



## REGENERACION INDIVIDUALIZADA DEL CODIGO ENCRIPTADOR

Gerardo Richarte

G.I.S.V.I.

(Grupo de Investigación en Seguridad y Virus Informáticos)

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

E-Mail: gisvi@zorzal.dc.uba.ar

gr5e@zorzal.dc.uba.ar

Bajo dirección de Fabián García

Colaboración de:

Ariel Futoransky

Alberto Soliño

Miriam Lea Teitelman

Damian Leibas

### **Abstract:**

RICE (Regeneración Individualizada del Código Encriptador) es una implementación de AEV2<sup>i</sup> (Auto Encriptado Variante) aplicable actualmente, y como tal garantiza un índice de variación de todo el código viral capaz de burlar cualquier intento de detección mediante los actuales métodos de búsqueda por String Matching.

La principio funcional de este método es volver a crear el código de Encriptación (con el de Desencriptación correspondiente) por cada nueva infección del virus. De esta forma el código de la rutina desencriptadora varía por cada nueva copia y el resultado de la encriptación es distinto cada vez, por ser distinta la lógica (fórmula).

Nota: Todos los códigos aquí presentados son en Assembler para la familia 80x86 de Intel<sup>ii</sup>, pero por sus características se pueden encontrar ordenes de utilidad similar en la mayoría de los microprocesadores generalmente utilizados<sup>iii</sup>.

**Keywords:** Virus, Polimorphic, RICE, AEV2.

### **Introducción:**

Hoy en día, la técnica de detección e identificación de virus de computadoras más utilizada es el String Matching: Los antivirus tiene unos pocos bytes extraídos cuidadosamente del código de cada virus. Para ver si cierto archivo está infectado recorre la base de datos de cadenas de identificación, toma una por una, y las compara contra el programa supuestamente infectado, si este último contiene exactamente estos bytes en el mismo orden que el virus, el programa antivirus da un positivo como resultado de la búsqueda. Si la cantidad de virus aumenta (lo hace a cada momento) la base de datos de los antivirus crece proporcionalmente, crece el tamaño del programa antivirus y a la vez se torna más lento, ya que tiene que contrastar contra más Strings.



Por otro lado, aunque en una primera etapa, este método era bastante efectivo, los programadores de virus no pudieron soportar este "desafío" y comenzaron a buscar la forma de evitar este tipo de búsqueda: ¿Qué pasaría si al código del virus se lo encripta, como un mensaje cifrado, y se le agrega como primera instancia en la ejecución, una rutina que desenscripte el resto del virus? Bueno, simplemente la detección se podría seguir llevando a cabo, el "String" no representaría código puro, si no, código encriptado, el cual es rígido, no varía de copia en copia. Esta encriptación solo le dificultaría la comprensión del código del virus a alguien que trate de ver que es lo que hace, el investigador.

Si seguimos en este camino de evolución, lo próximo sería hacer que la clave de Encriptación/Desenscriptación varíe de copia en copia. ¿Qué pasaría en estos casos? (AEV1). Esta parece una muy buena solución a simple vista, pero analizando un poco más el resultado de la variación de estos virus, nos damos cuenta que la rutina agregada para desenscriptar el resto del virus no varía en nada, más que la clave. La rigidez de ésta parte del código se aprovechó para crear nuevos Strings, y así poder seguir identificando los virus con el mismo método, pagando el precio de que el espacio de selección de un String representativo es mucho más reducido que en un virus de primera generación o un AEV1<sup>iv</sup>.

El siguiente paso lógico en esta cadena de refinación de la variación, sería transformar el único lugar del virus que aun es rígido: ¿Qué pasaría si no sólo se cambia la clave de la Encriptación/Desenscriptación, si no, toda la rutina?

Este problema aquí planteado, viene a darle la muerte a los actuales métodos de detección e identificación de virus por String Matching, pero no es tan simple de implementar. Estamos investigando a cerca de nuevos métodos de identificación; pero en un futuro bastante probable, la única defensa contra el ataque de los virus será la implementación de una Política de Seguridad.

