusano. Esto propició que se pusiera en duda el nivel de seguridad que ofrecían los sistemas UNIX.

UNIX ya ha recuperado su credibilidad en lo que se refiere a la seguridad. Era obvio pensar que sólo una persona que supiera "secretos" sobre UNIX pudiera crear un programa que se infiltrara en tantos equipos de diferentes arquitecturas y marcas. Robert Morris Jr. es el hijo de aquel que fuera parte del grupo que desarrolló el concepto de "core wars" y que pertenecía al equipo de trabajo de Ken Thompson (el desarrollador de UNIX).

Forma de contagio

El código del gusano que desarrolló Robert Morris Jr., estaba hecho parte en shell scripts (archivos de comandos de shell en UNIX) y parte en lenguaje C. Usaba tres mecanismos para infiltrarse en las redes:

- Ejecutando el comando rsh (remote shell command)

- Usando una puerta de acceso dejada en el programa "sendmail"

- Modificando el buffer del stack del programa "fingerd"

El comando rsh permite a usuarios en equipos considerados "de seguridad" (trusted) ejecutar comandos en máquinas remotas. De ésta manera, el posible ejecutar cualquier comando disponible en todos los nodos de la red. Los archivos /etc/hosts.equiv y .rhosts son los usados para obtener el acceso. La sintaxis del rsh es la siguiente:

rsh <nodo remoto> <comando>

Otra forma de penetración fue la usada en SMTP (Simple Mail Transfer Protocol) a través de la puerta de acceso del programa "sendmail". Algunas versiones de UNIX fueron compiladas con la opción "debug", por lo que contenían dicha puerta de accesso. El gusano usó este medio para pasar una pequeña secuencia de carga al sistema remoto, la cual se encargaba de ejecutar al gusano.

Un tercer método de penetración fue la alteración del programa fingerd. Este comando fue diseñado usando la función "gets()", que emplea un buffer de 512 bytes. El gusano sobrescribía el contenido de este buffer de tal manera que la dirección de regreso que enviaba provocaba que, en vez de terminar, ejecutara un shell (intérprete de comandos) en el sistema remoto.

Además, Morris incluyó su propia versión del algoritmo de encriptamiento de palabras clave (passwords) de UNIX. Este corría nueve veces más rápido que el original y comparaba los passwords generados con tabla de usuarios de UNIX (/etc/passwd). Gracias a la pobre selección de passwords de los usuarios, este procedimiento fue muy exitoso, ya que usaba varios criterios:

- Asumía que el usuario no tenía password
- El password era el mismo nombre del usuario
- El password era dos veces el nombre del usuario
- El password era el nombre del usuario o su apellido escrito al revés
- El password era un apodo o nombre personal (obtenido del /etc/passwd)
- El password era alguno de una listas previa de passwords exitosos
- El password era cualquiera de las palabras del diccionario (/usr/man/dict)

Además, para poder ejecutarse en máquinas que, aún teniendo a UNIX como sistema operativo, no eran de la misma arquitectura, lo que hacía era compilar nuevamente el código fuente del programa en el nodo remoto. De esta manera, siendo el lenguaje de programación C estándar a nivel código fuente, no había ningún nodo en las red en donde no se pudiera ejecutar.

Conclusiones

La idea general del programa era que corriera un número limitado de gusanos en cada computadora de la red. Si esto hubiera ocurrido, probablemente nunca se hubiese detectado, pero por alguna razón el programa no corrió como fue diseñado. Fue por esto que la mayoría de las computadoras que fueron penetradas tuvieron sobre-carga de procesos, trabajaron excesivamente lentas y se cayeron. Cerca de 6000 computadoras fueron infectadas y se requirieron 8000 horas para removerlo.

Morris afirma que él fue el único que escribió el código programa; sin embargo, el hecho de que algunas partes del código fueran escritas de manera muy elegante y eficiente, y otras con muchos errores, parece decir que por lo menos otra persona colaboró en su realización.

El gusano WANK en la red SPAN

El 16 de Octubre de 1989, un gusano se introdujo en la Red SPAN atacando sistemas VAX de Digital. Este atacaba únicamente a sistemas operativos del tipo VMS a través del protocolo DECnet. Aunque había otro tipo de conexiones de red, el gusano no fue programado para tomar ventaja de esta situación. De todas formas, esto mostró serias deficiencias en el que se consideraba uno de los sistemas operativos más seguros en el mercado.

El gusano WANK (Worms Against Nuclear Killers) tomaba ventaja de la pobre administración de passwords por parte de los administradores de sistemas creando cuentas

nuevas; era capaz de modificar archivos .COM (archivos de comandos en DCL) y de diseminaba a nodos remotos a través de el protocolo de red DECnet.

Forma de contagio

El programa W.COM fue ejecutado desde una cuenta privilegiada cuyo dueño tenía habilitados todos los permisos (lectura, escritura, ejecución y borrado). Una vez ejecutado, procedía a cambiar el password de la cuenta que por defecto (default) usa el protocolo DECnet para accesar a nodos remotos; este password consistía de una palabra de 12 caracteres aleatorios, misma que enviaba vía correo electrónico (mail) a un usuario llamado GEMPAK en una computadora remota llamada SPAN, donde recolectaba la información para ser usada por las nuevas versiones del gusano.

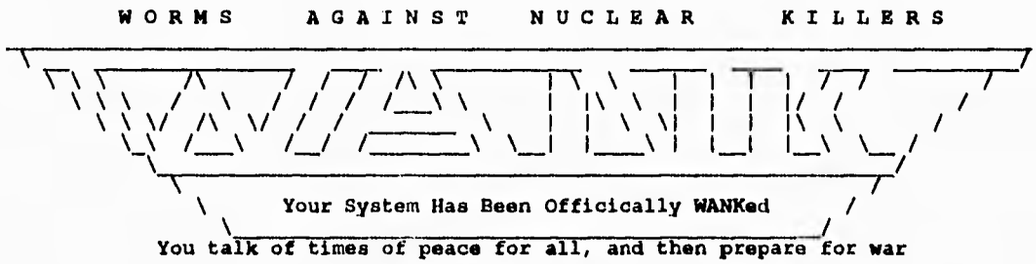
Por otro lado, el gusano rastreaba la tabla lógica de nombres de usuarios (definida para procedimientos de comandos) y trataba de modificar la cuenta de FIELD (cuenta con privilegios que se usa para dar mantenimiento al sistema) colocando un password conocido.

Una vez logrado lo anterior, intentaba accesar los nodos remotos a través de una lista de usuarios obtenida por el comando PHONE. El procedimiento para "adivinar" los passwords era asumir que eran el mismo nombre del usuario o simplemente no tenían. Si lograba penetrar a una cuenta con privilegios, copiaba el código y los ejecutaba.

Mecanismo de disparo

Cuando el gusano era ejecutado, éste revisaba si ya había otro proceso corriendo en memoria (alguno cuyos cinco primeros caracteres del nombre fueran "NETW_"). Si lo encontraba, entonces se destruía a sí mismo y terminaba su ejecución.

Una vez descartada la posibilidad de que otra copia estuviera en ejecución, verificaba si tenía privilegios del tipo SYSNAM y sobrescribía el mensaje de anuncio del entrada al sistema para que los usuarios, al entrar al sistema, recibieran:



Además, si el proceso tenía privilegios de SYSPRIV, deshabilitaba el mail de la cuenta del administrador (system) y modificaba el archivo de procedimientos de comandos de entrada (login) para dar la impresión de que eran borrados los archivos del usuario, lo cual no era cierto pero creaba histeria masiva.

Otro método que empleaba para irritar a los usuarios, era enviar llamadas a través del comando PHONE. Una vez que WANK obtenía la lista de usuarios, se hacía notar haciendo llamadas insistentes sin dar oportunidad al usuario de trabajar a gusto.

Conclusiones

El gusano WANK fue desarrollado en DCL (el lenguaje intérprete de comandos del sistema operativo VMS), lo que muestra que un virus no necesariamente tiene que ser un binario ejecutable hecho en ensamblador o lenguaje de alto nivel.

Por otro lado, el código del gusano almacenaba todas las operaciones que realizaba, es decir, se "acordaba" de las acciones que había realizado una copia anterior. Esto muestra claramente el concepto de la "genética" de programación, base del desarrollo de los virus mutantes actuales, que se automodifican en base versiones anteriores.

Una forma de evitar que el gusano penetre algún sistema es "engañándolo", esto es, ejecutando un proceso cuyo nombre inicie con "NETW_". A continuación se muestra el código de un programa en DCL para evitar el contagio. Cabe aclarar que este código sólo funciona si el gusano corre bajo la cuenta de SYSTEM y deberá modificarse para mutaciones de este gusano.

BLOCK_WORM.COM

```
$ set default SYS$MANAGER
$ Create BLOCK_WORM.COM
$ Deck/dollar=END_BLOCK
$ LOOP:
$    set process/Name=NETW_BLOCK
$    wait 12:0
$    Goto LOOP
END_BLOCK
$ run/input=SYS$MANAGER:BLOCK_WORM.COM/ERROR=NL:/output=NL:/ -
UIC=[1,4] SYS$SYSTEM:LOGINOUT
```

Otra medida de seguridad, es borrando el objeto TASK de la base de datos de NCP (network control program), que es la que permite ejecutar procedimientos de comandos en sistemas remotos; esto impediría que, aún entrando al sistema, se pudiera ejecutar el gusano. Lo que se debe hacer es modificar el archivo SYS$MANAGER:STARTNET.COM, colocando la instrucción:

CLEAR OBJECT TASK ALL

exactamente después de la línea que contiene la instrucción:

SET KNOWN OBJECTS ALL

Programas nocivos en sistemas personales

Los equipos de cómputo que han sido mayormente afectados por la aparición de los virus, y en general de programas nocivos, han sido los equipos personales, en particular las computadoras del tipo PC (Personal Computer). En esta sección se presentará la descripción de uno de los virus que más ha causado temor, el virus Miguel Angel, ya que acaparó incluso la atención de los medios de comunicación. Del mismo estilo se presentará el caso del "virus de Pakistán" (también conocido como BRAIN) y por último, el del virus "viernes 13" (mejor conocido por jerusalem).

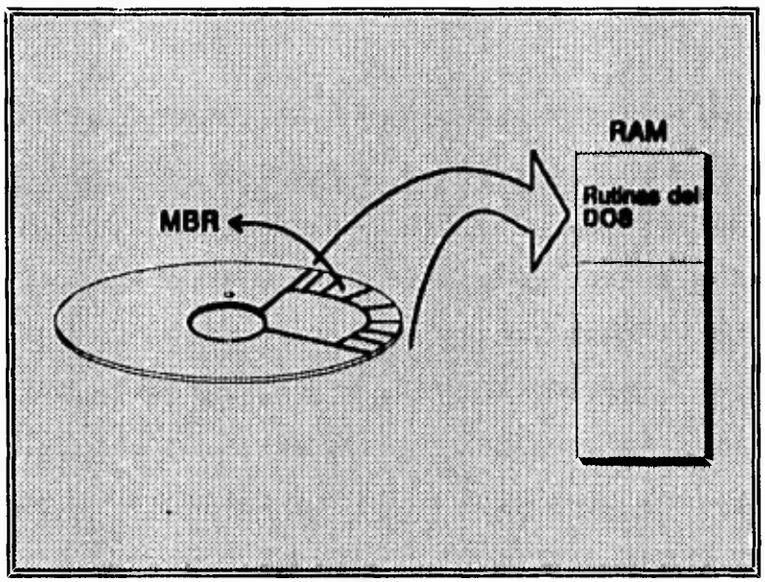
El virus Miguel Angel

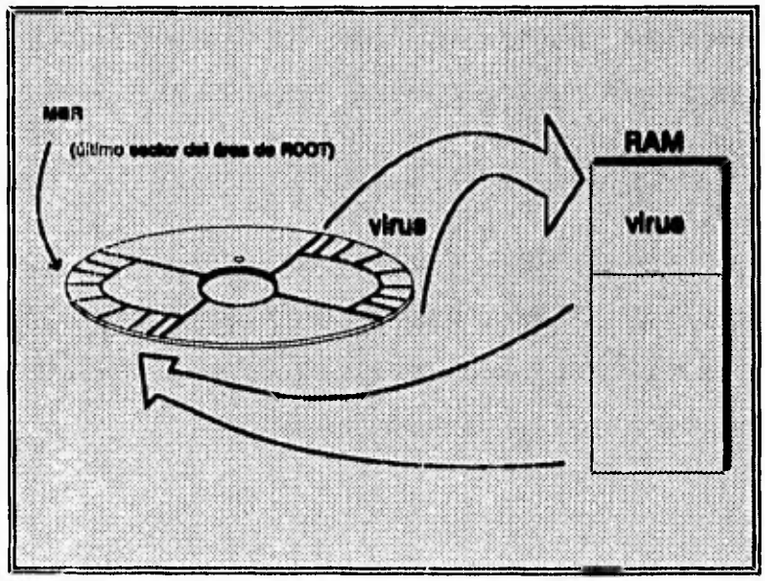
Miguel Angel es uno de los más famosos virus de computadora, su nombre lo debe a que su fecha de disparo (6 de marzo) coincide con el aniversario del natalicio del pintor y escultor Miguel Angel. La fama de este virus se debe principalmente a que los medios de comunicación se encargaron de hacerlo ver "extremadamente peligroso, despiadado e imbatible", sin embargo no es más que uno de tantos virus de sector de BOOT. Esto significa que este virus sustituye parte del Master Boot Record (MBR), que es el área que contiene las instrucciones para la ejecución y arranque del sistema operativo. En este sistema operativo **todos** los discos tienen MBR; de hecho, éste código es el que envía el mensaje de "Disco defectuoso o sin sistema" cuando los programas de arranque de MS-DOS no están presentes.

Forma de contagio

La única forma en que este virus puede infectar al disco duro de una computadora es al intentar arrancar la computadora con un disco contaminado en la unidad de disco flexible denominada A. Al momento que se intenta iniciar una sesión con un disco flexible infectado (ya sea que tenga o no el Sistema Operativo), el virus quedará residente en memoria. Entonces, Miguel Angel lee el Master Boot Record (MBR) del disco duro para ver si está infectado. Si el disco duro está libre de infección, lo infecta; para esto primero llama una rutina que copia el MBR en una localidad no utilizada del disco (un sector vacío entre el primer sector del disco y el principio de la primera partición). Entonces reemplaza el MBR con el código viral que se ejecutará la próxima vez que la PC se arranque desde el disco duro. La tabla de particiones permanece intacta, por lo que no es posible decir que el disco duro está infectado solamente al inspeccionar la tabla de particiones. La próxima vez que el sistema de arranque desde el disco duro, Miguel Angel será cargado a memoria de la misma forma que si se iniciara con el disco flexible infectado.

Cuando Miguel Angel infecta un disco flexible, hace una copia del sector de boot al último sector del área de directorios y reemplaza el código viral por el código de arranque (un cierto número de sectores son reservados para el área de directorios, este número depende de la densidad a la que se formatean los discos).



