Criptografía y seguridad informática

Virus en Python

Grupo 3

Primer cuatrimestre 2023

Alumno	**Padrón**
Carol Lugones Ignacio	100073
Lovera Daniel	103442
Torresetti Lisandro	99846
Zaietz Azul	102214

Introducción **3**
Definiciones **3**
Estructura de los virus **5**
Clasificación 6
Clasificación por Target 6
Boot-sector infectors 6
File infectors 6
Macro virus 8
Clasificación por Concealment Strategy (Estrategia de ocultación) 9
No Concealment 9
Encryption 9
Stealth (sigilo) 9
Oligomorphism 10
Polymorphism 10
Self-detection (auto detección) 10
Cambios en el Decryptor Loop 11
Metamorphism 14
Strong Encryption 14
Detección de virus **15**
Métodos de detección estáticos 15
Escáneres 15
Testeo de escáneres 15
Mejoras a la performance de los escáneres 16
Heurísticas 16
Chequeos de integridad 16
Métodos de detección dinámicos 16
Monitor de comportamiento 17
Emuladores 17
Verificación, cuarentena y desinfección 17
Verificación 17
Cuarentena 17
Desinfección 18
Técnicas Anti-sigilo 18
Detección de Macro Virus 19
Defensa contra antivirus **20**
Técnicas utilizadas contra los antivirus 20
Retrovirus 20
Entry Point Obfuscation (EPO) 20
Anti emuladores 20
Outlast 20
Outsmart 20
Overextend 20
Armoring 21

Anti-debugger.....21
Anti-disassembly..... 21
Tuneleo.....21
Ataques contra los chequeos de integridad..... 21
Avoidance.....21
Prueba de Concepto..... 22

Introducción

El objetivo principal de este trabajo práctico es profundizar el estudio del tipo de *malware* conocido como **virus**. O sea, programas de computadoras que se pueden copiar a sí mismos e infectar una computadora sin permiso o conocimiento del usuario. Además, se analizará su funcionamiento, su capacidad de propagación, así como las consecuencias que pueden tener en un sistema infectado.

A lo largo del trabajo se presentarán las distintas técnicas que se utilizan para detectar y eliminar los virus, y se realizará una comparación entre ellas para determinar cuál es la más eficaz en cada caso.

Finalmente, se realizará una prueba de concepto que consistirá en implementar un virus en *Python* y analizar cómo funciona en un entorno de pruebas controlado, con el fin de entender mejor su comportamiento.

Definiciones

El software malicioso también conocido como *malware,* es software con intenciones maliciosas, o el cual tiene un efecto malicioso. El espectro del malware cubre una gran variedad de amenazas específicas. El malware puede ser propagado a través de *spam*, o incluso puede ser utilizado para mandar spam, busca aprovecharse de los bugs que haya en el software o también puede ser montado para realizar un tipo de ataque, como por ejemplo *Denials of service (DoS).*

El malware puede ser categorizado en distintos tipos dependiendo del método de operación que posean. Hay tres características asociadas con los tipos de malware:

1. *Self-replicating malware:* busca activamente propagarse creando nuevas copias o instancias de sí mismo. También puede propagarse pasivamente, por ejemplo un usuario copiandolo por accidente, pero esto no se considera *self-replication*.
2. *Population-growth:* describe el cambio general en el número de instancias de malware debido a autorreplicación. El malware que no se autorreplica siempre tendrá un valor cero en esta métrica, pero el malware que tiene esta métrica en cero puede llegar a replicarse.
3. *Parasitic* (parásito): requiere otro código ejecutable para poder existir.

Un ***virus*** es un malware que, una vez ejecutado, trata de replicarse en otros códigos ejecutables, y cuando lo logra se dice que el código está *infectado*. El código infectado cuando se ejecuta a su vez puede infectar archivos que previamente no lo fueron. Esta autorreplicación es la clave para definir la característica de un virus.

Tradicionalmente se propagan dentro de una sola computadora, o pueden propagarse a otras a través de un humano por medio de un USB, CD, etc. En otras palabras, los virus no se propagan por las *networks* (este es el territorio de otro tipo de malware conocido como *worm*).