## **1. Autoenscriptado Variante de Segunda Generación.**

El objetivo de este tipo de código es hacer imposible la detección de un virus por medio de los actuales métodos de búsqueda por String Matching. Para lograr este objetivo es fundamental que entre copia y copia del virus, no quede ninguna cadena representativa, esto es, que no haya en una copia del virus, un grupo de bytes (cierta cantidad mínima) que repita su estructura en otra copia distinta del mismo virus.

Lo que el AEV2 (Auto-Encriptado Variante de 2<sup>a</sup> Generación) hace es variar la rutina de encriptación y así logra variar todo el virus.

## **2. El método RICE.**

### **2.1 Desarrollo Teórico.**

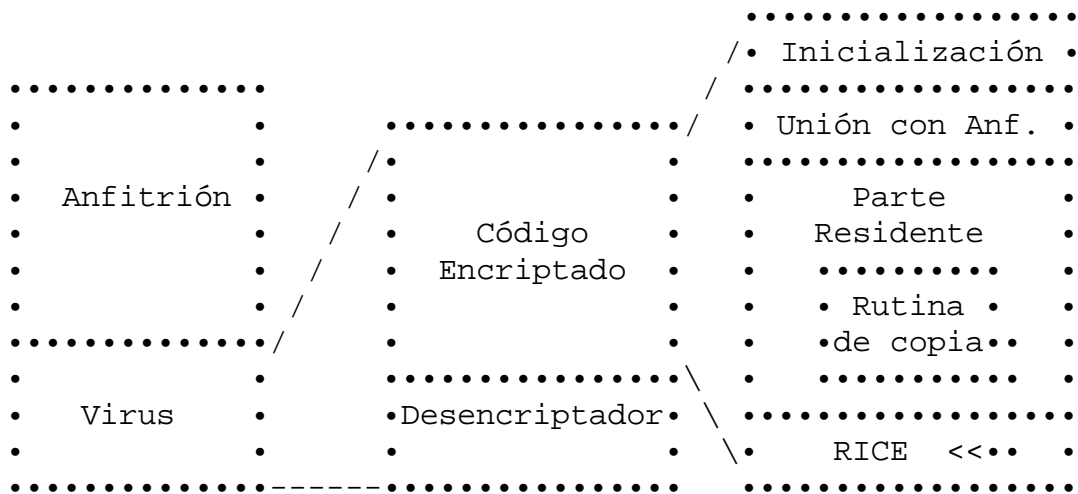
Este método basa su funcionamiento en la creación de nuevas rutinas de encriptación y desenscriptación en el momento en que el virus va a infectar un nuevo anfitrión.

Siendo estas rutinas creadas sin más que una base estructural, y eligiendo su longitud y las ordenes que la componen al azar, está asegurado el que esta parte del virus cambie de copia en copia. La rutina creada está compuesta por operaciones matemáticas cuyos parámetros también son elegidos al azar. De esta forma al encriptar el resto del código viral con esta rutina, se asegura la variación de la parte que quedará encriptada, a la que llamaremos parte fundamental, por ser la que contiene la mayor parte del código del virus y a la vez la rutina RICE propiamente dicha.

#### **2.1.1 Estructura de un Virus RICE.**



Si observamos un programa anfitrión ya infectado vemos que además de su código original, aparece el código del virus sumado. El virus puede ser dividido en dos grandes partes: La rutina de descryptación creada por él mismo en el momento de la infección, y la parte que quedó encriptada al aplicarle el proceso correspondiente. Esta parte encriptada, es la parte fundamental del virus, y contiene todo el código necesario para el su funcionamiento, la parte residente (si la hay), la rutina de copia, la rutina RICE propiamente dicha, las instrucciones que aseguran que el programa anfitrión siga funcionando lo más normalmente posible (para de esta forma pasar inadvertido), etc.



**Fig.1** Estructura de un virus RICE

### 2.1.2 Funcionamiento de un virus RICE.

Un virus RICE puede ser de ataque único (que sólo lleva a cabo la infección de algún nuevo anfitrión en el instante en el que es ejecutado un programa previamente infectado) o residente, que esta siempre activo en memoria esperando el momento indicado para llevar a cabo la infección.