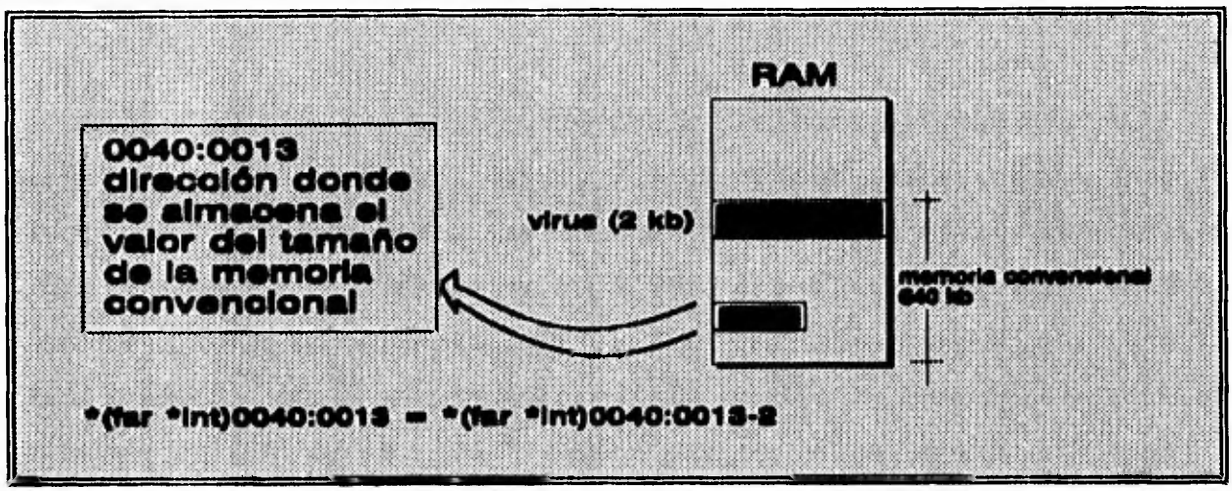


Forma de contagio de Miguel Angel

Forma de ocultamiento

Cuando se inicia una sesión desde el disco duro, el código viral del sector de boot toma el control. Primero, recupera el contenido de la palabra (2 bytes) de la dirección absoluta 0040:0013, que especifica el total de kilobytes presentes de memoria convencional, y le resta 2 para hacer que la computadora aparezca con 2k menos de RAM de la que realmente tiene. Entonces se copia el código del virus en los 2k de la parte superior de RAM que estarán ocultos para el resto del sistema. Por esto Miguel Angel no se muestra como parte de la cadena de DOS de bloques de control o como un programa residente (TSR), dado que DOS se carga hasta después del virus. Enseguida, Miguel Angel redefine la dirección de la interrupción 13H en el vector de interrupciones y hace que apunte a una de sus rutinas. La interrupción 13H es el puente para los servicios de disco del BIOS; al atrapar esta interrupción, Miguel Angel puede monitorear toda actividad de acceso a discos.

Como una medida de protección, Miguel Angel únicamente infecta los discos flexibles si el controlador de disco flexible se encuentra prendido, para lograr esto se inspecciona el bit 0 del byte de la dirección absoluta 0040:003FH. No todas las llamadas a la interrupción 13H causan que el motor del disco se encienda y el disco comience a girar. Al revisar si el motor del disco está funcionando, el virus asegura que la luz del drive no se encenderá sin causa aparente provocando la sospecha de la presencia de una virus.

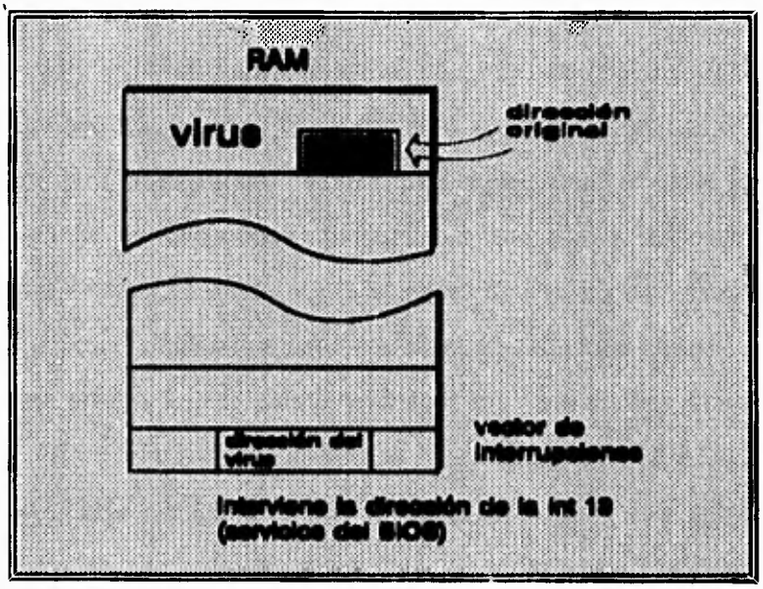


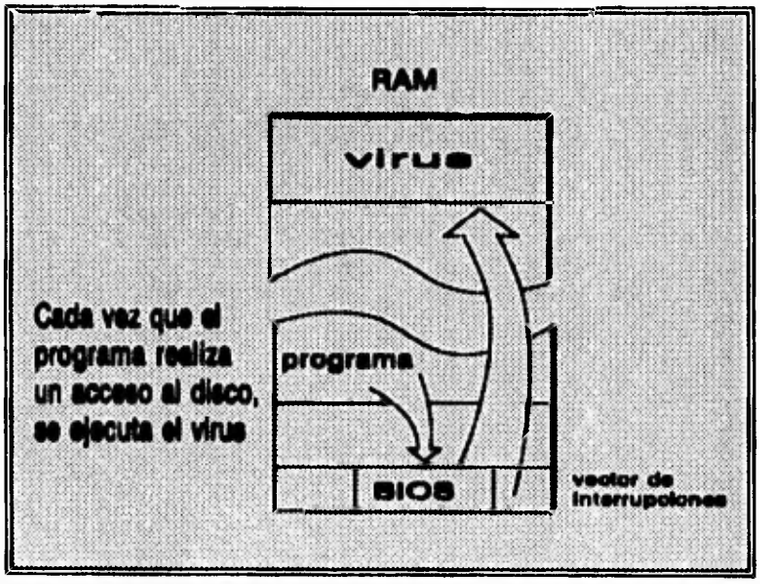
Forma de ocultamiento de Migule Angel

Mecanismo de disparo

Una vez que Miguel Angel está residente en memoria, intercepta todas las llamadas a la interrupción 13H, como son lectura, escritura o cualquier acceso a disco. Cuando una llamada es interceptada, el virus revisa si el disco está infectado por el mismo. Si lo está, Miguel Angel responde al llamado con la interrupción 13H del BIOS. Si el disco flexible no está infectado, Miguel Angel pasa la llamada al BIOS y luego devuelve el control del sistema cuando regresa de la llamada. En este momento, Miguel Angel infecta el disco flexible.

Cada vez que se levanta el sistema, Miguel Angel utiliza la interrupción 1AH, con la función 04H para recuperar la fecha actual desde el BIOS. Si en el registro DX se detecta un 0306H (mes=3 y día=6, o marzo 6), el virus invoca una subrutina interna que destruye los datos del disco al llamar repetitivamente la interrupción 13H con la función 03H (función del BIOS para escribir un sector a disco) para copiar datos aleatorios de memoria a disco, alternando las cabezas del disco para asegurarse que la destrucción es continua. Al tiempo que uno espera que el sistema inicie, las áreas de arranque son dañadas. Los respaldos de los archivos del disco duro no son suficientes para reparar el daño; se necesita un respaldo de la tabla de particiones para reconstruir el disco duro. Si se utiliza el sistema operativo DOS 5.0, se puede copiar la tabla de particiones a un disco flexible con el comando MIRROR /PARTN. En caso de no tener un respaldo de esa tabla de particiones, se pueden establecer nuevamente las particiones con el comando FDISK.





Mecamismo de disparo de Miguel Angel

Forma de Detectarlo

El virus Miguel Angel está lejos de ser perfecto. Una forma fácil y práctica de suponer la existencia de un virus, y específicamente de Miguel Angel, es utilizar el comando CHKDSK o MEM; si estos reportan 2k menos de memoria convencional es muy probable que exista el virus activo.

Conclusines

Erradicar a Miguel Angel de un disco duro es fácil: se restaura el master boot record y se reinicia el sistema. Si se está utilizando DOS 4.X o DOS 5.0, se puede usar el comando FDISK con el switch /MBR para hacer una copia del Master Boot Record del disco duro.

Otros virus no son tan fáciles de remover. El virus Joshi, por ejemplo, intercepta las llamadas a la interrupción 13H y silenciosamente evade el intento de restaurar el master boot record. FDISK /MBR parece trabajar, pero no lo hace. Si cualquier programa intenta leer el sector del disco duro que el contiene el master boot record, Joshi intercepta la petición y direcciona el programa para copiar el master boot record original archivado en el disco. La única forma de eliminar a Joshi el arrancar con un disco flexible sin infectar y luego utilizar FDISK /MBR para restaurar el master boot record y quitar el virus.

El virus de Paquistán

El virus de Paquistán, al igual que todos los demás virus conocidos, ha sufrido una serie de mutaciones, adiciones, modificaciones, etc., que propician que cada investigador que lo llega a percibir se refiera a él de manera diferente.

Este virus ocupa 9 kbytes de memoria, y cuando está presente en la computadora hace muy lentos los procesos de acceso al disco, sobre todo cuando busca algún disco al cual contagiar y éste se halla protegido contra escritura.

Forma de ocultamiento

A diferencia de otros virus, el virus de Paquistán sí graba su código en área de carga inicial (BOOT), sobre todo en los primeros 32 bytes, sección conocida como el BIOS Parameter Block (BPB). El área de BOOT original es copiada en el primer sector vacío que encuentra, así como el resto del virus en los sectores siguientes, marcando todos éstos como sectores dañados. Cuando se inicia una sesión desde un disco duro o diskette infectado, el código viral del sector de BOOT toma el control. Primero, recupera el contenido de la palabra (2 bytes) de la dirección absoluta 0040:0013, que especifica el total de kilobytes presentes de memoria convencional, y le resta 9 para hacer que la computadora aparezca con 9k menos de RAM de la que realmente tiene (exactamente igual que el virus Miguel Angel). Entonces se copia el código del virus en los 9k de la parte superior de RAM que ahora están ocultos para el resto del sistema. En el caso del Paquistán, este virus sólo intercepta las rutinas de lectura y escritura de la interrupcion 13H (servicios del BIOS).

Mecanismo de disparo

Una vez que el virus de Paquistán está residente en memoria, intercepta las llamadas de lectura y escritura de la interrupción 13H. Cuando una llamada es interceptada, el virus revisa si el disco está infectado por el mismo. Si lo está, responde al llamado con la interrupción 13H del BIOS. Si el disco no está infectado, entonces procede a hacerlo; cabe aclarar que esto hace muy lento al sistema, lo que permite despertar sospechas.

Al virus de Paquistán le han sido desactivadas las rutinas que dañan archivos de manera aleatoria, pero hay antecedentes de que sus primeras versiones eran altamente peligrosas.

Forma de Detectarlo

Un problema por el cual se puede sospechar la presencia del virus, es que no posee un medio eficaz para detectar la protección contra escritura, por lo que trata siempre de escribir y provoca que sea notorio el retraso al tiempo de lectura y escritura de datos en discos flexibles.

Por otro lado, una forma muy sencilla de descubrir su presencia es revisando el área de BOOT del disco, a través de utilerías de monitoreo. Los mensajes que aparecen en un diskette infectado resultan muy obvios (de hecho, tienen ese propósito). El caso de un disco duro resulta menos obvio, pero las características son similares.

Conclusiones

Es muy interesante observar la evolución de dos virus que tienen casi cinco años de diferencia. Primero, el virus de Paquistán ocupa 9K en RAM, mientras que el virus Miguel Angel sólo 2K. Otro aspecto interesante, es que el virus Miguel Angel no deja rastro de su presencia en discos (dado que se oculta en áreas del sistema), mientras que el de Paquistán hace uso de áreas de datos que marca como dañadas.

Al igual que el virus Miguel Angel, lo único que se requiere para erradicar al virus Paquistán de un disco es restaurar el master boot record y reiniciar el sistema. Si se está utilizando DOS 4.X o DOS 5.0, se puede usar el comando FDISK con el switch /MBR para hacer una copia del Master Boot Record.

El virus de Jerusalem

A diferencia de los dos virus anteriores, éste es un virus infector de archivos ejecutables (programas con extensión .EXE o .COM) y es uno de los más peligrosos que hemos tenido la oportunidad de estudiar.

Se le conoce como virus Jerusalem, Israelí o del Viernes 13 y se ha difundido mucho en Estados Unidos, España y en toda Latinoamérica, por lo que es muy famoso.

Forma de ocultamiento

El virus de Jerusalem está tan bien diseñado que cuenta con numerosas protecciones para evitar ser borrado o sobrescrito en la memoria. Utiliza las protecciones que proporciona el sistema operativo DOS para protegerse en su stack, y crea áreas de memoria de trabajo que evitan que sea "tocado" por otros datos.

Jerusalem infecta los archivos con extensión .COM, introduciéndose en su código, al principio del programa, siempre y cuando la suma (longitud del archivo) sea menor o igual a 64 Kbytes, y lo hace una sola vez.

A los archivos con extensión .EXE los puede infectar tantas veces como sea las que se ejecuta, hasta que el disco se llene. En este caso se posiciona al final del código del programa por medio de un APPEND y modifica el punto de entrada (start point) del programa.

Mecanismo de disparo

Este virus es uno de los más contagiosos porque infecta los programas, y no se necesita más que ejecutar el programa infectado para que se instale en la memoria de la computadora. Una vez en la memoria, infectará todos los programas que se ejecuten en la misma sesión de trabajo.

El virus de Jerusalem utiliza la interrupción 1AH, con la función 04H para recuperar la fecha actual desde el BIOS. Cuando se cumple que la fecha del sistema coincida con algún

viernes 13, se activa una parte del virus que va borrando cualquier programa o archivo que se ejecute, incluso los de extensión .OVL, .OVR, etc.

Forma de detectarlo

Existen muchos programas antivirus para detectar y erradicar este virus que lo reconocen en sus diferentes versiones y lo eliminan de los programas infectados, pero es conveniente no volver a utilizar esos programas porque pueden presentar fallas al ejecutarlos.

Cuando está en la memoria de la computadora, se activa una bomba de tiempo que realiza un corrimiento de una parte del texto hacia abajo, lo que produce un efecto visual en ala pantalla, como si se abriera una pequeña ventana.

Causa errores de operación en la computadora y hace lentos los procesos, y en el momento de estar trabajando con algún programa infectado puede borrar información de la memoria o "congelar" el sistema.

Conclusiones

El virus de Jerusalem es un virus dañino y peligroso porque destruye los programas que se ejecuten en la fecha programada para su activación, que es cualquier viernes 13 (después de 1987).

El código, aunque no tan "elegante" como el del virus de Paquistán, ha permitido a otros programadores realizar cambios, cancelar o incluir rutinas como la que controla la cantidad de infecciones a un mismo programa.

PC MAGAZINE (Edición en Inglés)
Vol. 12, No. 6
Marzo 30, 1993

VIRUS INFORMATICOS
Juan José Nombela
Paraninfo
Segunda edición, 1991

Virus en las Computadoras
Gonzalo Ferreira
Macrobit
Segunda Edición 1991

ULTRIX System Manager II: Security and Performance
Servicios Educacionales
Digital Equipment

CERT ADVISORY (Computer Emergency Response Team)
"WANK" worm on SPAN Network
Octubre 17, 1989

Elaboración de programas antivirus 10

Como se mencionó anteriormente, el uso de defensas de software no garantizan, por sí mismas, la erradicación de los virus. Es importante hacer notar que el empleo de programas de defensa representan una solución temporal y sólo en la medida que se usen como elementos de una estrategia corporativa, serán realmente efectivos.

La elaboración de programas antivirus es una tarea interesante, que requiere horas de estudio e investigación, lo que puede representar un reto al programador de aplicaciones. Sin embargo, y antes de continuar, vale la pena hacerse la siguiente pregunta:

¿Cuál es el propósito del desarrollo de un programa antivirus?

Cuando se inicia en el desarrollo de programas antivirus, se debe estar consciente del costo que esto representa y si realmente la inversión redituará.