Los virus se pueden hallar en varios estados de replicación. Para ello se utiliza la siguiente terminología:

- *Germ:* es la forma original del virus, previo a cualquier replicación.
- *Intended:* son los virus que fallan al intentar replicarse, esto puede darse por un bug en el mismo virus o por encontrar una versión inesperada de un sistema operativo.
- *Dormant:* se da cuando el virus está presente pero no infectó nada aún, por ejemplo un virus en Windows puede encontrarse en un file server que utiliza Unix, en el cual no tiene efecto pero puede llegar a ser exportado hacia computadoras que utilicen Windows.

En relación con las tres características de los malware se puede decir que un virus posee todas.

Estructura de los virus

Está compuesto por tres partes:

1. ***Mecanismo de infección:*** es cómo el virus se extiende, modificando otros archivos para que contengan una copia del virus o una versión alterada del mismo. Un término utilizado para referirse a cómo se propaga un virus es *vector de infección*. No tiene por qué ser único, un virus que infecta en muchas direcciones se conoce como *multipartite.*
2. ***Trigger (disparador):*** para decidir si enviar el *payload* o no.
3. ***Payload:*** lo que el virus hace, además de propagarse. El payload puede incluir daño ya sea intencional o accidental (cuando el virus en sí contiene bugs o no funciona como lo esperado).

A excepción del mecanismo de infección, las otras dos partes son opcionales, dado que la infección es la característica principal de un virus. En la ausencia de infección, sólo quedan el payload y el trigger, lo cual conforman un tipo de malware conocido como *Logic Bomb.*

A continuación se muestra el pseudocódigo de la estructura general de un virus:

```
def virus():
    infect()
    if trigger():
        payload()

def infect():
    for i in range(1, k): # Repeat k times
        target = selectTarget()

        if !target:
            return

        infectCode(target)
```

El *target* es un código que puede ser alcanzado localmente por el virus que se está ejecutando en la máquina.
Generalmente se hace una infección progresiva, en el sentido de que se busca infectar *k* targets cada vez que se ejecuta el código del virus. La parte complicada radica en la función *selectTarget* dado que no es deseado que el virus infecte archivos que ya fueron infectados previamente, por dos motivos: puede revelar que hay un virus y es un desperdicio de trabajo. *SelectTarget* tiene que tener una forma de detectar si el código ya fue infectado o no, pero si el virus puede detectarse a sí mismo es posible que lo pueda detectar también algún antivirus.

infectCode es la función que termina infectando el código agregando una versión del virus en los targets.

Clasificación

Los virus pueden ser clasificados de diferentes maneras,como por ejemplo según el tipo de target que buscan infectar o el método que utilizan para esconderse y así no ser detectados por usuarios o antivirus.

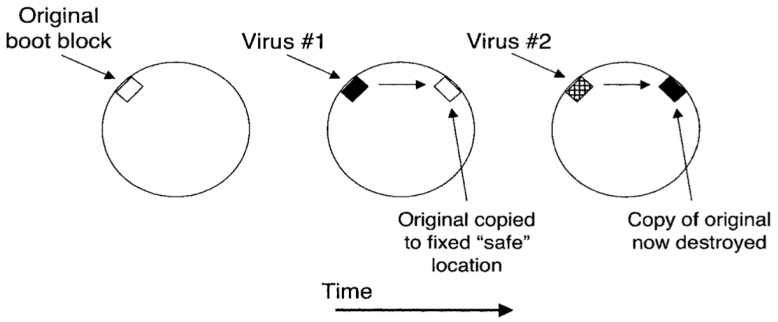
Clasificación por Target

En esta categoría se mencionan tres tipos: *boot-sector infectors, executable file infectors* y *data file infectors* (también conocidos como *macro virus*).

Boot-sector infectors

Es un virus que infecta copiandose a sí mismo en el bloque de booteo. Primero suele realizar una copia del bloque de booteo actual en otra parte del disco, así una vez que complete su proceso le pasa el control a ese bloque para que complete la etapa de booteo.

Hoy en día este tipo de virus no son muy comunes dado que las computadoras no se *rebootean* tanto y muchos sistemas operativos previenen la escritura en disco en el bloque de boot si no tienen la autorización correspondiente.