En el momento en el que se va a infectar un nuevo anfitrión es cuando la rutina RICE debe entrar en funcionamiento. Es en este momento cuando se crean en memoria las nuevas rutinas de Encriptación y Descryptación. Una de las tantas formas para cumplir con este paso del proceso, aunque no sea la única, podría ser:

- Elegir al azar una cantidad de operaciones matemáticas N.
- Crear la rutina de encriptación componiéndolas por N operaciones tomadas al azar de un grupo de ordenes útiles.
- Elegir al azar la orden de operación sobre el dato a encriptar.

**Fig.2** Creación de la rutina de encriptación.



```

.....
• Duplicado del •>•Encriptado de la• • Unión •
••virus en memoria. •>•copia del virus. •>•Virus-Anfitrión. •
• .....
• .....
• • Creación de las ••
• • nuevas rutinas. •
• .....
•• Momento de disparo del
Proceso de infección.

```

**Fig.3** Secuencia de Infección

Una vez creada esta rutina, se hace una copia del virus en memoria y se la encripta. En este estadio del proceso se encuentra en memoria la copia funcional (desencriptada) del virus, la copia del virus encriptada con la nueva rutina, y la rutina recién creada. Para crear la rutina de desencriptación, sólo hace falta cambiar la orden de operación sobre el dato a encriptar, en la rutina de encriptación, por su inversa.

Lo último que quedaría es unir al programa anfitrión la rutina de desencriptación y el virus encriptado, armando correctamente la cadena lógica de ejecución. Esto no se diferencia mucho de los métodos utilizados comúnmente, aunque si hace falta tener en cuenta que, al ser variable (al azar) la cantidad de ordenes en la rutina de desencriptación, también lo es su largo, y quizás haga falta tener un poco de cuidado al calcular el punto de entrada al programa, ya que este varía con respecto al final del archivo. Pero esto no tiene porque transformarse en un problema.

## 2.2 Implementación.

Hay muchas formas de llevar a la práctica este tipo de virus, lo que comentamos a continuación no es una implementación en sí, si no, algunas formas de resolver las distintas partes del código, y un repaso de algunas de las cosas que podrían pasar inadvertidas, potenciales objetivos de un posible método de identificación.

### 2.2.1 Operaciones Útiles para Encriptar.

La rutina de encriptación y desencriptación va siendo creada tomando, al azar y de una tabla, distintas operaciones. Estas operaciones no van a afectar directamente al dato a encriptar, si no que van a hacer variar la clave con la que finalmente se va a encriptar.

Orden
ADD XX,YY
ROL XX,YY
XOR XX,YY
NOT XX
NEG XX
DEC XX
SHL XX,YY
OR XX,YY
AND XX,YY
*RCR XX,YY



\*ADC      XX,YY  
\*SUB      XX,YY

**Fig.4** Tabla de algunas ordenes básicas para encriptar. Las ordenes marcadas no pueden ser utilizadas en primer lugar por depender del estado del Carry Flag, pero de ser necesario ampliar el numero de ordenes se pueden tener en cuenta.

Los parámetros de estas operaciones son también elegidos al azar, de esta forma se le da una varianza mayor a la parte fundamental del código.

Estas ordenes no parecen muchas, pero la lista se puede seguir ampliando: Como en este punto lo fundamental es la cantidad de ordenes de las que se disponga a la hora de elegir, se pueden tomar las ordenes inversas a las de esta tabla. De esta forma la lista crece y aumenta el grado de variación de la rutina descriptadora, que es la que va a estar al acecho de los Scanners de virus.

Orden

---

ADD XX,YY  
XOR XX,YY  
NOT XX  
NEG XX  
DEC XX  
SUB XX,YY  
ROR XX,YY  
INC XX  
SHL XX,YY  
OR XX,YY  
XCHG XL,XH  
XOR XL,XH  
SUB XL,XH  
DEC XH  
SHL XL,YY  
OR XH,YY  
AND XH,YY

**Fig.5** Tabla ampliada de ordenes para encriptar.



Si tenemos en cuenta la arquitectura de los registros en un microprocesador de la familia 80x86 de Intel (la de los 68XX y 68XXX de Motorola es similar en este aspecto), utilizando un registro de 16 bits como base de las transformaciones, se podría operar sobre sus componentes de 8 bits, lo que virtualmente triplicaría la cantidad de ordenes (Fig.5).