Si el escenario en el que uno se mueve es el de la educación o investigación, el desarrollo de antivirus tiene valor por sí mismo, ya que es impresionante la cantidad de nuevos algoritmos y tecnologías de programación que han surgido de la mano con los virus. Por otro lado, si el propósito es comercial, hay que tomar en cuenta que existen en el mercado una variedad de programas antivirus capaces de detectar y erradicar "cientos" de tipos diferentes de virus, por lo que nuestro producto deberá ser excepcionalmente bueno para penetrar el mercado.

El parecer de los autores de este documento, es que no existe ningún otro propósito en donde sea redituable el desarrollo de programas antivirus.

Sin embargo, quizá existan personas (o tal vez instituciones) interesadas en el desarrollo e implantación de sus propias soluciones. Es por lo anterior que ha dedicado este capítulo al análisis, estudio y desarrollo de programas antivirus.

Metodología de desarrollo

El desarrollo de programas antivirus no es un simple trabajo de programación, requiere horas de estudio y análisis, mantener registros o bitácoras de los síntomas, efectos, archivos o áreas del sistema contagiados; así como el uso y/o desarrollo de herramientas que permitan revisar el contenido de archivos infectados, buscar cadenas de caracteres, revisar la memoria, restaurar las áreas del sistema, etc.

Se puede dividir el proceso de elaboración de un programa antivirus en dos fases: fase de estudio y fase de desarrollo.

Fase de estudio

Durante esta fase, se realiza la búsqueda del o los virus a atacar, su detección y aislamiento, y finalmente el análisis del funcionamiento (tamaño, forma de contagio, cargado en memoria, efectos, etc.). Esta fase consta de las siguientes etapas: detección, reproducción e identificación, aislamiento e interpretación.

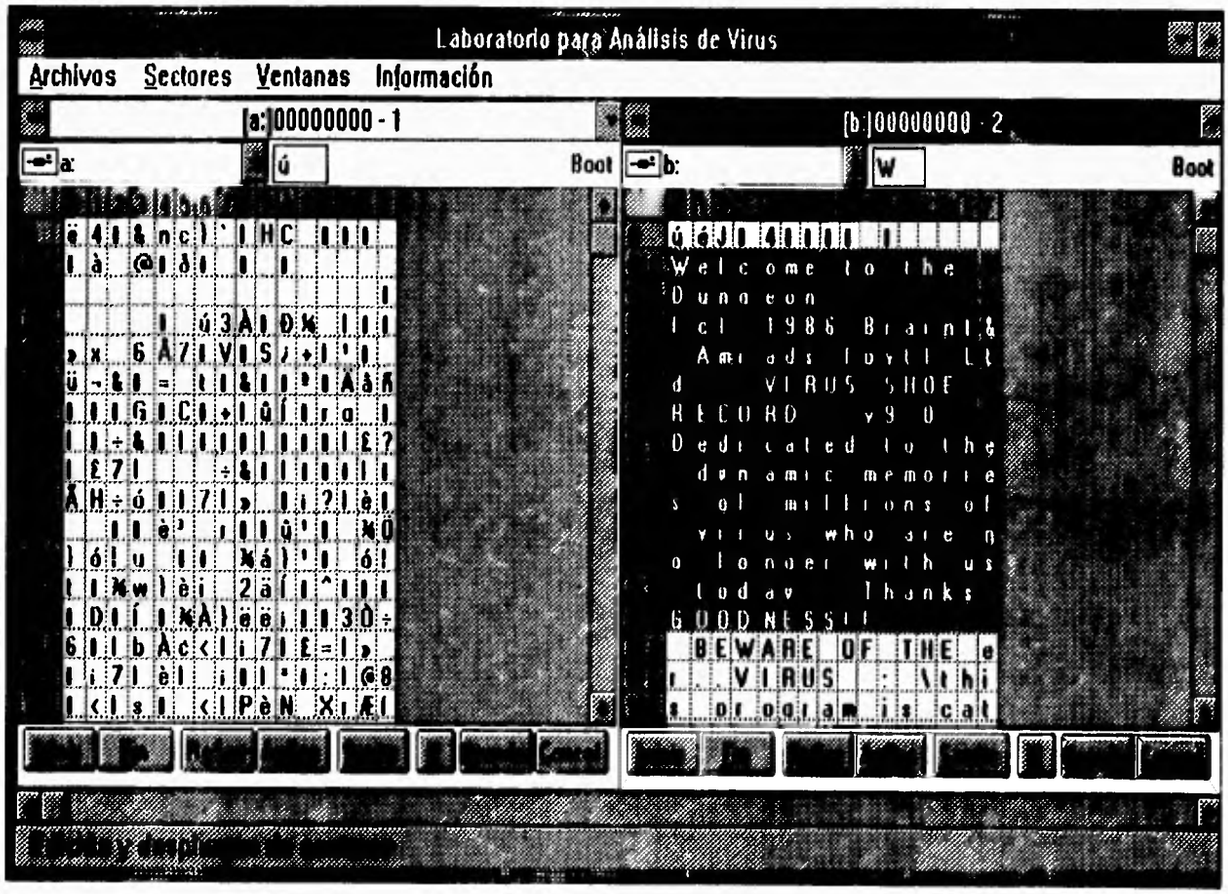
Detección

Dependiendo del tipo de virus, existen diferentes metodologías para detectar o presumir su presencia:

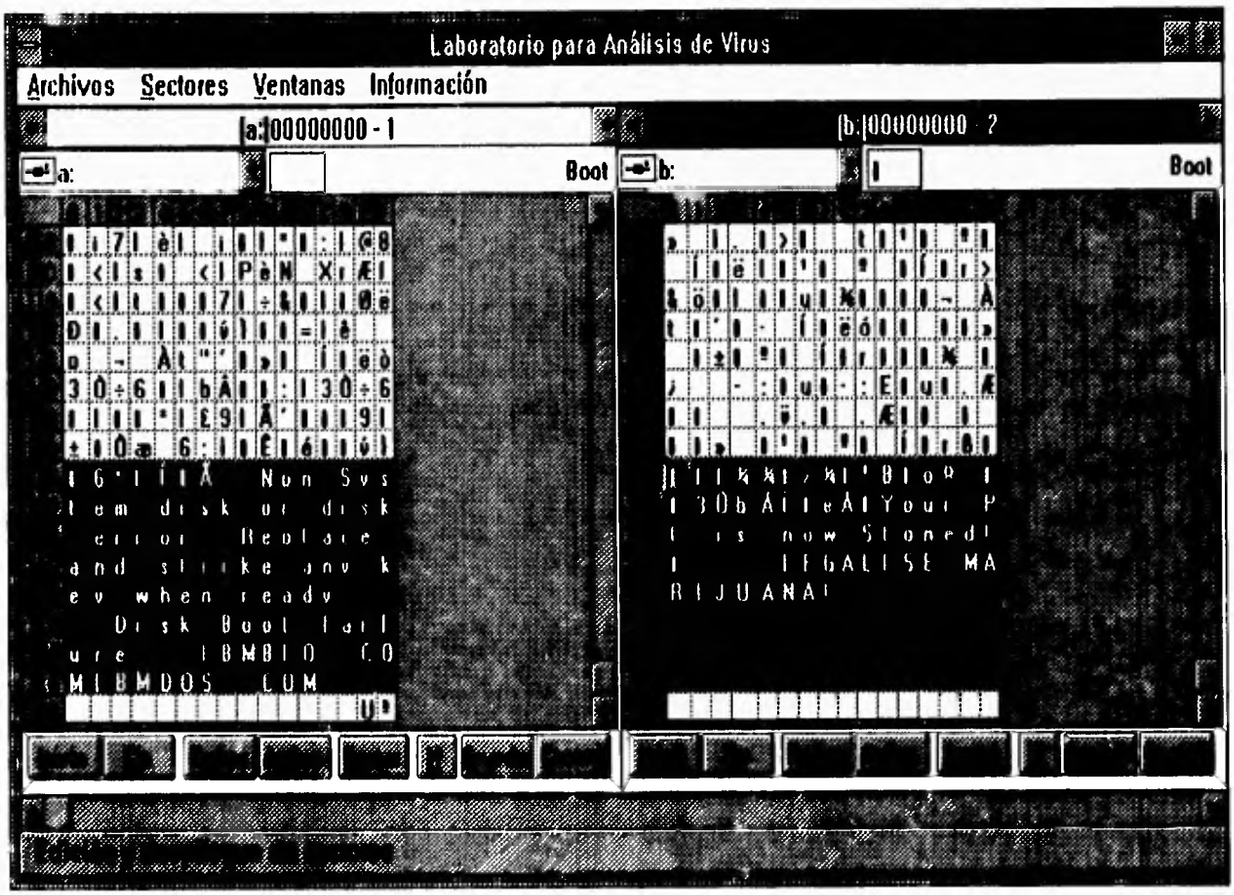
Rastreo de áreas del sistema

Identificar estereotipos de éstas áreas (obtenidas de libros o equipos que se asegura no están infectados), compararlas con las obtenidas de equipos supuestamente contaminados y buscar algún patrón en común.

Existen muchas herramientas en el mercado que permiten hacer un rastreo de las áreas del sistema, por lo que esta tarea no necesariamente requiere de la realización de programas para este fin. De todas maneras, se ha incluido como parte de este documento un diskette con programas de rastreo y utilerías.



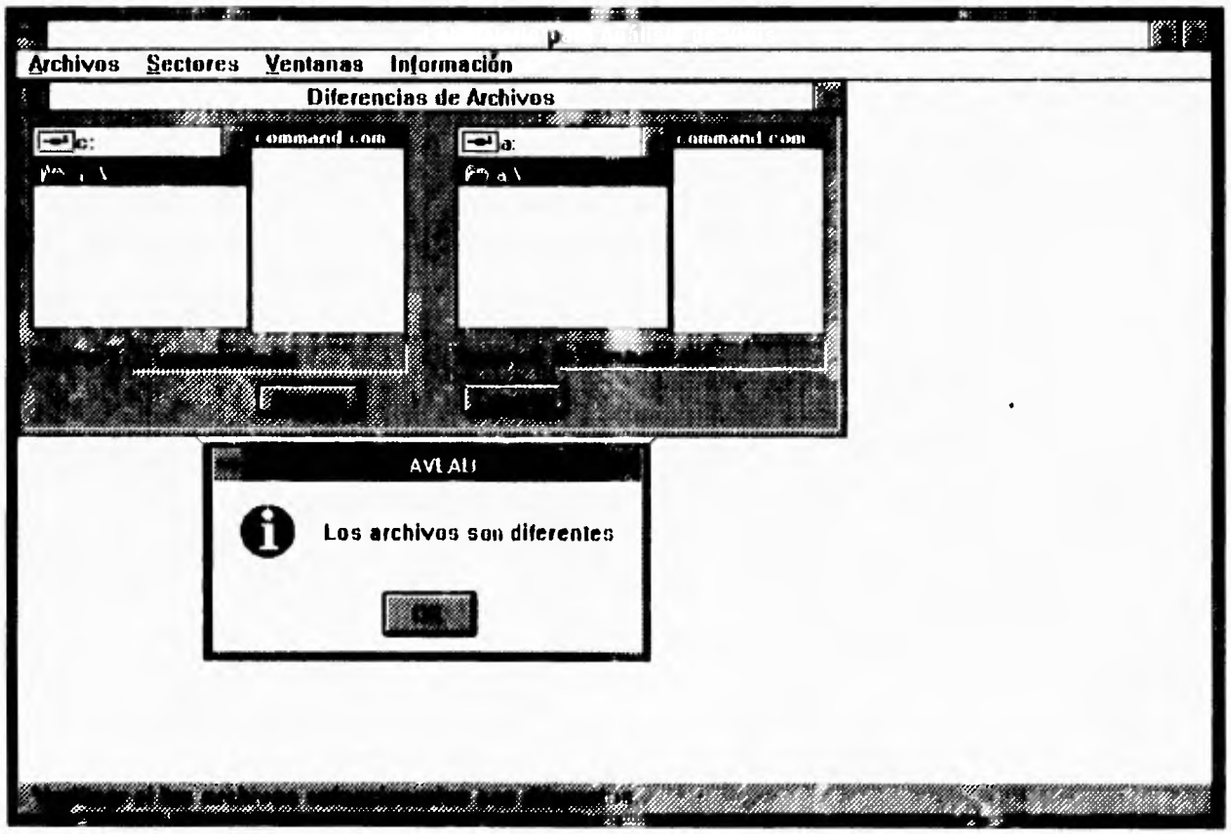
Comparación de áreas de BOOT (una contagiada por el virus de Paquistán)



Comparación de áreas de BOOT (una contagiada por el virus Stoned)

Búsqueda de diferencias en archivos

Comparar archivos que se presume están "infectados" con versiones originales de los mismos (tamaño, tiempo que tarda en ejecutarse, diferencias en código, etc.)



Diferencias en archivos (uno de ellos contagiado por el virus viernes 13)

Monitoreo

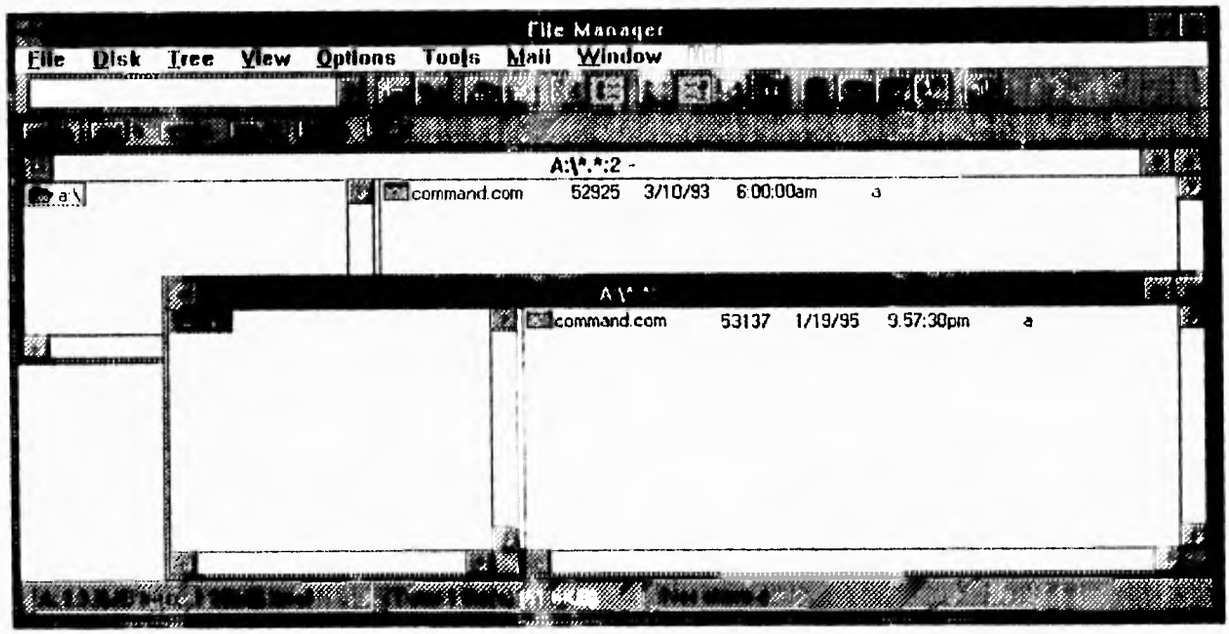
Chequeo constante de archivos y áreas del sistema que permitan identificar algún cambio sensible en su estructura o apariencia.

Se debe mantener una bitácora o registro de síntomas o efectos que permitan describir el comportamiento del virus.