Multiples Boot Sector Infections

File infectors

Este virus lo que busca infectar son archivos que el sistema operativo o *shell* consideran como ejecutables. ¿Dónde está puesto el virus? hay muchas opciones:

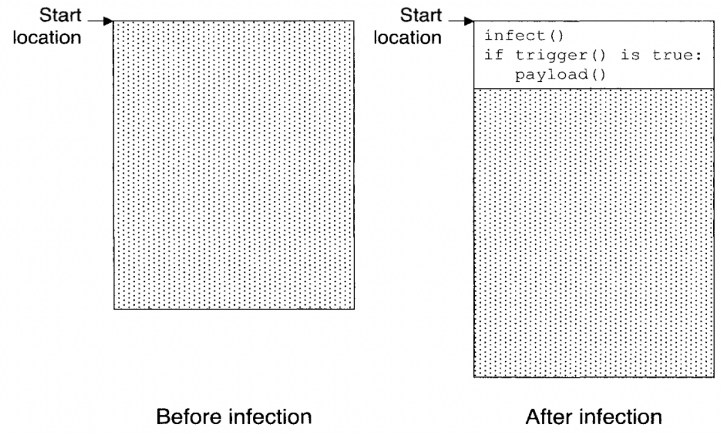
- *Beggining of File*: se posiciona al comienzo del archivo y de esta forma tiene control cuando el archivo infectado se ejecuta. Este tipo de virus se conocen como *preprending virus*. Insertarse al comienzo de un archivo involucra cierta copia de por medio, no es difícil pero a su vez no es la forma más sencilla de infectar.

- *End of File*: a diferencia del caso anterior, agregar un virus al final de un archivo es extremadamente fácil. A estos virus se los conoce como *appending virus.* ¿Cómo obtienen el control? Hay dos alternativas:
    - La instrucción original puede ser guardada y reemplazada por un *jump* al código viral. Luego de que el virus se ejecuta, devuelve el control al flujo principal del archivo infectado para que haga las instrucciones que restan.

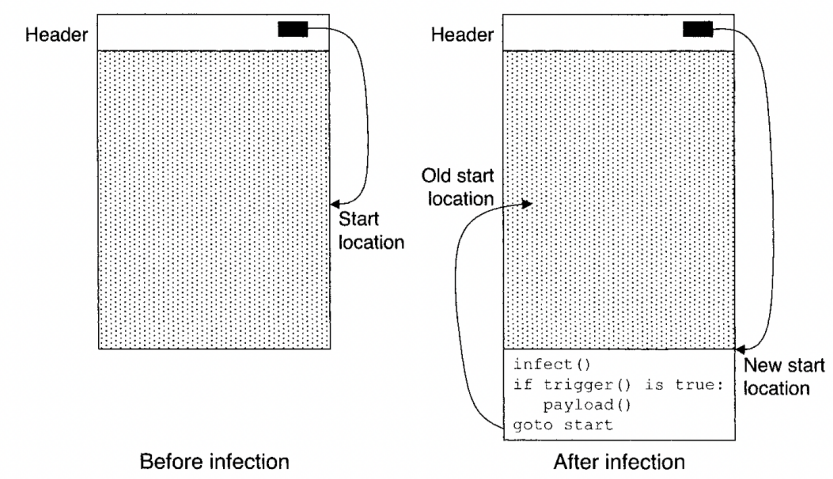
    - Muchos archivos ejecutables especifican la dirección de inicio en un *file header*. El virus puede cambiar esta dirección para que apunte a su código y luego volver a la dirección de inicio original cuando termina de ejecutarse.

- *Overwritten into File*: este tipo de virus se caracterizan por tener un tamaño pequeño que evita un cambio obvio en el tamaño del archivo a infectar, a diferencia de los preprending/appending virus. Obviamente al sobreescribir el archivo original puede llevar a que se lo detecte fácilmente, pero existen varias opciones para tratar de ocultarlo mejor:
    - Puede sobrescribir secciones que tengan valores repetidos en el afán de evitar dañar el archivo original.
    - Sobrescribir una parte aleatoria lleva a que se preserve el contenido original en otra sección.
    - Puede comprimir parte del código original para hacer espacio para sí mismo, y descomprimirlo una vez que completó su ejecución

- *Not in File:* un *companion* virus es aquel que se instala de tal forma que es ejecutado antes del código original. Estos virus nunca modifican el código infectado y ganan control tomando ventaja de los procesos en los cuales el sistema operativo o *shell* buscan archivos ejecutables. Por ejemplo: un companion virus puede posicionarse antes en el *search path* con el mismo nombre que el target file, de esta forma el virus se ejecuta primero cuando se intenta ejecutar el target file.