Para seguir ampliando la cantidad de ordenes a utilizar en la rutina de descriptación, y por consiguiente su índice de variación, se puede implementar una lógica que, en el momento de creación de las rutinas, valla cambiando el registro sobre el que operan las distintas ordenes. De esta forma vuelve a aumentar considerablemente la cantidad de ordenes utilizables.

Para manejar esto hace falta ampliar la estructura de datos, agregando una tabla para cada registro y una lista de las tablas disponibles. Hay que tener en cuenta que cada vez que se cambia de registro, hace falta agregar una orden de traspaso de los datos entre los distintos registros.

Para elegir las órdenes que sirven para operar directamente sobre el dato a encriptar hay que tener un poco más de cuidado, ya que para cada una de estas tiene que haber una operación inversa: la que se va a usar para descriptar.

También hay que tener en cuenta que para cada registro tiene que haber por lo menos una.

```
Lista: ([OPER_AX],[ENC_AX]),
        ([OPER_BX],[ENC_BX]),
        ([OPER_CX],[ENC_CX]),
        ([OPER_DX],[ENC_DX])

OPER_AX:  [ADD AX,YY],[XOR AX,YY],
          [ROL AX,YY],[AND AX,YY]...
OPER_BX:  [ADD BX,YY],[XOR BX,YY],
          [ROL BX,YY],[AND BX,YY]...
OPER_CX:  ...
OPER_DX:  ...

AX_TO_BX: [MOV BX,AX],[PUSH AX POP BX]
AX_TO_CX: [MOV CX,AX]
...
DX_TO_CX: [MOV CX,DX]

ENC_AX:   ([ADD [SI],AX],[XOR [SI],AX])...
ENC_BX:   ...
...
```

**Fig.6** Estructuras necesarias para crear las rutinas de encriptación y descriptación

### 2.2.2 Rutinas de Encriptación y Descriptación.

La cantidad de ordenes de las que estas rutinas están compuestas es elegida al azar entre un mínimo y un máximo. De esta forma (y a la vez por ser las ordenes de distinto largo), el tamaño del virus entre copia y copia es variable. Hagamos un calculo aproximado de la cantidad de



variantes del código generado: Si el mínimo y el máximo son 8 y 17 respectivamente, y en la tabla hay ordenes de 1 hasta 4 bytes, obtendríamos una variación en la longitud de 60 bytes en un caso extremo.

Supongamos que las rutinas van a estar compuestas por N operaciones: Debemos tomar N ordenes al azar de las tablas, y formar con ellas una rutina, a la vez hay que llevar a cabo los cambios de registros, que también deben ser elegidos al azar.

<p>N Veces</p> <p>ADD AX,XX</p> <p>ROL AH,XX</p> <p>...</p> <p>MOV DX,AX</p> <p>XOR DX,XX</p> <p>...</p> <p>SUB [SI],DX</p>	<p>Elección de N</p> <p>Orden #1</p> <p>Cambio de registro? NO</p> <p>Orden #2</p> <p>Cambio de registro? NO</p> <p>Orden #L</p> <p>Cambio de registro? Si, a DX</p> <p>Orden #L+1</p> <p>Cambio de registro? No</p> <p>Orden #N (Ultima orden)</p> <p>Orden de Operación Directa (Con el último registro)</p>
---	--

**Fig.7** Posible rutina de encriptación generada al azar a partir de las tablas.

Cada una de las operaciones elegidas para transformar la clave de encriptación, lleva un parámetro que es elegido al azar en el momento de insertar la orden en la rutina, a medida que se va creando.

El registro índice utilizado puede ser indistintamente SI o DI. El incremento del índice puede realizarse de varias formas distintas, de esto también se podría confeccionar una tabla para elegirlo al azar.

Orden	Cambia SI	Cambia DI	Longitud
CMPSW	SI=SI+2	DI=DI+2	1 byte
SCASW		DI=DI+2	1 byte
2*SCASB		DI=DI+2	2*1 byte
2*CMPSB	SI=SI+2	DI=DI+2	2*1 byte
2*INC SI	SI=SI+2		2*1 byte
2*INC DI		DI=DI+2	2*1 byte

**Fig.8** Ordenes para incrementar los registros de índice.

Estas ordenes, las de incremento de los registros índice pueden llevarse a cabo en cualquier instante del proceso de encriptación, al igual que la orden de operación directa sobre el código a encriptar, para así cambiar un poco la estructura de la rutina desencriptadora. También pueden ser utilizados los registros BP y BX como índice.