Ejemplos de una bitácora de monitoreo:

Valores iniciales			
Equipo: **Número de usuarios:** **Fecha de inicio:**			
Objeto	Tiempo promedio de ejecución	Tamaño	Fecha

Bitácora de cambios y síntomas					
Equipo:					
Objeto cheksum	Fecha de registro	Tamaño	Fecha del objeto	Tiempo de ejecución	Observaciones



Diferencias de un archivo contagiado por el virus viernes 13

Reproducción e identificación

Una vez que se obtiene un patrón de comportamiento es posible tratar de identificar al virus. De los equipos que se presume están infectados, se requiere obtener "muestras" del virus (es decir provocar el contagio de un diskette u algún otro medio de almacenamiento manejable). Esto no es sencillo, pero si se tiene un buen registro se "síntomas" y probables archivos o áreas del sistema infectadas, esto facilitará el proceso. Lo recomendable es extraer varias muestras de diferentes equipos.

La reproducción de un virus es la etapa más complicada, ya que no es posible trabajar en un equipo al que tengan acceso más de una persona o se almacene información de importancia. Es necesario separar un equipo de pruebas y garantizar que inicialmente este equipo no este infectado. Esto significa que se deberán reinicializar (a bajo nivel) los dispositivos de almacenamiento que se pudieran tener, reconfigurar y reinstalar el sistema operativo, usando únicamente software original y protegido contra escritura. De esta forma no habrá duda de que se analiza el comportamiento de un virus.

La reproducción del virus se logra siguiendo el patrón de comportamiento planteado anteriormente (de una de las muestras tratar de contaminar al equipo de prueba). Si la pista es correcta, se habrá identificado totalmente al virus; pero si aún después de haber probado todas las muestras no se dan los resultados esperados, habrá que replantearse nuevamente la etapa de detección.

Aislamiento

Una vez identificado al virus, es necesario extraer el código para poderlo interpretar. La mejor manera de hacerlo, es a través de un programa que seleccione el código sospechoso y lo guarde en forma de archivo, lo que facilitará posteriormente su interpretación. Así mismo, se han incluido programas capaces de realizar esta función.

Interpretación

La única manera de asegurar que lo que se ha aislado es realmente un virus, es interpretar su código. Aquí, el recurso que se tiene es el uso de desensambladores, en los que hay que tratar de obtener:

- La forma en que se posiciona en memoria
- Qué partes del sistema operativo modifica
- Disparadores del virus
- El procedimiento de infección
- Si tiene alguna forma de identificar archivos infectados por él mismo.

Existe la posibilidad de que el código que se este interpretando sea una evolución de un virus mutante (también llamados genéticos). En este caso, hay una consideración adicional para la fase de desarrollo:

- Habrá que estar consciente de que probablemente sea muy difícil obtener evoluciones anteriores del mismo virus, por lo que de entrada el programa antivirus estará limitado a detectar sólo ocurrencias de esa mutación.

Fase de desarrollo

La fase de desarrollo consiste en elaborar un programa que sea capaz de identificar al virus, erradicarlo de la computadora y, si es posible, restaurar el daño que haya hecho. Cada uno de los pasos de hecho, son deducidos por puro sentido común. Un programa antivirus deberá realizar las siguientes operaciones: identificar al virus, erradicar al virus de memoria y erradicar al virus de los medios de almacenamiento.

Identificar al virus

Un vez estudiado el código del virus es posible establecer qué secuencia de bytes identifican al virus corriendo en memoria. La selección de esta secuencia debe hacerse muy cuidadosamente, ya que una mala decisión puede provocar que el antivirus arroje falsos positivos (es decir, que detecte virus donde no hay). Por otro lado, una buena selección, puede hacer a una secuencia de bytes ideal para detectar a más de un virus de la misma clase (por ejemplo, cierta combinación de bytes en el virus de Turín permiten identificar también al virus de Pakistán).

Erradicar al virus de memoria

Esto es lo más importante a realizar, ya que si el virus se encuentra corriendo en la memoria de la computadora, ninguno de los siguientes pasos servirá. La opciones son múltiples de acuerdo con el tipo de virus:

- Restaurar llamadas al sistema que han sido intervenidas por el virus
- Eliminar procesos residentes
- Eliminar las condiciones que disparan su ejecución (único recurso cuando eliminar al virus representaría la caída del sistema)

Erradicar al virus de los medios de almacenamiento secundarios

Cuando se garantiza que el virus ya no corre en memoria, entonces se puede rastrear al virus para erradicarlo de otros medios de almacenamiento. Dependiendo del tipo de virus, los mecanismos varían:

Si el virus ha modificado áreas del sistema

En este caso, la mayoría de los virus tiene copias de las áreas originales (ya que ellos mismos las usan para dar la apariencia de que todo esta en orden), por lo que sólo hay que regresarlas a su posición original. Por lo general, esta operación es más que suficiente para

erradicar al virus; sin embargo, hay que revisar el código del virus para descartar la posibilidad de que tenga algún disparador adicional.

Si el virus ha modificado programas

Normalmente hay una secuencia que dispara al virus antes de ejecutar la aplicación (generalmente un salto a otra localidad de memoria), por lo que removiendo esa secuencia o restaurando los valores originales es posible erradicar al virus.

Restaurar las partes dañadas o modificadas por el virus

No siempre es posible restaurar las partes dañadas o modificadas por un virus; en ocasiones tratar de hacer esto puede causar más daños que los que el propio virus ocasionaría. Solamente se recomienda intentar este paso por dos razones:

- Si se tiene conocimientos firmes del sistema y una muy buena interpretación del código del virus, o

- Si resulta imperativo hacerlo, es decir, que sin la restauración no se podrá volver a trabajar con el sistema.

erradicar al virus; sin embargo, hay que revisar el código del virus para descartar la posibilidad de que tenga algún disparador adicional.

Si el virus ha modificado programas

Normalmente hay una secuencia que dispara al virus antes de ejecutar la aplicación (generalmente un salto a otra localidad de memoria), por lo que removiendo esa secuencia o restaurando los valores originales es posible erradicar al virus.

Restaurar las partes dañadas o modificadas por el virus

No siempre es posible restaurar las partes dañadas o modificadas por un virus; en ocasiones tratar de hacer esto puede causar más daños que los que el propio virus ocasionaría. Solamente se recomienda intentar este paso por dos razones:

- Si se tiene conocimientos firmes del sistema y una muy buena interpretación del código del virus, o

- Si resulta imperativo hacerlo, es decir, que sin la restauración no se podrá volver a trabajar con el sistema.

La elaboración de antivirus depende del conocimiento previo que se tenga del o los virus a erradicar. Existe mucha bibliografía que describe a los virus más conocidos y facilita el trabajo del programador de antivirus; de manera que si se tiene esta información, la fase de estudio podría simplificarse considerablemente. Cabe mencionar, sin embargo, que existe la posibilidad de que el programador se enfrente a un virus totalmente desconocido, lo que requeriría de una mayor dedicación. Es por esta última razón que en este capítulo no se proponen tiempos de duración por fase o por etapas.

Implementación

A continuación se muestran los pasos para el desarrollo de un antivirus basados en la metodología antes mencionada, se tratará un caso práctico.

Antecedentes

Durante los últimos 15 días se ha observado en las computadoras un comportamiento anormal. En diferentes horas del día, mientras los usuarios trabajan, aparece una pelotita rebotando por la pantalla de los monitores de las computadoras. Lo que ha alarmado a los usuarios es que aún en máquinas sin disco duro se presenta este fenómeno, por lo que no se tiene certidumbre sobre cuántos discos presentan el mismo problema. Sin duda esto es síntoma de la presencia de un virus.

El virus de Turín es uno de los más antiguos y conocidos que existen para computadoras personales, su aparición data del año 1988, por lo que resulta obvio que a la fecha existan muchos programas antivirus para erradicarlo. El propósito de esta sección es mostrar la funcionalidad de la metodología sin tomar en cuenta el caso particular de estudio, es decir, sin dar tanta importancia al virus en cuestión.

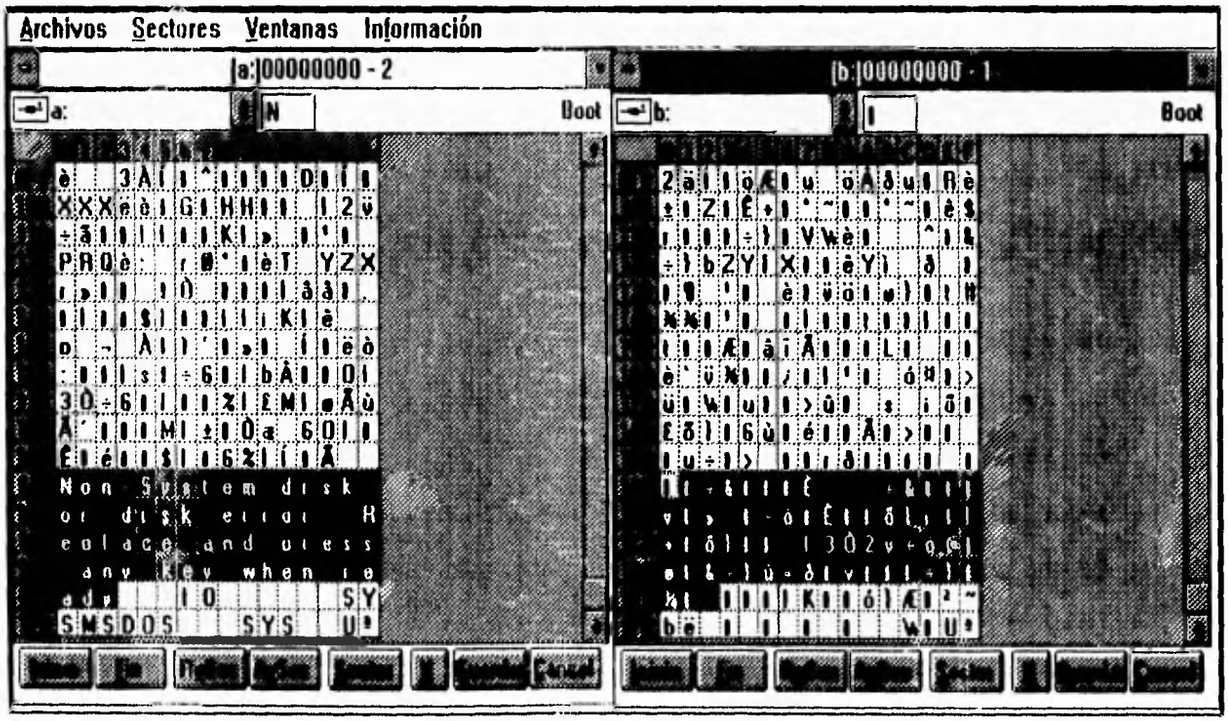
El primer paso es elaborar una hipótesis del comportamiento del supuesto virus dadas las observaciones y comentarios de los usuarios del sistema. Los usuarios están familiarizados con el comportamiento de sus equipos, así que escuchar sus impresiones puede permitir establecer una idea inicial de como funciona y cual es el procedimiento de contagio. Para este caso particular, la aparición de la pelotita es absolutamente aleatoria en cuanto a la hora del día, sin embargo, todas las apariciones coinciden con algún tipo de acceso a disco, por lo que se puede pensar que existe alguna condición que combinada por el acceso a disco, dispara la acción de la pelotita. Asimismo, aparentemente la aparición de la pelotita, de acuerdo con las revisiones constantes del sistema, no tiene nada que ver con la destrucción de información.

Fase de estudio

Detección

Durante la etapa de detección lo primero que procede es el rastreo de las áreas del sistema, ya que esto consume menos tiempo que las metodologías de búsqueda de diferencias en archivos y el monitoreo. El proceso de rastreo de las áreas del sistema requiere del conocimiento de la estructura de sistemas de archivos que corresponden al sistema operativo en estudio.

El sistema operativo DOS consta de cuatro áreas en su sistema de archivos: El área de BOOT, el área de FAT, el área de DIRECTORIOS y el área de DATOS. El área de BOOT es la más atacada por los virus debido a que es el área de arranque de DOS; contiene secuencias de instrucciones equivalentes a las de cualquier programa que permiten cargar los archivos del sistema operativo, por lo que es fácil sustituirlas por el código de un virus. Así, lo que se requiere inicialmente es comparar el área de BOOT que se presume infectada con el área de BOOT de cualquier disco que se asegura no infectado. La siguiente tabla muestra la comparación de dos áreas de BOOT, una original, después de haber sido formateada por medios originales y protegidos contra escritura; la otra, después de haber notado la aparición de la pelotita.



Es importante aclarar que el área de BOOT esta presente en todos los discos formateados en DOS, sin importar si éstos se utilizarán para inicializar el sistema operativo (es decir, a los archivos MSDOS.SYS e IO.SYS). Independientemente de este hecho, la computadora cargará la información del BOOT del disco, incluso el mensaje "Non system disk or disk error" proviene de dicho código, como se puede observar en la tabla anterior, y no del sistema operativo.

Tal como se puede observar en las tablas existen diferencias notables entre una y otra, por lo que inicialmente se puede pensar en la existencia de un virus. Si al verificar no se encontrara alguna diferencia, se tendría que recurrir al siguiente paso, es decir, el chequeo de diferencias en archivos y monitoreo del sistema.

Cuando se han encontrado evidencias de cambios estructurales en las áreas del sistema o, en el caso de archivos ejecutables, cambios en el contenido y forma del archivo, podemos asegurar que esto es consecuencia de un virus. Sin embargo lo que no se puede asegurar es que estas diferencias correspondan a los efectos del virus o al código del mismo. Vale la pena, de todas formas, separar todos medios de almacenamiento con estos cambios, para decodificarlos posteriormente y llegar a una conclusión.

Este procedimiento es muy efectivo para virus con patrones de conducta constantes. No hay que olvidar que en el caso de virus mutantes o genéticos, no hay patrones consistentes de comportamiento. Para estos casos se debe considerar que probablemente el código que se separe sea sólo una cadena encriptada en un proceso de mutación y aún cuando se pueda decodificar, este será un patrón único cuya reproducción y efectos serán diferentes en cualquier nuevo contagio.

Reproducción e Identificación

Primero identificamos aquellos discos que presentan diferencias con respecto de nuestros patrones originales, a estos llamaremos discos infectados. Se probaran en dos diferentes computadoras; con una se ejecutará el BOOT con un disco que contiene al sistema operativo, con la otra, se ejecutará el BOOT pero de un disco sin sistema.

Lo siguiente es tratar de reproducir al virus. Si la hipótesis es correcta, cuando se inserte un disco no infectado (disco de prueba), deberá existir algún procedimiento que le infecte.

Volviendo a nuestro caso de estudio, al ejecutar el procedimiento de BOOT en una máquina que se asume infectada y trabajar con el disco prueba, se advierte que las áreas de este disco han sido cambiadas. Esto nos lleva a concluir que se ha identificado la reproducción del virus.

Se observa que en ambas computadoras se presenta el mismo efecto; esto quiere decir que cualquier disco formateado en DOS y contaminado con este virus, podrá contaminar a otros con el simple hecho de tratar de ejecutar el sistema con él; pero en una computadora que ya ha sido inicializada con un sistema operativo no contaminado, el disco contaminado no tendrá ningún efecto.

Otro aspecto que vale la pena mencionar, es que el virus se aloja en áreas de memoria que no son borradas al ejecutar una reinicialización de la computadora desde software (usando por ejemplo en DOS la secuencia de teclas CTRL-ALT-DEL), por lo que para garantizar que el virus ha sido borrado de memoria, será necesario apagar completamente la computadora.

Aislamiento

El proceso de aislamiento consiste en separar todo el código que ha sido modificado y depositarlo en archivos separados que puedan ser leídos por cualquier programa de desensamblado. Para nuestro caso de estudio, se obtuvieron dos archivos, un archivo directamente del área de BOOT y el otro, producto del hallazgo del área de BOOT original. Otro aspecto que se observó fue que el área de disco donde se encontró el BOOT original estaba marcada como dañada, de manera que el sistema operativo no la fuera a utilizar para guardar información. En la actualidad, otros virus en lugar de usar área de datos marcadas como dañadas, usan áreas del sistema tales como el área de FAT o el área de directorios; algunos otros utilizan el último sector de datos de disco sin marcarlo como dañado, ya que de esta forma es más difícil localizarlos.