Prepending virus



Appending virus

Macro virus

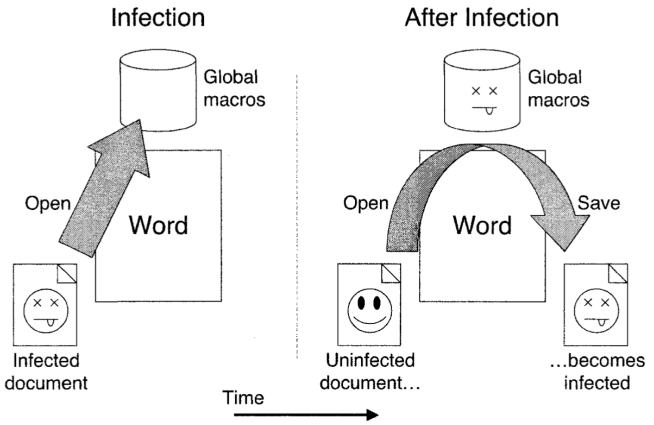
Los macro virus se propagan utilizando macros, que son secuencias de comandos o instrucciones que se ejecutan automáticamente en aplicaciones como procesadores de texto, hojas de cálculo o programas de presentación. Estos virus infectan archivos de documentos que contienen macros.

Cuando un archivo infectado con un macro virus se abre en una aplicación que admite macros, el virus se activa y puede realizar diversas acciones maliciosas, como alterar o destruir datos, propagarse a otros archivos y sistemas, o incluso controlar remotamente el sistema comprometido.

Un ejemplo es el caso de *Concept virus,* el cual tenía como objetivo documentos de Microsoft Word, este virus tiene dos macros:

- **AutoOpen**: cualquier código en esta macro se ejecuta automáticamente cuando el archivo se abre. De esta manera es cómo un archivo infectado toma control.
- **FileSaveAs**: se utiliza para infectar archivos que aún no hayan sido guardados por el usuario.

En la siguiente imagen se muestra el funcionamiento de Concept.



Clasificación por Concealment Strategy (Estrategia de ocultación)

No Concealment

Obviamente no es la estrategia más efectiva, dado que una vez que se lo encuentra es fácil de analizar, pero de todas formas es una estrategia que se utiliza en ciertos casos.

Encryption

La idea de este tipo de virus es encriptar su código (infection, trigger y payload) de cierta manera que sea difícil de detectar. Más que una encriptación lo que buscan hacer es ofuscar esa información.
Cuando el código del virus se encuentra encriptado no es ejecutable hasta que se lo desencripte, para ello se utiliza un *decryptor loop*, que se encarga de esto y transferirle el control. En principio este loop es muy pequeño en comparación con el código del virus, y provee cierto grado de ocultación para evitar ser detectado por un antivirus. ¿Cómo se lleva a cabo la encriptación? Se mencionan 5 formas a continuación:

- **Encriptación simple**: no se utiliza ninguna *key*, solo operaciones básicas sin parámetros como NOT, bitwise rotation, etc.

- **Static Encryption Key**: es una clave de encriptación estática que no cambia de infección en infección. Suele usarse con operaciones aritméticas como suma y operaciones lógicas como XOR.

- **Variable Encryption Key**: la clave comienza con un valor constante, pero cambia a medida que proceden las infecciones.

- **Substitution Cipher**: utiliza un mapeo 1:1. En el caso de un *homophonic substitution cipher* se permite hacer un mapeo 1:n, incrementando la complejidad al permitir que múltiples valores encriptados se correspondan con un valor desencriptado.