Sólo falta hacer que todo esto se repita, y eso se puede hacer comparando el registro que se usa como índice con el valor de la última posición del código a encriptar y desencriptar. En este punto podría presentarse un problema, ya que la orden de comparación debe ser siempre la

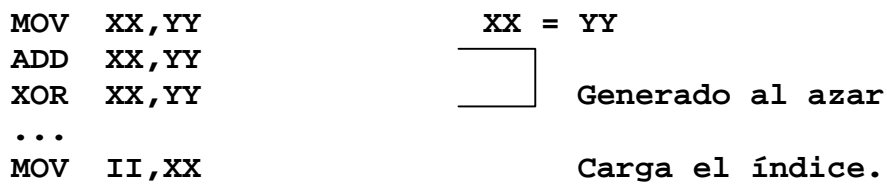


misma, pero el valor contra el cual se compara varia, y sólo queda un byte que aparecería en todas las copias del virus.

Como el valor contra el cual se compara varia en un rango muy chico, y para darle una varianza mayor a la orden de operación directa, se le puede sumar un desplazamiento al índice. De esta forma también la inicialización es variable.

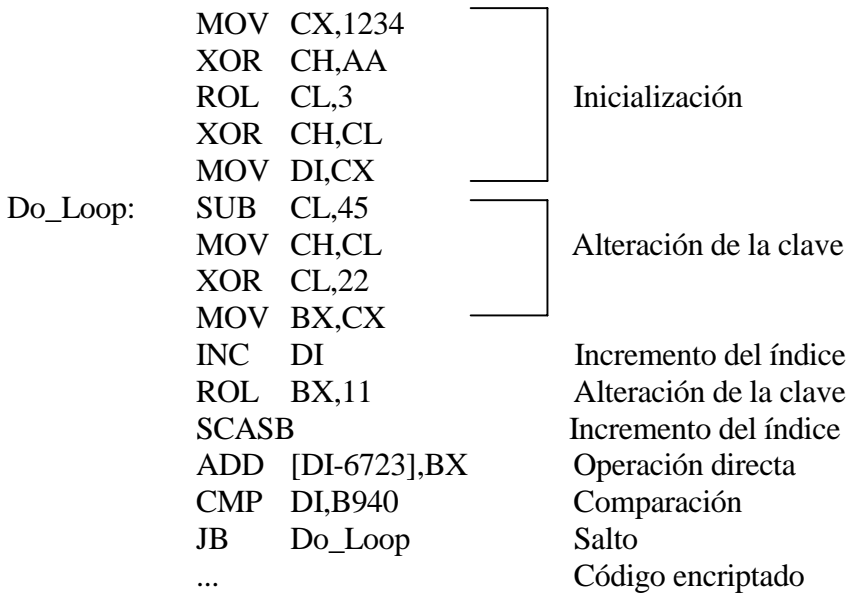
### 2.2.3 Inicialización de los Registros.

Para poder llevar a cabo la descriptación es necesario que los registros de índice y de operación partan de valores conocidos, no importa cuales, pero deben ser los mismos en el proceso de encriptación y en el de descriptación.



**Fig.9** Posible rutina de inicialización.

De esta forma podemos crear completamente, una rutina de encriptación y descriptación variable.



**Fig.10** Rutina de Descriptación creada a partir de las tablas.

### 2.2.4 Encriptación de la Copia del Virus en Memoria.

Una vez terminada la rutina de encriptación hay que encriptar todo el código, esto se hace llamando a la rutina, que por si misma inicializara todas las variables necesarias para llevar a cabo la encriptación.



Para hacer variar un poco más las rutinas creadas, se puede comenzar a encriptar a partir del salto inclusive (el 'JB' en la Fig.10), ya que éste va a ser lo primero en desencriptarse y cuando se ejecute por primera vez ya estará en su forma correcta.

Esta operación nos sugiere que también se podrían encriptar de una forma variable, agregando unas ordenes a la rutina de inicialización, todas las partes del código con poca varianza, como la de comparación del índice. De esta forma, abusando de la gran varianza que nos ofrecen ciertos sectores del código le damos más varianza a otros sectores.