Interpretación

Funcionamiento

El virus de la pelotita es un segmento de código que afecta los sectores del disco y su operación es diferente a la de los virus que modifican archivos ejecutables. A continuación se describe de manera general su funcionamiento.

Otro aspecto que vale la pena mencionar, es que el virus se aloja en áreas de memoria que no son borradas al ejecutar una reinicialización de la computadora desde software (usando por ejemplo en DOS la secuencia de teclas CTRL-ALT-DEL), por lo que para garantizar que el virus ha sido borrado de memoria, será necesario apagar completamente la computadora.

Aislamiento

El proceso de aislamiento consiste en separar todo el código que ha sido modificado y depositarlo en archivos separados que puedan ser leídos por cualquier programa de desensamblado. Para nuestro caso de estudio, se obtuvieron dos archivos, un archivo directamente del área de BOOT y el otro, producto del hallazgo del área de BOOT original. Otro aspecto que se observó fue que el área de disco donde se encontró el BOOT original estaba marcada como dañada, de manera que el sistema operativo no la fuera a utilizar para guardar información. En la actualidad, otros virus en lugar de usar área de datos marcadas como dañadas, usan áreas del sistema tales como el área de FAT o el área de directorios; algunos otros utilizan el último sector de datos de disco sin marcarlo como dañado, ya que de esta forma es más difícil localizarlos.

Interpretación

Funcionamiento

El virus de la pelotita es un segmento de código que afecta los sectores del disco y su operación es diferente a la de los virus que modifican archivos ejecutables. A continuación se describe de manera general su funcionamiento.

Una parte de éste segmento de código se graba sobre el área de BOOT; sin embargo, hacer esto implicaría que el sistema no pudiera ser cargado, porque ahí se encuentran las instrucciones para hacerlo. Para solucionar este problema, lo que hace el virus es copiar el área de BOOT en otro sector del disco y marcar a éste como dañado en el área de FAT para que no pueda ser utilizado por el sistema operativo.

Hay que hacer notar que también en el área de BOOT se encuentran los datos del tipo de FAT, versión del sistema operativo y algunos datos más, por lo que el virus deja intactos los primeros 32 bytes del área de BOOT que corresponden al BPB (BIOS Parameter Block), estructura de datos que describe las características físicas del disco y permite al dispositivo manejador del disco calcular las direcciones propias para un sector lógico dado. Contiene también información usada por DOS y varios sistemas de utilerías para calcular la dirección y el tamaño de cada área de control del disco (FAT y ROOT) y se copia a partir de ellos. La otra parte del código del virus es grabada de manera anexa a la copia del área de BOOT, y ésta, entre otras cosas, es la que produce el efecto de la pelotita activándose de acuerdo a algún tiempo específico del reloj.

Acción propagadora

Cuando se enciende la computadora y se intenta levantar el sistema con un disco contagiado, lo primero que sucede es la carga de las instrucciones del virus colocado en el área de BOOT, donde éste toma el control de la computadora (es cargado en el último Kbyte de la memoria disponible al usuario) y procede a realizar el direccionamiento del área de BOOT para

poder así cargar al sistema operativo. Por lo anterior el sistema operativo nunca se entera que el virus ha sido cargado, y cada vez que realiza una lectura o escritura al disco el virus realiza su tarea.

Cada vez que se hace una interrupción al sistema operativo, el virus toma el control del sistema, es por esto que aparenta estar trabajando paralelamente a los procesos, cuando realmente está trabajando en "Robo–de–Ciclo" al procesador.

Por todo lo anterior si una persona que no tiene discos contaminados entra a sesión sin antes apagar la computadora, sus discos serán contagiados inmediatamente al efectuar cualquier tipo de acceso a disco, siempre y cuando éstos no estén protegidos contra escritura.

Esta forma de contagio es una manera eficiente de propagación a diferencia de otros virus que interfieren el código de programas ejecutables anexándose a ellos, requiriendo un algoritmo mucho más complicado. El tamaño de este virus es de apenas 1 KByte.

A continuación se presenta la descripción de las partes más importantes del código del "virus de la pelotita". Como parte aclaratoria, es necesario agregar que, debido al método de desensamblado del código del virus, algunas partes podrían haber sido mal desensambladas y por lo tanto, no se tiene la certeza que sea totalmente correcta.

El virus deja intactos los primeros 32 bytes del área de BOOT.

```
0F25:0100JMP 011E
0F25:0102      ---------┐
0F25:0103               |
0F25:0104               |  BPB
...                     |
...            ---------┘
0F25:011E      XOR AX,AX          ;Primera instrucción ejecutable de virus
```

A continuación procede a inicializar el sistema de disco flexible (Reset Floppy Disk System). Esto es con el fin de preparar la sustitución del vector de interrupciones para ese nivel.

```
0F25:014A      XOR AH,AH
0F25:014C      INT 13
```

De manera inmediata procede a sustituir la dirección de la rutina de la interrupción 13H por la dirección del código del virus:

```
                                        ;Guarda el
0F25:0175      MOV AX,[004C]            ;Segmento y
0F25:0178      MOV BX,[004E]            ;Offset original
0F25:017C      WORD PTR [004C],7CD0     ;Nueva
0F25:0182      [004E],CS                ;Dirección
```

NOTA: El nuevo offset en el registro CS (Code Segment) es 07C0. De esta manera, cada vez que se realice una interrupción al BIOS, primero pasará a ejecutar el código del virus.

Para realizar la copia de su código, el programa se basa en la interrupción 13 (ROM BIOS, servicios para disco flexible), funciones 2 (Read Floppy Disk) y 3 (Write Disk).

```
0F25:0198       MOV AX,0301          ;Función 3 de es—
                                     ;critura y el nú—
                                     ;mero de sectores
                                     ;a transferir (1)

0F25:019B       JMP 01A0
0F25:019D       MOV AX,0201          ;Función 2 de lec-
                                     ;tura y número de
                                     ;sectores a transferir.
```

La función de lectura permite al programa virus tomar el área de BOOT original y trasladarla a otra sección (donde, por supuesto, la marcará como dañada).

Cuando se realiza una interrupción, en lugar de realizar el viaje completo al sistema operativo primero pasa por el código del virus, éste realiza su acción e inmediatamente despúes entrega el control, por lo que parece una situación normal para el usuario. Para poder activar la parte visual del virus (la pelotita), se emplea la interrupción 1AH (set/read real time clock), que permite comparar el tiempo leído para determinarlo.

```
0F25:01F0       XOR AH,AH            ;Servicio 0, lectu—
0F25:01F2       INT 1A               ;ra de reloj.
```

Existe una interrupción de hardware, conocida como "programable tick" (08H) o tiempo programable, que se genera 18.2 veces cada segundo (exactamente 1193180/65536 veces por

segundo). Esta interrupción ejecuta a la rutina de la dirección guardada en la posición 8H en la tabla del vector de interrupción, que conserva al día la información del tiempo mantenida por el BIOS.

De esta forma, si se sustituye la dirección por la del virus, entonces parecerá que la pelotita se mueve paralelamente al proceso que se esté ejecutando en ese momento.

```
0F25:03C4       MOV AX,[0020]              ;Guarda dirección
0F25:03C7       MOV BX,[0022]              ;original.
0F25:03CB       MOV WORD PTR [0020],7EDF
0F25:03D1       MOV [0022],CS              ;Segmento y offset del virus.
```

Hay que hacer notar que esta rutina no reemplaza el manejador del "clock tick", sólo es llamada después que la rutina del virus regresa el control.

La parte visual del virus es un código que inicia leyendo el modo del video para realizar su deplegado, lo compara con el modo de video guardado en la dirección 7FD4 y si coincide procede a realizar su juego. En caso contrario, guarda el nuevo modo en la misma dirección.

```
0F25:03E6       MOV AH,0F
0F25:03E8       INT 10
0F25:03EA       MOV BL,BL
0F25:03EC       CMP BX,[7FD4]
0F25:03F0       JZ  0427
0F25:03F2       MOV [7FD4],BX
```

En esta parte del código, se inicializa con el nuevo modo de video a renglón = 1 y columna = 1

```
0F25:0427    MOV AH,03              ;Obtiene la posi—
0F25:0429    INT 10                 ;ción del cursor
0F25:042B    PUSH DX
0F25:042C    MOV AH,02              ;Nueva posición del
0F25:042E    MOV DX,[7FCF]          ;cursor, leída de
0F25:0432    INT 10                 ;la dirección 7FCF
```

Ahora se ocupa de jugar con los contadores de renglón y columna y entregar el control a la parte de propagación del virus

```
0F25:04A0    MOV AH,02              ;Nueva posición del
0F25:04A2    INT 10                 ;cursor (REN,COL).
0F25:04A4    MOV AH,08              ;Lee carácter y su
0F25:04A6    INT 10                 ;atributo, en el
                                    ;cursor.
0F25:04A8    MOV [7FCD],AX          ;Se guarda carácter
                                    ;en la dirección
                                    ;7FCD.
0F25:04AB    MOV BL,AH              ;se guarda el atri—
                                    ;buto en BL
0F25:04AD    CMP  BYTE PTR [7FD3],01    ;si el contenido es igual a 01
                                                ;cambiaelatributo.
0F25:04B2    JNZ  04B6
0F25:04B3    MOV BL,83              ;Atributo
0F25:04B6    MOV CX,0001            ;Escribirá un carác—
                                    ;ter en la pantalla
0F25:04B9    MOV AX,0907            ;Escribe carácter y
0F25:04BC    INT 10                 ;atributo (ASCII 7)
0F25:04BE    POP DX                 ;Lee nuevamente ca—
                                    ;rácter y atributo ;en el ;cursor
0F25:04BF    MOV AH,02              ;coloca el cursor
0F25:04C1    INT 10                 ;en la nueva posición
```

Fase de desarrollo

Usando herramientas convencionales de programación se pueden elaborar programas antivirus con base a la información obtenida de la interpretación del código del virus. El trabajo difícil quedo atrás y lo que resta es solamente labor de programación. A continuación se muestra el código de un programa, hecho en lenguaje de programación C, para detectar y erradicar al virus de la pelotita.

```
#include <dos.h>
int erradica_memoria();
int erradica_disco(int);

void main(int argc, char *argv[])
{
        int drive;

        if (argc < 1)
                drive = 'C';
        else
                drive = toupper(*argv[1]) - 'A';
        if (!erradica_memoria()) {
                ptintf("Error, No se puede localizar al virus de memoria\n");
                printf("favor de reinicializar con un disco de sistem original\n");
                exit(1);
        }
        if (!erradica_disco(drive)){
                printf("Eror, no se puede desinfectar al disco\n");
                exit(1);
        }
        exit(0);
}

#define BLOQUE 0x0C
int memoriaReal;
int memoriaReportada;
int IDmaquina;
```

```
void obtiene_memoria(void)
{
        int RAM;
        int bloques;
        int equipo

        equipo = biosequip();
        memoriaReportada = biosmemory();
        Idmaquina = peek(0xF000,0xFFFE)&0x00FF;
        switch ((equipo & BLOQUE) >> 2 ){
                case 0x00:
                        RAM = 16; /* Mbytes */
                        break;
                case 0x01:
                        RAM = 32;
                        break;
                case 0x02:
                        RAM = 48;
                        break;
                case 0x03:
                        RAM = 64;
                        break;
        }
        if (Idmaquina == 0xFC ) RAM = 64;
        bloques = memoriaReportada / RAM + ((memoriaReportada%RAM)?1:0);
        memoriaReal = bloques * RAM;
}
```

```
void localiza(int cantidad,int segmentoVirus)
{
        for (; cantidad; cantidad -= 2 ){
                if (peek(0x0000,0x0022) == segmentoVirus ){
                        poke(0x0000,0x0020,peek(segmentoVirus,0x7FC9));
                        poke(0x0000,0x0020,peek(segmentoVirus,0x7FCB));
                }
                segmentoVirus -= 0x0080;
        }
}

int Virus[] = {0xC033,0xD08E,0x00BC,0x8E7C,0x0000};

int erradica_memoria()
{
        int resultado;

        resultado = rastrea_memoria();
        switch (resultado) {
                case 0x01:
                        printf("Virus desactivado de Memoria\n");
                        return(1);
                        break;
                case 0x02:
                        return(0);
                case 0x03:
                        printf("El virus no se encuentra en Memoria\n");
                        return(1);
        }
        return(0);
}
```

```
int rastera_memoria()
{
        unsigned int segmentoVirus;
        int dimensionVirus;

        dimensionVirus = memoriaReal - memoriaReportada;
        while (memoriaReal - memoriaReportada ){
                segmentoVirus =  (unsigned) memoriaReportada * 64;
                segmentoVirus -= 0x7C0;
                if (encuentra(Virus,segmentoVirus,0x7C1E)){
                        memoriaReportada += 2;
                        printf("Virus Localizadon en memoria\n");
                        if (peek(0x0000,0x0022) != segmentoVirus){
                                poke(segmentoVirus,0x7FC9,peek(0x0000,0x0020));
                                poke(segmentoVirus,0x7FCB,peek(0x0000,0x0022));
                                poke(0x0000,0x0020,0x7EDF);
                                poke(0x0000,0x0020,segmentoVirus);
                                sleep(5);
                        }
                        poke(0x0000,0x0413,memoriaReportada);
                        poke(0x0000,0x004C,peek(segmentoVirus,0x7D2A));
                        poke(0x0000,0x004E,peek(segmentoVirus,0x7D2C));
                        if (!(memoriaReal - MemoriaReportada)){
                                revisa(dimensionVirus,segmentoVirus);
                                return(1);
                        }
                }else {
                        revisa(dimensionVirus,segmentoVirus);
                        reurn(2);
        }
        return(3);
}
```

```
int encuentra(int *virus,unsigned int segmento,unsigned int offset)
{
        int *aptrVirus;
        int far *aptrMem;

        aptrMem = (int far *)MK_FP(segmento,offset);
        for (aptrVirus = Virus; *aptrVirus; aptrVirus++,aptrMem++)
                if (*aptrVirus != *aprtMem)
                        return(0);
        return(1);
}

int erradica_disco(int drive)
{
        char Buffer[512];
        unsigned BOOT;
        int error;

        error = absread(drive,1,0,Buffer);
        if (error) return(0);
        while (encuentra(Buffer,FP_SEG((char far *)(BOOT+0x1E)),
                  FP_OFF((char far *)(BOOT+0x1E)))) {
                printf("El disco esta infectado\n");
                BOOT = Buffer[506]*0x100+Buffer[505]+1;
                error = absread(drive,1,BOOT,Buffer);
                if (error)
                        return(0);
                return(1);
        }
}
```

Juan José Nombela
"Virus Informáticos"
Paraninfo
Segunda edición, 1991

Charles Preston
"Computer Virus Protection"
Digital Equipment Corporation

Conclusiones *11*

El tema de los virus informáticos tiene gran importancia actualmente. Estos programas han cambiado la forma de ver a los sistemas; no se trata simplemente de una curiosidad más, sino de un profundo cambio de la manera de establecer los sistemas de seguridad; el tema de virus informáticos es una realidad que conocen y algunas veces previenen desde el usuario final menos experto, hasta el más experimentado en seguridad.

Existe una pregunta obligada con respecto a este fenómeno: ¿qué se puede hacer respecto al problema?

No es posible prohibir que se escriban virus, los muchos métodos y programas de protección no son confiables, ya que a pesar de todo siguen apareciendo nuevos virus. Se estima que cada día aparecen dos o tres virus nuevos, aunque algunos son variantes de los ya conocidos. La tecnología en materia de software se ha desarrollado de manera importante en los últimos años, lo que nos lleva a concluir que el problema tenderá a empeorar.