- **Strong Encryption**: no hay ninguna razón por la cual los virus no puedan usar *strong encryption.*

Stealth (sigilo)

Un virus de este tipo no solo trata de ocultar su código, sino también la infección producida. Trata de ocultarse de todo, no solo de anti-virus. A continuación se listan un par de técnicas para lograr esto:

- El timestamp del archivo original infectado puede ser restablecido después de la infección, de esta forma el archivo no parece que haya cambiado.
- El virus puede guardar toda la información previa a la infección de un archivo, incluido su timestamp, tamaño, y contenido. De esta forma cuando se ejecute puede

seguir utilizando el comportamiento de antes para que parezca como que nunca fue infectado.

Una variación de este tipo son los *reverse stealth virus*, los cuales hacen que todo parezca infectado, haciendo que los antivirus comiencen a cometer errores al tratar de desinfectarlos.

Oligomorphism

Este tipo de virus tiene un número finito y acotado de diferentes *decryptor loops* a su disposición. El virus selecciona uno de ellos para cada nueva infección.

Para dar unos ejemplos, *Whale* contaba con 30 descifradores, mientras que Memorial contaba con 96.

Polymorphism

Es similar a los *oligomorphism virus* pero la diferencia radica en que para estos virus se cuentan un número infinito de descifradores. Tremor tiene aproximadamente 6 billones de descifradores.
Con respecto a este tipo de virus surgen dos preguntas:

1. ¿Cómo puede un virus detectar que ya ha infectado un archivo si es que se encuentra bien oculto?
2. ¿Cómo el virus cambia el *decryptor loop* de infección en infección?

Estas preguntas serán respondidas a continuación.

Self-detection (auto detección)

El mecanismo de detección tiene que ser independiente del código que utiliza el virus, por lo que se pueden analizar los siguientes atributos para detectar una infección previa:

- ***File timestamp:*** un virus puede cambiar el timestamp del archivo infectado, haciendo que la suma de la hora y la fecha den una constante de valor *K* para todas las infecciones.

- ***File size:*** un archivo infectado podría tener su tamaño aumentado a un tamaño significativo, como un múltiplo de 1234.

- ***Data hiding:*** un virus podría esconder un *flag* en un área que no sea utilizada, o buscar una combinación inusual de los atributos que contiene el archivo.

- ***Filesystem features:*** algunos *filesystems* permiten que se agregue una etiqueta con atributos arbitrarios a un archivo. El virus puede tomar ventaja de esto y puede agregar código, data o flags para indicar que ese archivo ya fue infectado.

- ***External storage:*** un virus podría usar una función de hash para mapear el nombre de un archivo infectado a un string ofuscado, y usar este resultado para crear una *key* en el *Windows Registry,* y de esta manera en un futuro caso utilizar esta key para saber si el archivo fue infectado o no.

Ninguno de estos mecanismos tiene que ser perfecto, dado que un falso positivo lo único que hace es que el virus no infecte un archivo que previamente pudo haber sido infectado.

Cambios en el Decryptor Loop

El código en un virus polimórfico es transformado en cada infección utilizando un *motor de mutación (mutation engine).* Éste cuenta con varios trucos/formas de transformar el código, toma como entrada una secuencia de código y la salida es otra secuencia de código equivalente. El motor para elegir la técnica y en qué parte aplicarla puede utilizar un generador de números aleatorios, permitiendo que el código pueda variar en múltiples formas. A continuación se mencionan diferentes transformaciones:

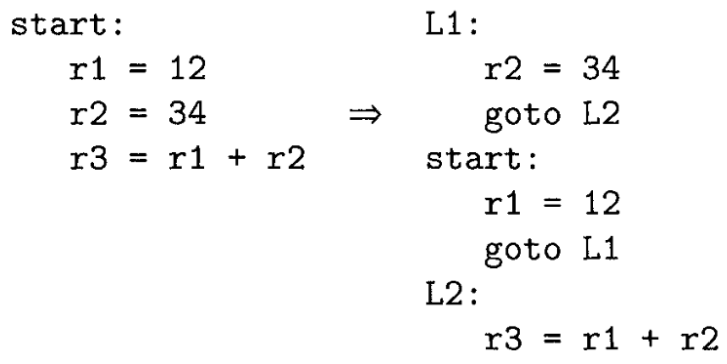
- ***Instrucciones equivalentes:*** más que nada en arquitecturas *CISC* hay múltiples instrucciones que realizan lo mismo. Por ejemplo todas las instrucciones siguientes setean el registro r1 en 0:
  - clear r1
  - xor r1, r1
  - and 0, r1
  - move 0, r1

- ***Secuencia de instrucciones equivalentes:*** son casos más portables y aplicables tanto en lenguajes de alto nivel como de bajo nivel. Por ejemplo: x = 1 ⟷ y = 21, x = y - 20.