	XOR	[CMP_ENC],1234	Desencripta CMP DI,YY
	...		
Do_Loop:	...		
	...		
CMP_ENC:	DB	??,??	CMP Índice,YY Encriptado
	JB	Do_Loop	Esto queda encriptado

**Fig.11** Encriptación de partes poco variables.

### 2.2.5 Unión con el Anfitrión.

La unión con el anfitrión de este tipo de virus no es muy distinta a la de cualquier otro tipo: Puede colocarse atrás, o adelante (físicamente), pero siempre formando bien la secuencia de ejecución.

Quizás la única diferencia se pueda encontrar en el proceso de desinfección, que se hace casi imposible en los casos en que el virus se coloca físicamente atrás del anfitrión. Al reemplazar los primeros bytes del anfitrión con el salto necesario para armar la secuencia de ejecución, es necesario guardar los bytes cambiados para luego restablecerlos en el momento de ejecución. Estos bytes son necesarios para la desinfección, pero por encontrarse dentro de la parte encriptada, no se los puede obtener, entonces el programa infectado no puede ser desinfectado sin que deje de funcionar. Esto se podría considerar como un principio de virus dependencia.

### 2.3 Índice de Variación de un Virus RICE.

Son muchas las implementaciones posibles de virus RICE, por eso analizamos los índices de varianza de los virus creados de la forma que explicamos en el punto 2.2.

#### 2.3.1 Índice de Variación de la Rutina Desencriptadora.

El índice de Varianza del sector de inicialización se puede calcular como  $7 \cdot 10^5 \cdot 8 = 5600000$  rutinas distintas: el 7 responde a la cantidad de ordenes de inicialización de distintos registros, el  $10^5$  a las ordenes de operación (5 ordenes tomadas entre 10), y el 8 a las de asignación de registros índice.

Para calcular el índice de variación de la parte que opera sobre la clave de encriptación, tenemos que tener en cuenta que se eligen un mínimo de 8 y un máximo de 17 operaciones de una lista de aproximadamente 10, esto da un total de entre  $10^8$  y  $10^{17}$  distintas rutinas de variación de la clave.

El salto de la comparación para hacer que el proceso se repita queda encriptado junto con el resto del código del virus.



La comparación del registro índice para reconocer el final del proceso, queda encriptada bajo la orden agregada en la rutina de inicialización. Esta orden es tomada al azar entre una lista de al menos 10, y es colocada en cualquier punto de la rutina de inicialización.

### **2.3.2 Índice de Variación de la parte Fundamental.**

La rutina de encriptación funciona haciendo variar la clave y aplicándola al código. La clave va variando acorde con una fórmula que es creada como ha sido explicado. Por la estructura de esta fórmula, la secuencia que sigue la clave se puede decir que es pseudoaleatoria y la encriptación responde a este parámetro.

### **2.4 Posibles Puntos Débiles.**

Para buscar puntos que puedan ser Scanneados mediante String Matching, hay que identificar los puntos del código que menos varíen, estos son las ordenes de 7 y 8 posibilidades en la rutina de inicialización, esto nos da 56 posibles Strings representativas con solo 2 bytes conocidos.

Al elegir una String representativa, hay que tener en cuenta que esta no debe aparecer comúnmente en programas de uso común, porque las detecciones falsas pueden ser casi tan peligrosas (o quizás más) que las detecciones acertadas. Por eso, quizás de las 56 Strings que podemos formar en la rutina de inicialización, sólo 20 a 30 sean representativas, pero por el otro lado quedan de 16 a 26 versiones del virus que no pueden ser detectadas de esta forma.

Puede implementarse un método de búsqueda de estructuras, al que le sea posible identificar una rutina de descrición como tal, pero como podría este proceso identificar la parte encriptada como un virus? De utilizarse este proceso, la incertidumbre creada al encontrar una "Posible Infección" sería tal que, o se perderían programas sanos, o no se le prestaría atención a la advertencia en los casos de infección verdadera.

### **2.5 Actualidad del método RICE.**

El peligro que representa el que en la actualidad se encuentren a disposición de quien sepa buscarlas, librerías de códigos para hacer que un programa se transforme en AEV2 (AutoEncriptado Variante de Segunda generación), exige que se tome alguna medida en lo que concierne a la integridad de la seguridad en sistemas informáticos, y quizás exija un replanteamiento de la política adoptada con respecto a esto.