Aún no se ha desarrollado ninguna herramienta que pueda detener por completo el ataque de un virus y, las que existen actualmente no son eficientes, lo serían en función de que no se desarrollaran nuevos virus.

La herramientas de programación cada vez son más sofisticadas. Actualmente no se necesita conocer ningún lenguaje de bajo nivel (ensambladores), para hacer un virus; es más, ahora sólo basta con recompilar y se puede tener el mismo virus para diferentes plataformas.

Por otro lado, la migración de sistemas operativos en diferentes plataformas provoca que virus diseñados para atacar a un sistema operativo en particular sean capaces de causar daños en otros, sin necesidad de recompilar al código fuente del virus. Tal es el caso de sistemas operativos como SCO UNIX o Windows NT, corriendo en máquinas de arquitectura intel, que son dañados por un virus diseñado para MS-DOS.

Intel, una de las empresas más exitosas en venta de microprocesadores, base de la arquitectura de las computadoras personales, está migrando del microcómputo al minicómputo. Si se considera que este procesador es uno de los más estudiados, las perspectivas pueden ser preocupantes.

DOS (Disk Operating System), aunque se han desarrollado otros sistemas operativos, sigue siendo el más vendido, conocido y estudiado. Microsoft ya ha anunciado que se seguirán elaborando nuevas versiones. Es sabido también que éste sistema operativo es el que ha sufrido mayores ataques de virus.

La tendencia hacia la estandarización ha puesto a UNIX como el preferido de los sistemas abiertos, ya que éste corre en casi todas las plataformas existentes. Es lógico pensar que aún no hay programas que puedan correr en diferentes plataformas, sin necesidad de recompilar el código fuente (aunque en algunos sistemas operativos, esto se logra a través de emuladores). Los intérpretes de comandos (shells) de UNIX ejecutan archivos texto ordinarios y son casi tan poderosos como los programas compilados. Es decir, estos archivos de comandos podrán ser interpretados en cualquier arquitectura que tenga a UNIX como sistema operativo. Recordemos que el ataque más imporante que un virus ha realizado fue en una red con equipos UNIX.

La redes de computadoras permiten comunicar y transferir información entre sistemas remotos, lo que permite establecer un medio de proliferación. La tendencia es la interconexión de todos los equipos dentro de las empresas, instituciones y casas particulares. Este razonamiento lo que nos indica es que hay que tener un mayor cuidado en la conexión de nuestros sistemas, que es necesario incrementar la seguridad y establecer un plan que permita mantener la integriad de la información y los equipos.

Lo que antes podía parecer un simple tansferencia de información entre sistemas, ahora tiene que ser monitoreada y restringida. Los virus vinieron a modificar malas costumbres y a crear mayor conciencia sobre la importancia de la información a nivel personal, grupal e institucional.

Los virus informáticos son un problema importante, sin embargo, no debemos dejarnos llevar por la comercialización exagerada y tendenciosa de la forma en que lo tratan las compañías desarrolladoras de productos antivirus, que muestran cualquier publicidad con tal de vender.

Se puede comparar el reto actual de batallar contra virus computacionales con la desesperada situación que tendrían los centros para la protección y control de enfermedades si tuviesen que encontrar una cura para cada nuevo tipo de virus de la gripe. Aquí, se sugiere que las medidas protectoras de los sistemas de cómputo deben emular las capacidades extraordinarias del sistema inmune de los seres humanos, el cual destruye los organismos patógeneos por sus propios medios.

Existen tres grandes diferencias entre los virus informáticos y los demás programas dañinos: generalidad, extensión y persistencia. El ataque de un virus es general, ya que el virus avanza dentro de la computadora sin violar ningún sistema de seguridad llevando consigo el código de ataque hasta es lugar adecuado. Los virus son extensibles ya que si un virus puede infiltrarse bajo una cuenta determinada contaminará todas las cuentas que tengan acceso a la cuenta infiltrada y así sucesivamente hasta infectar todo el sistema de cómputo. Un virus es persistente porque al respaldar un programa infectado estamos dándole oportunidad al virus de quedar almacenado para la posteridad. Con todo esto, podemos asegurar que un virus escrito en 1981 podrá seguir infectando computadoras en el siglo XXI si se siguen utilizando los sistemas operativos para los cuales fue escrito.

Al igual que los virus biológicos, no podemos deshacernos de los virus informáticos, por lo que debemos aprender a convivir con ellos. Es claro que aún el programa de computadoras más sofisticado requiere la supervisión de los humanos, todavía no podemos pensar que los sistemas operativos sean capaces por si mismos de evitar daños. Esto implica establecer un esquema que aplique las medidas sugeridas en este documento para que a través de ellas podamos desempeñar nuestro trabajo con las computadoras con el menor riesgo posible.

Apéndices

Estadísticas *A*

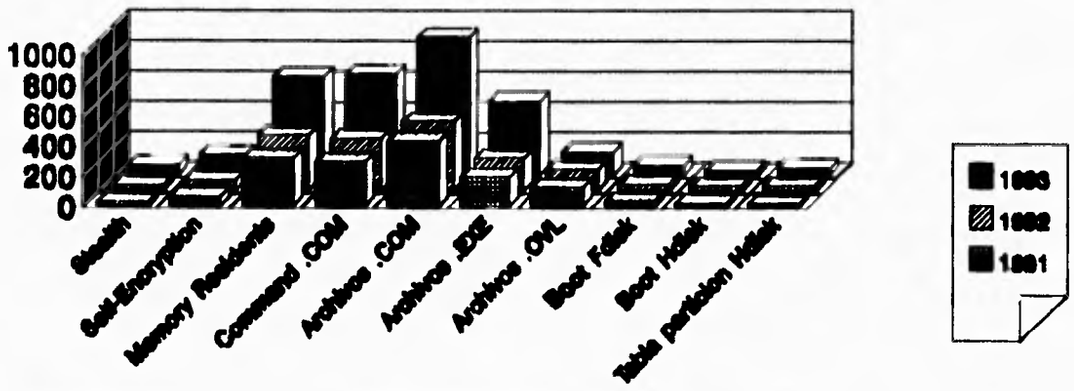
Estadísticas de infecciones de virus para DOS

Los datos en las estadísticas relacionadas con infecciones por virus informáticos son muy difíciles de obtener, por lo que su grado de confiabilidad es bajo. Por varias razones el número de compañías o agencias piensan que es de mayor importancia no revelar o poner al descubierto la vulnerabilidad de los mecanismos de seguridad en sus computadoras o ataques a sus sistemas. No existe un punto central de recolección de información en el cual se mantengan registros públicos sobre incidentes de ataques de virus. Los reportes de virus o infecciones están sujetos a rumores o histeria provocados por la falta de conocimiento del problema, además muchos ataques de virus son confundidos con problemas de software y/o hardware en los sistemas.

Las gráficas en este apéndice están relacionadas con virus para DOS, debido a que la gran mayoría de computadoras son de este tipo, por lo que la información es más fácil de obtener.

Virus de DOS

Características de los virus más comunes



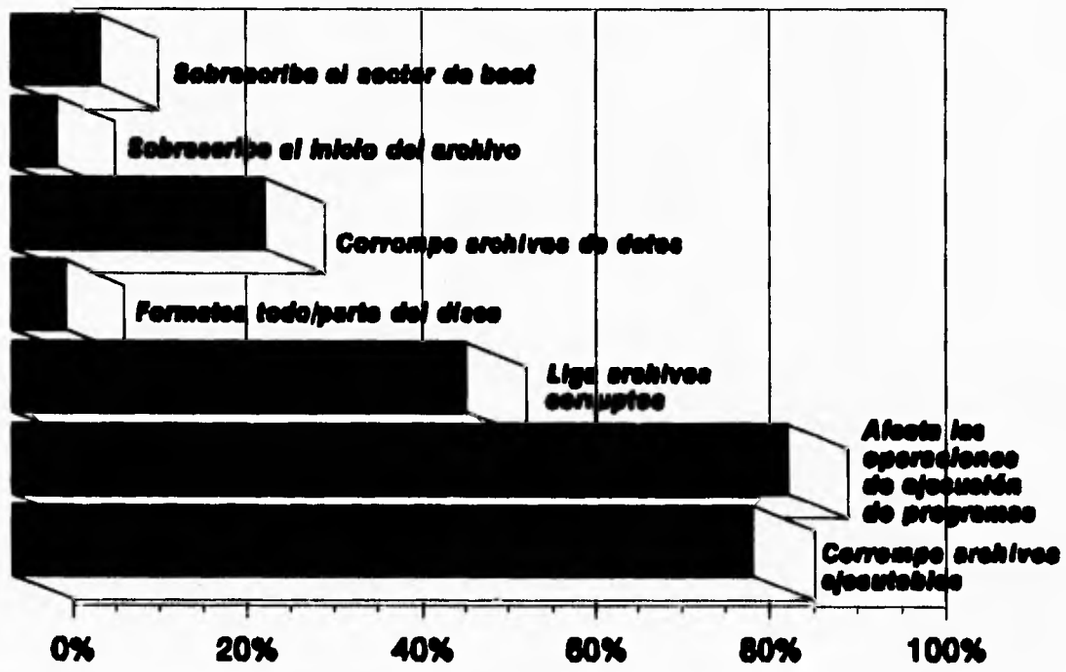
1993	63	129	641	660	600	481	160	62	31	47
1992	32	66	368	338	453	211	137	43	25	28
1991	22	59	309	284	423	185	117	32	16	12

Total de virus 1993 = 2149
Total de virus 1992 = 1263
Total de virus 1991 = 482

Fuentes: National Computer Security Association (NCSA)
Fifth Generation System, INC

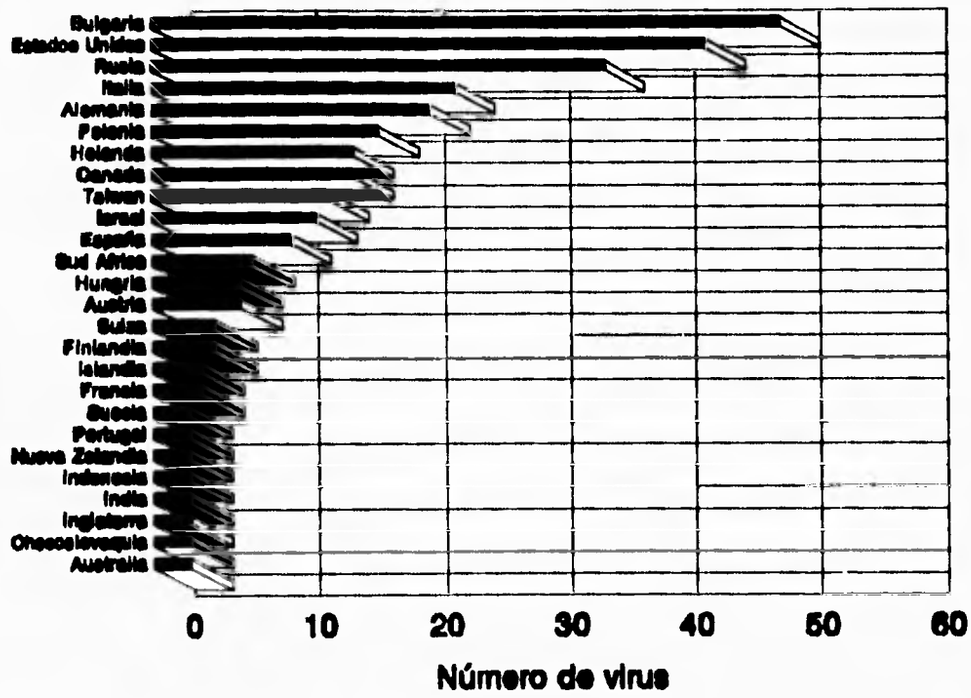
En la actualidad, se tiene registrados al rededor de 5500 diferentes variaciones de virus y para finales de 1995, según la NCSA, habrán al rededor de 38,000 variaciones de virus en el mundo.

Mayores efectos de virus para DOS



Fuente: McAfee Associates

Lugares con mayor frecuencia de virus descubiertos

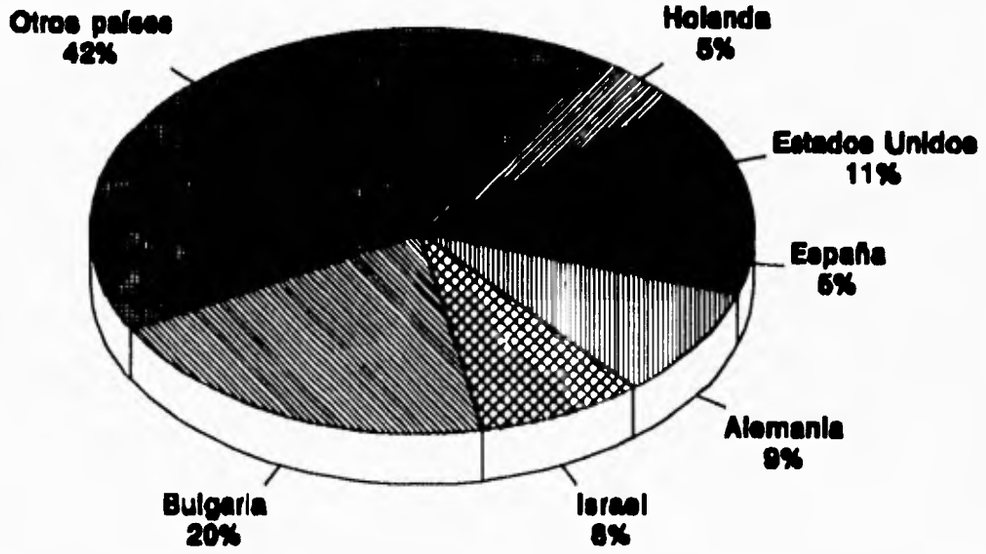


Fuente: McAfee Associates

Menos de tres virus han sido descubiertos por primera vez en cada uno de los siguientes países: Austria, Bélgica, Bolivia, Dinamarca, Grecia, Hong Kong, Irlanda, Korea, Malaysia, Malta, México, Paquistán. Los autores de los virus se mantienen anónimos en muchas de las ocasiones. En algunos casos el autor es conocido, o se tiene alguna evidencia de los lugares de origen para los virus.

La naturaleza internacional del problema su puede ver en la información de la siguiente gráfica, de acuerdo a un estudio en 1994 de la NCSA (National Computer Security Association), Bulgaria mantiene el más alto porcentaje de programación de nuevos virus, le sigue Estados Unidos de América (particularmente en el estado de California) con el 11%.

Países donde se generan la mayoría de los virus



Fuente: National Computer Security Association

AvLab: guía del usuario **B**

Laboratorio para Análisis de Virus: AvLab

El programa AvLab (Laboratorio para Análisis de Virus) es una herramienta que fue desarrollada para complementar el trabajo de investigación sobre virus informáticos, este programa NO es una vacuna ni un rastreador de virus específicos, este programa proporciona las herramientas necesarias para poder rastrear señales que nos sirven como indicadores de la posible presencia de virus en ambientes DOS. Los usuarios del AvLab serán, entonces, aquellas personas interesadas en estudiar virus informáticos, ya que con este programa contamos con herramientas que nos auxilian (si ese es nuestro objetivo) a desarrollar rastreadores y/o vacunas para virus informáticos de computadoras personales.

El programa AvLab fue desarrollado con *Visual BASIC 3.0 de Microsoft* y la librería *QuickPak Professional for Windows 3.2 de Crescent Software Inc.* Este programa corre en ambiente Windows, por lo que proporciona una interface amigable y accesible prácticamente para cualquier usuario (manejo del ratón, ventanas, menús, botones, etc.).