- ***Register renaming:*** un cambio menor pero significativo se logra cambiando los registros que alguna instrucción utiliza. Esto no genera cambios viéndolo desde una perspectiva de alto nivel, lo que logra es cambiar el patrón de bits que conforman la instrucción, haciendo que sea más difícil para un antivirus analizar las instrucciones del virus debido a las variaciones producidas.

- ***Reorganización de la data:*** cambiar la posición en memoria de los datos tendrá un efecto similar al caso anterior, o sea es otra forma de lograr que cambie la codificación de la instrucción.

- ***Spaghetti:*** consiste en escribir el código de distintas formas, unas más eficientes que otras. Por ejemplo:



- ***Insertando código basura:*** es código que no hace nada, el código original no presenta cambios en cuanto a su funcionamiento. Por ejemplo:

```
r1 = 12          r1 = 12              r5 = 42
inc r1      ⇐    r2 = 34       ⇒      r1 = 12
inc r1           r3 = r1 + r2      X:
r1 = r1 - 2                           r2 = 34
r2 = 34                               dec r5
r3 = r1 + r2                          bne X
                                      r3 = r1 + r2
```

  El código de la mitad es el original, a la izquierda se puede ver una versión que lo único que hace es incrementar dos veces el valor de r1 para luego volver a restarle 2. En el segundo se agrega un loop que lo único que hace es ir decrementando r5 en cada iteración hasta que llegue a cero, luego realiza la suma de r1 con r2 (sus valores no fueron modificados en el *loop*).

- ***Run-time code generation:*** una forma de transformar el código es haciendo que no siempre esté presente la totalidad del mismo hasta que se ejecute. Por ejemplo:

```
r1 = 12              r1 = 12
r2 = 34         ⇒    r2 = 34
r3 = r1 + r2         generate r3 = r1 + r2
                     call generated_code
```

- ***Concurrencia:*** el código original puede ser separado en múltiples *threads,* lo cual no solo transforma el código, sino que también genera complicaciones extra al momento del análisis automático. Ejemplo:

```
r1 = 12              start thread T
r2 = 34         ⇒    r1 = 12
r3 = r1 + r2         wait for signal
                     r3 = r1 + r2
                        ...
                     T:
                        r2 = 34
                        send signal
                        exit thread T
```

- ***Inlining and outlining:*** *inlining* es una técnica que se utiliza normalmente para evitar sobrecarga de llamadas de subrutinas, esto hace que se reemplace un llamado a una subrutina con el código de la misma, por ejemplo:

```
   ...                        ...
call S1                    r1 = 12
call S2                    r2 = r3 + r2
   ...              ⇒      r4 = r1 + r2
S1:
   r1 = 12                 r1 = 12
   r2 = r3 + r2            r2 = 34
   r4 = r1 + r2            r3 = r1 + r2
   return                     ...
S2:
   r1 = 12
   r2 = 34
   r3 = r1 + r2
   return
```

Outlining es la operación inversa como se muestra a continuación:

```
   ...                        ...
r1 = 12                    r1 = 12
r2 = r3 + r2               r2 = r3 + r2
r4 = r1 + r2               call S12
                     ⇒     r3 = r1 + r2
r1 = 12                       ...
r2 = 34                    S12:
r3 = r1 + r2                  r4 = r1 + r2
   ...                        r1 = 12
                              r2 = 34
                              return
```

- ***Subroutine interleaving:*** las técnicas anteriores mantienen el código original, pero lo rellenan de diferentes maneras. El código también puede ser transformado combinando subrutinas independientes, como en el siguiente ejemplo:

```
   ...                        ...
call S1                    call S12
call S2                       ...
   ...                  ⇒  S12:
S1:                           r5 = 12
   r1 = 12                    r1 = 12
   r2 = r3 + r2               r6 = r3 + r2
   r4 = r1 + r2               r2 = 34
   return                     r4 = r5 + r6
S2:                           r3 = r1 + r2
   r1 = 12                    return
   r2 = 34
   r3 = r1 + r2
   return
```

Metamorphism

Son polimórficos en términos de su código. No son encriptados, por lo que no requieren de un descifrador, pero evitan la detección: una nueva versión del código del virus se produce con cada infección.