Hay varias librerías circulando en el ambiente underground, una de ellas, la MtE (Mutant Engine) cuenta con varias versiones y un gran abanico de distribución, por lo que en la actualidad debemos sufrir el ataque de severos virus creados con éste método.



### 3. Virus Mutantes.

A pesar de que la rutina de descryptación varíe de copia en copia, no se puede decir que el virus creado bajo una técnica RICE es mutante, ya que hay una gran cantidad de características de mutabilidad con las cuales estos virus no cumplen.

El término *mutación*, al igual que *virus*, surgieron por analogía de los nuevos hechos con los más familiares de la naturaleza. Para poder decir que un programa o un virus es mutante, es necesario analizar el significado de este término en un entorno informático. Este concepto está siendo estudiado actualmente por el GISVI, y será el tema central en un futuro trabajo.

### 4. Sistema Monocopia.

Uno de los problemas que presenta la creación y puesta en funcionamiento de este tipo de virus, es evitar la multiinfección de un mismo archivo: Como el virus cambia al azar de copia en copia, ni él mismo puede asegurar cuándo un programa se encuentra previamente infectado para no volverlo a infectar.

El sistema monocopia, lo que propone es que por disco rígido halla un solo programa (de uso muy frecuente) infectado, y que se lleve a cabo un control de cuando hay que infectar o no un nuevo disco rígido por detección del virus en memoria. A la vez, este sistema infecta indiscriminadamente programas en disquette, para asegurarse de esta forma la dispersión de sus copias.

### 5. Peligros que Presenta el Metodo RICE y posibles Soluciones.

Los virus de tipo AEV2 (y RICE en particular) pueden eludir los programas de detección de virus más difundidos actualmente: los que trabajan por String Matching.

Métodos Euristicos están siendo implementados para la detección de virus, pero las respuestas que ofrecen son bastante cuestionables, ya que es posible que una búsqueda resulte en un falso positivo o un falso negativo. Esta técnica esta recién en sus inicios, con el tiempo es posible que sistemas expertos de detección de virus sean lo suficientemente exactos como para poder confiar en ellos siegamente

Una solución perfectible para este problema, es controlar la integridad de los programas. Esto es relativamente facil de implementar en un ambiente de trabajo reducido (usuarios particulares o donde son pocos conocidos los que comparten una computadora), una de las formas más simples de hacer esto es controlar que la longitud de los archivos no cambie aparentemente sin razon, una forma un poco más sofisticada sería controlar que el CRC no cambie. Para poder hacer esto hay que tener cuidados adicionales, como el estar completamente seguro de que en el momento de guardar los valores (longitud o CRC) el programa está libre de virus, y que cuando se controlan, el sistema está libre de virus.

Para implementar este tipo de soluciones en ambientes de trabajo más grandes, es necesario analizar cada situación en particular, y montar toda una política de seguridad antivirus<sup>v</sup> que directamente no le permita al virus ingresar al sistema.

### 6. Conclusiones.

- No son necesarios amplios conocimientos para poder crear un virus de este tipo.
- El método de Scanning por String Matching pasaría a cumplir un papel secundario en la búsqueda e identificación de una infección.



- Ante el peligro que esto significa, es necesario un replanteamiento de la Política de Seguridad Informática para enfatizar la prevención, más que seguir investigando en métodos de búsqueda, para mejorar el remedio.
- Este método ya es una realidad y es necesario que se tome alguna medida rápidamente.

---

<sup>i</sup>GARCIA, Fabián. HOJMAN, Gabriel. CAMPANELLO, José Luis.  
Selfcrypting-Varying Virus (Second Degree)  
1991

<sup>ii</sup>WYATT, Allen L.  
Using Assembly Language.  
2nd Edition.  
1990, QUE Corporation

<sup>iii</sup>MC6809 Data Sheet  
1983, Motorola Semiconductors.

<sup>iv</sup>GARCIA Fabián. HOJMAN, Gabriel. CAMPANELLO, José Luis.  
Selfcrypting-Varying Virus (First Degree)  
1991

<sup>v</sup>Políticas de Seguridad Anti Viurus. G.I.S.V.I. Actualmente en Preparación.)