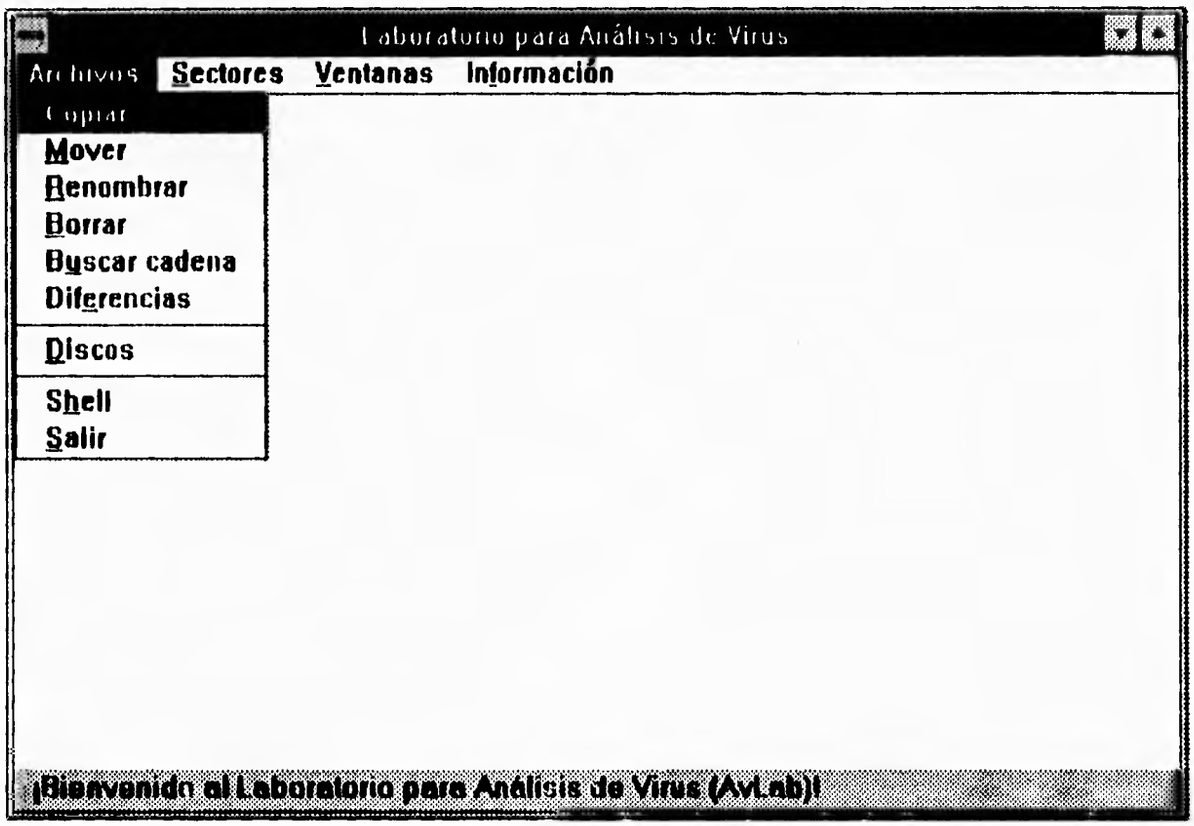
Requisitos

Los requisitos mínimos de software y hardware que necesita el AvLab para poder ejecutarse son:

* Una computadora personal con procesador Intel 80286, 80386 o 80486
* MS-DOS versión 3.1 o posterior
* Windows 3.1 o Windows for WorkGroups 3.11
* 2 Megabyte de memoria RAM (aunque se recomiendan 4)
* 2 Megabytes de disco duro
* Tener una copia en el directorio de WINDOWS\SYSTEM de los archivos de controles externos para Visual BASIC: grid..vxb, threed.vbx, cmdialog.vbx.
* Tener una copia del archivo QPRO200.DLL en el directorio WINDOWS\SYSTEM
* Tener una copia del archivo VBRUN300.DLL en el directorio WINDOWS\SYSTEM
* Ratón
* Cualquier monitor soportado por Windows 3.1.

El programa AvLab está constituido de una forma MDI (Multiple Document Interface), la cual agrupa dentro de la misma un conjunto de herramientas que son desplegadas a través de menús. A continuación se describirán las opciones que proporciona el AvLab.

En el menú **Archivos** se tienen las siguientes opciones:



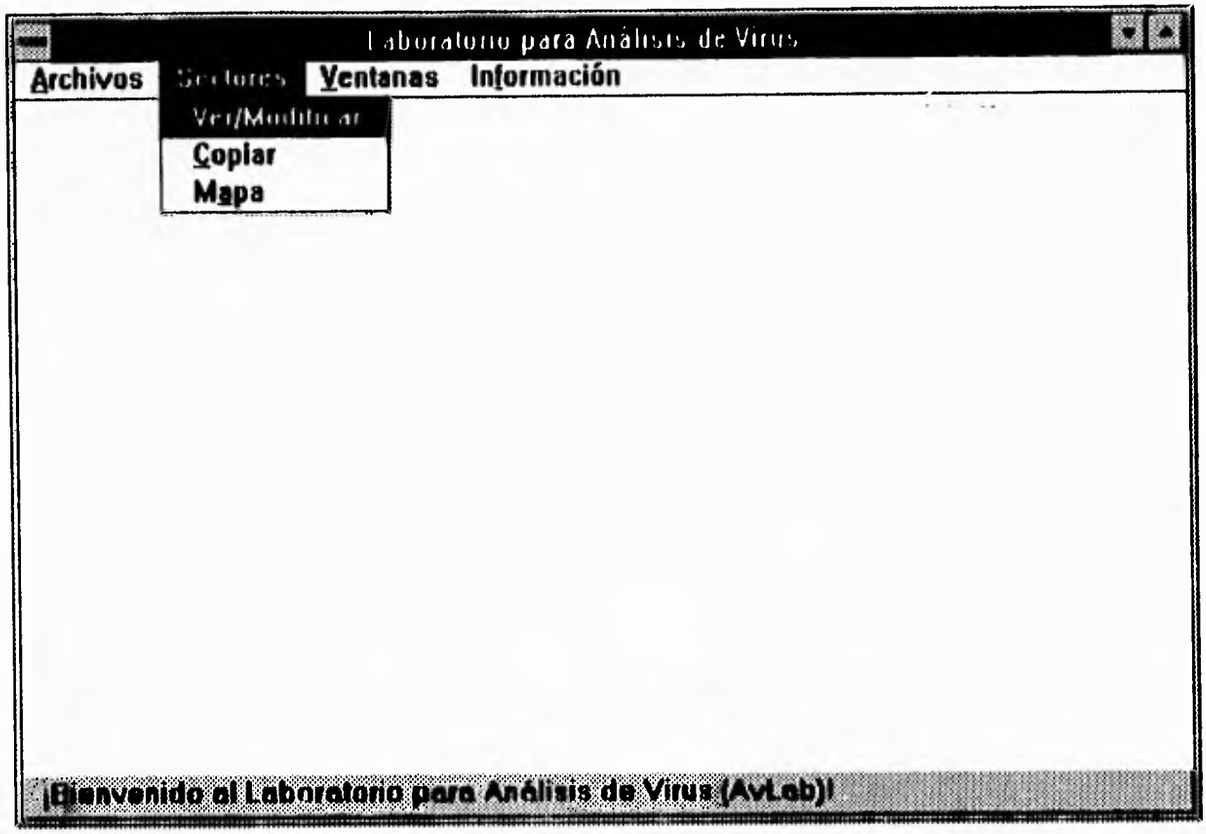
Copiar Con esta opción se puede copiar un archivo con otro nombre en el mismo subdirectorio o a otro subdirectorio o disco con el mismo u otro nombre.

Mover Esta opción es utilizada para cambiar de directorio o disco la ubicación de un archivo.

Renombrar Se utiliza para cambiarle el nombre a un archivo.

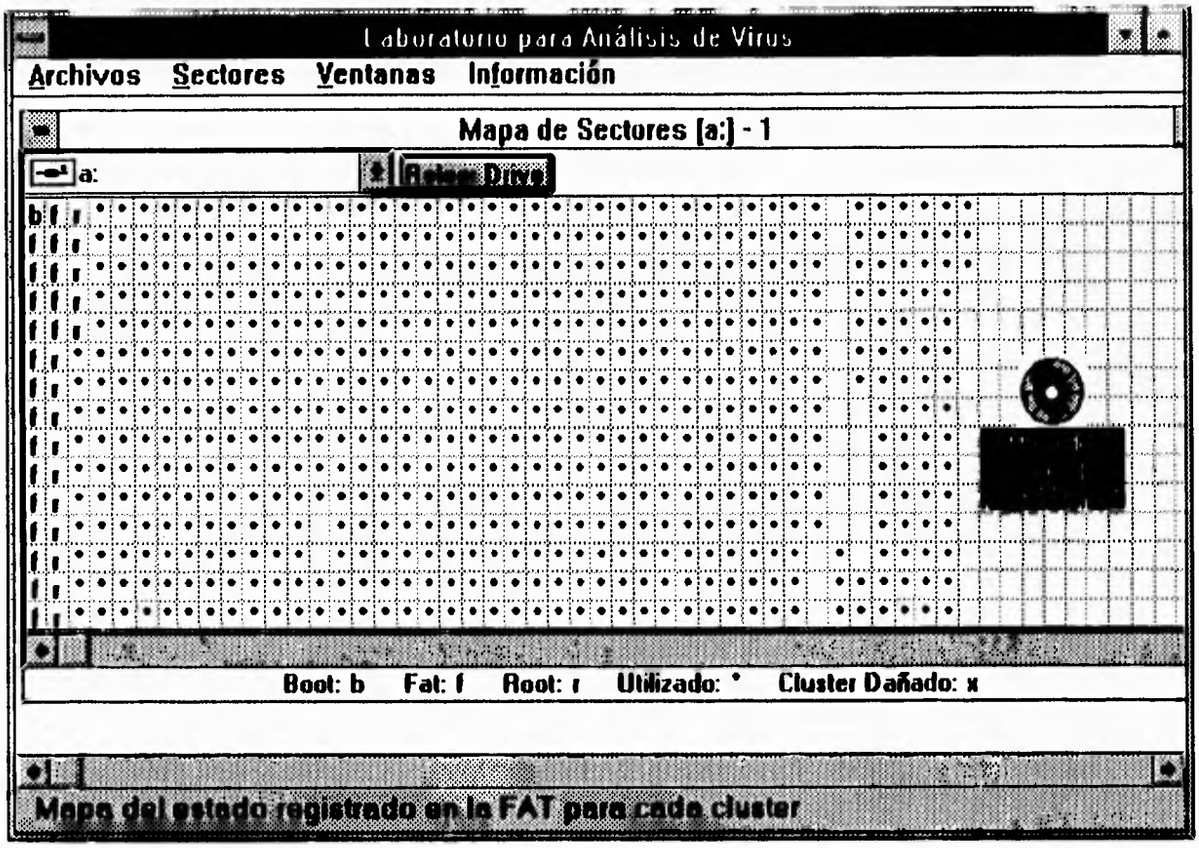
Borrar	A través de esta opción podemos borrar un archivo.
Buscar cadena	Con la ventana que se despliega al seleccionar esta opción es posible buscar en un archivo cadenas Ascii o hexadecimales.
Diferencias	Con esta opción se puede consultar si dos archivos son diferentes.
Discos	A través de la ventana que se despliega al seleccionar esta opción tenemos la posibilidad de copiar discos flexibles o darles formato.
Shell	Se utiliza para salir al ambiente DOS momentáneamente sin terminar el AvLab.
Salir	Terminar la ejecución del AvLab.

Del menú **Sectores** podemos seleccionar:



Ver/Modificar sectores Esta opción despliega múltiples ventanas con el mismo funcionamiento. Cada ventana tiene capacidad para estar desplegando y editando sectores de discos. Cuenta con facilidades para moverse en las diferentes áreas lógicas de los discos.

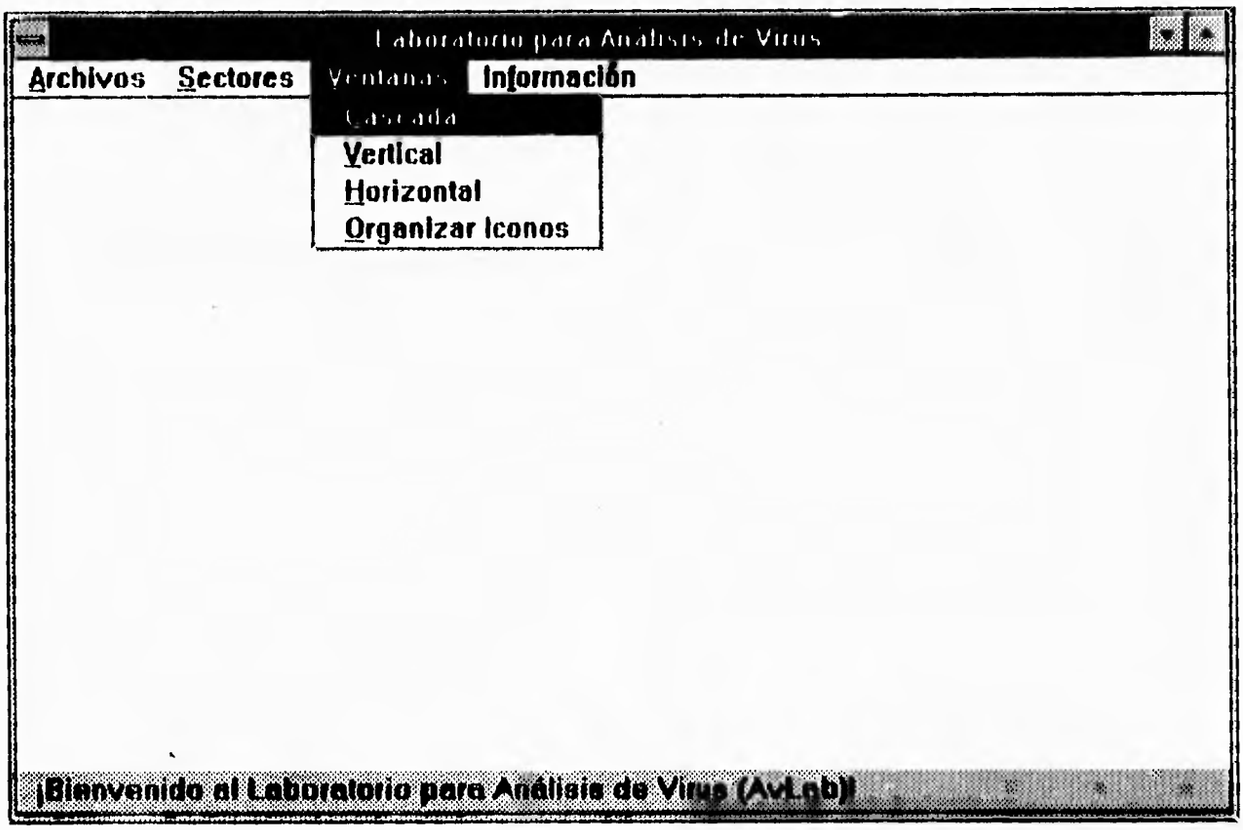
Copiar sectores Con esta opción podemos copiar el contenido de un sector a otro.



Mapa

A través de esta opción podemos ver un estado del contenido lógico de los clusters que conforman un disco. Es posible tener varias ventanas de este tipo para visualizar mapas simultáneos de diferentes discos.