El objetivo principal de los virus metamórficos es eludir los mecanismos de detección de los programas antivirus, aprovechando su capacidad para mutar y generar variantes únicas que no coinciden con las firmas conocidas de virus. Al cambiar constantemente, estos virus se vuelven altamente evasivos y difíciles de detectar incluso para los sistemas de seguridad más avanzados.

Si bien los virus polimórficos y metamórficos son difíciles de detectar por los antivirus, también son muy difíciles de implementar correctamente, y esta es la razón por la cual la cantidad de estos virus es pequeña en comparación con los otros tipos.

Strong Encryption

El mayor problema de las técnicas mencionadas anteriormente no es la estrategia de encriptación, sino que los virus cargan consigo sus claves de desencriptación.
Hay dos posibilidades para evitar esta debilidad:

1. La clave proviene por fuera del sistema infectado:

2. La clave proviene del sistema infectado: se generan claves de desencriptación a partir de elementos ya presentes en el target, como por ejemplo:
    - El nombre de dominio de la máquina.
    - La fecha u hora.
    - Alguna data en el sistema (por ejemplo contenido de los archivos).
    - El nombre de usuario.
    - Las configuraciones de la interfaz de idioma.

Combinando la generación de la clave con *strong-encryption* hace que los virus sean muy difíciles de analizar incluso cuando se los detecta. Para analizar completamente uno de estos virus, se tiene que poder desencriptar, y quizás se pueden encontrar los elementos que conforman la clave, pero no se puede descubrir el valor exacto. En este caso la única esperanza para desencriptar correctamente es que la clave se haya elegido de manera muy sencilla, por ejemplo utilizar muy pocos valores puede ser susceptible a que por fuerza bruta se encuentre su valor.

Detección de virus

- Detección: determina si una porción de código es maliciosa o no (booleano).
- Identificación: identifica qué tipo de virus es, en caso de que sea código malicioso.
- Desinfección: remueve los virus detectados.

La detección es el paso más importante, debido a que una detección temprana antes de que haya ocurrido una infección hace que no sea necesaria la identificación ni la desinfección. La detección no es perfecta, existen falsos positivos y falsos negativos. También existen los “positivos fantasma”, estos casos se dan cuando se detectan virus que ya no están presentes, esto ocurre porque ya fueron detectados anteriormente y pasaron por un proceso de desinfección incompleto que hizo que quedaran remanentes que todavía son detectados.

Métodos de detección estáticos

Escáneres

Se clasifican de acuerdo a cuándo son llamados:

- Bajo demanda: los escáneres bajo demanda empiezan a correr cuando explícitamente son iniciados por el usuario. Se utilizan cuando se sospecha que existe una infección o cuando se descarga un archivo sospechoso.
- En acceso: los escáneres en acceso corren continuamente, escaneando cada archivo cuando se accede a él. Como al correr continuamente pueden consumir todos los recursos, algunos permiten configuraciones para indicar en qué casos deberían correr y en cuáles no.

El escaneo es la búsqueda de las secuencias de bytes que representan los virus (firmas). El desafío técnico es encontrar algoritmos que puedan encontrar múltiples patrones de manera eficiente y escalable. Tres ejemplos de algoritmos utilizados son: Aho-Corasick, Veldman y Wu-Manber.

Testeo de escáneres

¿Cómo puede un usuario determinar si su escáner antivirus está funcionando correctamente? Existen archivos de prueba que el antivirus debería detectar como virus en caso de funcionar correctamente. Los archivos de prueba no son maliciosos, están diseñados especialmente para testear los escáneres y no representan ningún tipo de daño potencial para el usuario.

Mejoras a la performance de los escáneres

- Reducir la cantidad escaneada: escanear un archivo completo es lento y aumenta la posibilidad de falsos positivos.
- Reducir la cantidad de escaneos: escanear solo un tipo específico de archivos, tener guardados archivos que ya pasaron el escaneo exitosamente para no volver a escanearlos.
- Reducir el uso de recursos: usando un set más pequeño de firmas.
- Cambiar el algoritmo utilizado: mejorar el algoritmo siempre es una posibilidad.