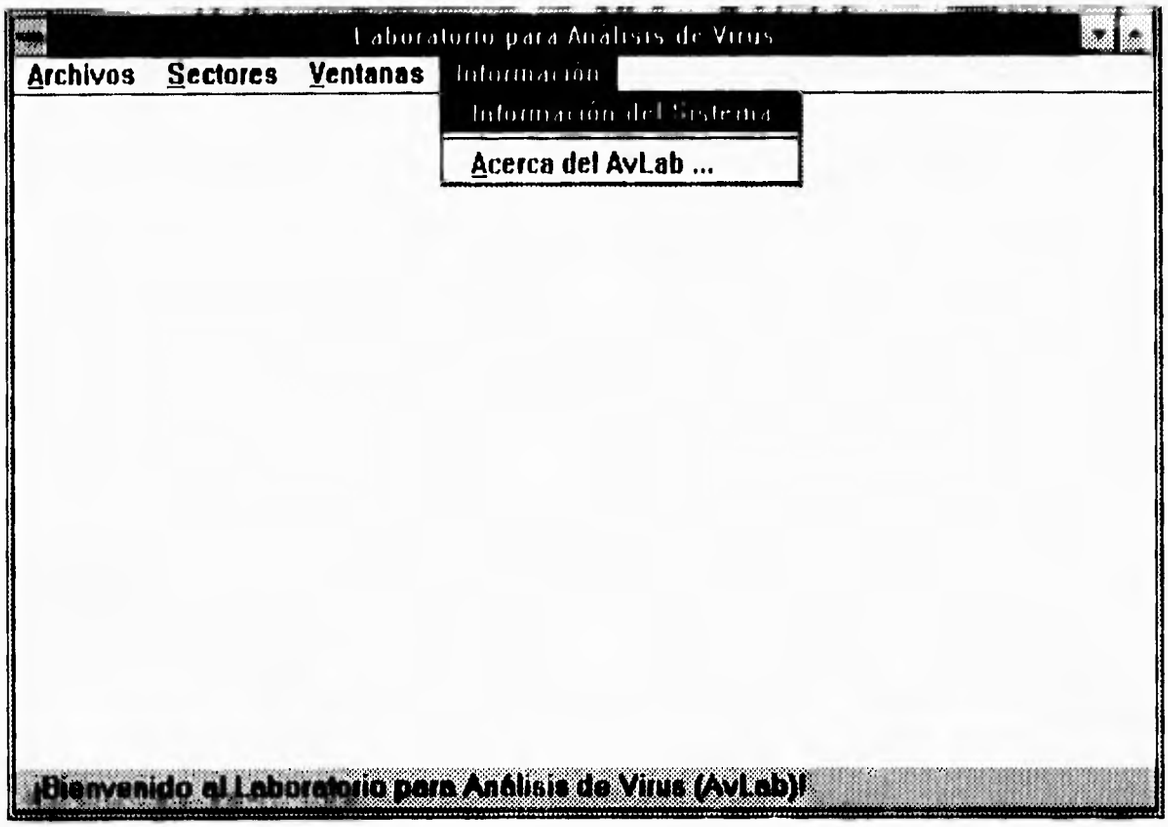
Las opciones disponibles en el menú **Ventanas** son:



Cascada	Organizar todas las ventanas en cascada.
Horizontal	Organizar todas las ventanas para que se ajusten en forma horizontal al tamaño de la ventana principal.
Vertical	Organizar todas las ventanas para que se ajusten en forma vertical al tamaño de la ventana principal.

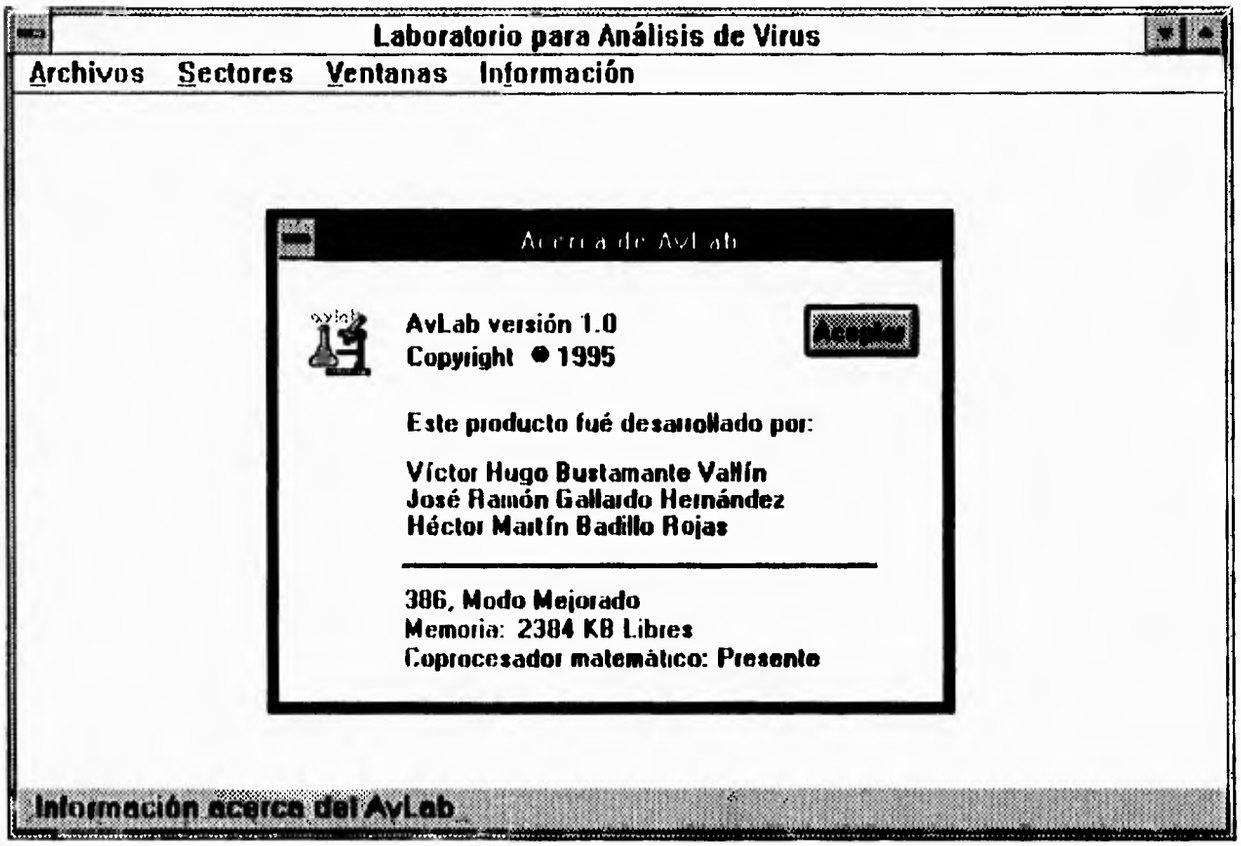
Organizar Iconos Ordenar los iconos de las ventanas minimizadas en la parte inferior de la ventana principal.

En el menú **Información** se tienen las opciones:



Información Obtener información general del estado del sistema.

Acerca del AvLab Información sobre versión y autores del AvLab.



Reflexiones sobre la confiabilidad de lo confiable C

Reflections on Trusting Trust

Ken Thompson

Introduction

I thank the ACM for this award. I can't help but feel I am receiving this honor for timing and serendipity as much as technical merit. UNIX swept into popularity with an industry-wide change from central mainframes to autonomous minis. I suspect than Daniel Bobrow would be here instead of me if he could not afford a PDP-10 and he had to "settle" for a PDP-11. Moreover, the current state of UNIX is the result of the labors of a large number of people.

There is an old adage, "Dance with the one that brought you", which means that I should talk about UNIX. I have not worked on mainstream UNIX in many years, yet I continue to get undeserved credit for the work of others. Therefor, I am not going to talk about UNIX, but I want to thank everyone who has contributed.

That brings me to Dennis Ritchie. Our collaboration has been a thing of beauty. In the ten years that we have worked together, I can recall only one case of miscoordination of work. On that occasion, I discovered that we both had written the same 20-line assembly language program. I compared the sources and was astounded to find that they matched character-for-character. The result of our work together has been far greater than the work that we each contributed.

I am a programmer. On my 1040 form, that is what I put down as my occupation. As a programmer, I write programs. I would like to present to you the cutest program I ever wrote. I will do this in three stages and to bring it together at the end.

Stage I

In college, before video games, we would amuse ourselves by posing programming exercises. On of the favorites was to write the shortest self-reproducing program. Since this is an exercise divorced from reality, the usual vehicle was FORTRAN. Actually, FORTRAN was the language of choice for the same reason that three-legged races are popular.

More precisely stated, the problem is to write a source program that, when compiled and executed, will produce as output an exact copy of its source. If you have never done this, I urge you to try it on your own. The discovery of how to do it is a revelation that far surpasses any benefit obtained by being told how to do it. The part about "shortest" was just an incentive to demostrate skill and determine a winner.

Figure 1.1 shows a self-reproducing program in the C programming language. (The purist will note that the program is not precisely a self-reproducing program, but will produce a self-reproducing program). This entry is much too large to win a prize, but it demostrates the technique and has two important properties that I need to complete my story:

```
char s[]={
        '\t',
        '0',
        '\n',
        '}',
        ';',
        '\n',
        '\n',
        '/',
        '*',
        '\n',
        (213 lines deleted)
        0
};

/*
 *The string s is a
 *representation of the body
 *of this program from '0'
 *to the end.
 */

main()
{
        int i;

        printf("char\ts[]={\n");
        for(i=0;s[i];i++)
                printf("\t%d,\n",s[i]);
        printf("%s",s);
}
```

Here are some simple transliterations to allow a non-C programmer to read this code.

=	assignment
==	equal to .EQ.
!=	not equal to .NE.
++	increment
'x'	single character constant
"xxx"	multiple character string
%d	format to convert to decimal
%s	format to convert to string
\t	tab character
\n	newline character
!=	not equal to .NE.
++	increment
'x'	single character constant
"xxx"	multiple character string
%d	format to convert to decimal
%s	format to convert to string
\t	tab character
\n	newline character

FIGURE 1.1

1. This program can be easily written by another program.

2. This program can contain an arbitrary amount of excess baggage that will be reproduced along with the main algorithm. In the example, even the comment is reproduced.

Stage II

The C compiler is written in C. What I am about to describe is one of many "chicken an egg" problems that arise when compilers are written in their own language. In this case, I will use a specific example from the C compiler.

C allows a string construct to specify an initialized character array. The individual characters in the string can be escaped to represent unprintable characters. For example,

"Hello world\n"

represents a string with the character '\n', representing the new line character.

Figure 1.2 is an idealization of the code in the C compiler that interprets the character escape sequence. This is an amazing piece of code. It "knows" in an completely portable way what character code is compiled for a new line in any character set. The act of knowing then allows it to recompile itself, thus perpetuating the knowledge.

```
...
c=next();
if (c!='\\')
        return(c);
c=next();
if (c=='\\')
        return('\\');
if (c=='n')
        return('\n');
...
```

FIGURE 1.2

Suppose we wish to alter the C compiler to include the sequence "\v" to represent the vertical tab character. The extension to Figure 1.2 is obvious and is presented in Figure 1.3. We then recompile the C compiler, but we get a diagnostic. Obviously, since the binary version of the compiler does not know about "\v" means, then our new change will be become legal C. We look up on an ASCII chart that a vertical tab is decimal 11. We alter our source to look like Figure 1.4. Now the old compiler accepts the new source. We install the resulting binary as the new official C compiler and now we can write the portable version the way we had in Figure 1.3

```
...
c=next();
if (c!='\\')
        return(c);
c=next();
if (c=='\\')
        return('\\');
if (c=='n')
        return('\n');
if (c=='v')
        return('\v');
...
```

Figure 1.3

```
...
c=next();
if (c!='\\')
        return(c);
c=next();
if (c=='\\')
        return('\\');
if (c=='n')
        return('\n');
if (c=='v')
        return(11);
...
```

Figure 1.4

Stage III

Again, in the C compiler, Figure 1.5 represents the high level control of the C compiler where the routine "compile" is called to compile the next line of source. Figure 1.6 shows a simple modification to the compiler that will deliverately miscompile source whenever a particular pattern is matched. If this were not deliberate, it would be called a compiler "bug". Since it is deliberate, it should be called a "Trojan horse".

```
compile(s)

char *s
{
        ...
}
```

Figure 1.5

```
compile(s)

char *s;
{
        if (match(s,"pattern")) {
                compile("bug");
                return;
        }
        ...
}
```

FIGURE 1.6

The actual bug I planted in the compiler would match code in the UNIX "login" command. The replacement code would miscompile the login command so that it would accept either the intended encrypted password or a particular known password. Thus if this code were installed in binary and the binary were used to compile the login command, I could log into that system as any user.

Such blatant code would not go undetected for long. Even the most casual perusal of the source of the C compiler would raise suspicions.

The final step is represented in Figure 1.7. This simply adds a second Trojan horse to the one that already exists. The second pattern is aimed at the C compiler. The replacement code is a Stage I self-reproducing program that inserts both Trojan horses into the compiler. This requires a learning phase as in the Stage II example. First we compile the modified source with the normal C compiler to produce a bugged binary. We install this binary as the official C. We can now remove the bugs from the source of the compiler and the new binary will reinsert the bugs whenever it is compiled. Of course, the login command will remain bugged with no trace in source anywhere.

```
compile(s)

char *s;
{
        if (match(s,"pattern1")) {
                compile("bug1");
                return;
        }
        if (match(s,"pattern2")) {
                compile("bug2");
                return;
        }
        ...
}
```

Figure 1.7

Moral

The moral is obvious. You can't trust code that you did no totally create yourself. (Especially code from companies that employ people like me). No amount of source-level verification or scrutiny will protect you from using untrusted code. In demostrating the possibility of this kind of attack, I picked on the C compiler. I could have picked on any program-handing program such as an assembler, a loader, or even hardware microcode. As the level of program gets lower, these bugs will be harder and harder to detect. A well-installed microcode bug will be almost impossible to detect.

After trying to convince you that I cannot be trusted, I wish to moralize. I would like to criticize the press in its handing of the "hackers", the 414 gang, the Dalton gang, etc. The acts performed by these kids are vandalism at the best an probably trespass and theft at worst. It is

```
compile(s)

char *s;
{
        if (match(s,"pattern1")) {
                compile("bug1");
                return;
        }
        if (match(s,"pattern2")) {
                compile("bug2");
                return;
        }
        ...
}
```

Figure 1.7

Moral

The moral is obvious. You can't trust code that you did no totally create yourself. (Especially code from companies that employ people like me). No amount of source-level verification or scrutiny will protect you from using untrusted code. In demostrating the possibility of this kind of attack, I picked on the C compiler. I could have picked on any program-handing program such as an assembler, a loader, or even hardware microcode. As the level of program gets lower, these bugs will be harder and harder to detect. A well-installed microcode bug will be almost impossible to detect.

After trying to convince you that I cannot be trusted, I wish to moralize. I would like to criticize the press in its handing of the "hackers", the 414 gang, the Dalton gang, etc. The acts performed by these kids are vandalism at the best an probably trespass and theft at worst. It is

only the inadequacy of the criminal code that saves the hackers from very serious prosecution. The companies that are vulnerable to this activity (and most large companies are very vulnerable) are pressing hard to update the criminal code. Unauthorized access to computer systems is already a serious crime in a few states and is currently being addressed in many more state legislatures as well as Congress.

There is an explosive situation brewing. On the one hand, the press, television, and movies make heros of vandals by calling then whiz kids. On the other hand, the acts performed by these kids will soon be punishable by years in prison.

I have watched kids testifying before Congress. It is clear that they are completely unaware of the seriousness of their acts. There is obviously a cultural gap. The act into a neighbor's house. I should not matter that the neighbor's door is unlocked. The press must learn that misguided use of computer is no more amazing than drunk driving of an automobile.

Acknowledgment

I first read of the possibility of such a Trojan horse in an Air Force critique of the security of an early implementation of Multics. I cannot find a more specific reference to this document. I would appreciate it if anyone who can supply this reference would let me know.

Conferencia original dictada por Ken Thompson
Agosto de 1984
Asociación para la Maquinaria de la Computación