Heurísticas

Las heurísticas pueden encontrar virus conocidos y desconocidos buscando líneas de código que suelen componer los virus en lugar de escanear firmas específicas de los virus.

Es un análisis estático, esto significa que el código que se analiza nunca es ejecutado. El análisis se realiza en dos partes:

- Data: se recopila información usando un número estático de heurísticas. Los resultados de las heurísticas son combinados y analizados más adelante.
- Análisis: el análisis de los datos de las heurísticas estáticas es tan simple como ponderar el valor de cada heurística y sumar los resultados. Si la suma pasa algún umbral, entonces la entrada se considera infectada.
  Métodos más elaborados de análisis de datos podrían usar redes neuronales, sistemas expertos o técnicas de minería de datos.

Chequeos de integridad

Funciona observando cambios no autorizados en archivos. Almacena los checksum originales de los archivos, para luego poder comparar siempre contra los originales.

Hay tres tipos de chequeos de integridad:

- Offline: solo se realizan chequeos cada un período determinado (por ejemplo, una vez por semana)
- Auto chequeo: los archivos ejecutables son modificados para realizar un auto chequeo de integridad antes de ejecutarse.
- Cascarones de integridad: el checksum de un archivo se ejecuta inmediatamente antes de la ejecución, este chequeo se incorpora en el kernel del sistema operativo.

Métodos de detección dinámicos

Monitor de comportamiento

Es un software antivirus que monitorea el comportamiento de un programa en ejecución en tiempo real. En caso de encontrar algún tipo de actividad sospechosa puede bloquear el comportamiento para impedir que se ejecuten, cortar el programa o puede solicitar al usuario que ejecute una acción apropiada.

Emuladores

Las técnicas antivirus que usan emuladores hacen que el código analizado se ejecute en un ambiente emulado. La emulación puede ser aplicada de dos maneras:

- Con heurísticas dinámicas: son iguales que las heurísticas estáticas. La única diferencia está en cómo se recopilan los datos: heurística dinámica recopila sus datos del emulador sobre el código que se analiza. El análisis se realiza de la misma manera que para las heurísticas estáticas.
- Con descripción genérica: para los virus polimórficos, el bucle de descifrado puede ser muy difícil de detectar para el software antivirus. El descifrado genérico evita este problema confiando en el propio ciclo de descifrado del virus para descifrar el cuerpo del virus.

Verificación, cuarentena y desinfección

Además de la detección, estas tres tareas son parte del antivirus.

Verificación

Los antivirus generalmente ejecutan una segunda verificación luego de la detección inicial de un virus, debido a que la detección no siempre determina de manera correcta si existe un virus o no. Entonces, la verificación se ejecuta para reducir falsos positivos y falsos negativos.

Además, generalmente los programadores de virus hacen que sus virus se parezcan a otros para que sean identificados incorrectamente y falle la desinfección.

Se usa un proceso de “Radiografía” para transformar el virus y obtener más información. Consiste en desencriptar el cuerpo del virus para obtener una firma más larga.

Cuarentena

Cuando se detecta un virus en un archivo, el software antivirus tiene la tarea de aislar el archivo infectado del resto del sistema. Esta medida es temporal, solo hasta que el usuario decida cómo manejar el archivo o hasta que el antivirus haga disponible una versión que pueda eliminar el virus descubierto.

La cuarentena consiste en hacer una copia del archivo en un directorio especial, eliminar el archivo original infectado y deshabilitar todos los permisos de acceso al archivo infectado. Para prevenir casos inesperados, en los cuales los permisos de acceso son modificados, una opción es encriptar el archivo infectado con algún método sencillo para que deje de ser ejecutable.

Desinfección

La desinfección no significa que un sistema infectado sea restaurado a su estado inicial, a pesar de que la desinfección sea exitosa.
Hay diferentes tipos de desinfección:

- Restaurar los archivos infectados desde las copias de seguridad: como mucha gente guarda copias de seguridad, los archivos infectados pueden ser restaurados a sus versiones iniciales guardadas en la copia de seguridad. Esto puede generar pérdida de datos, porque muchos archivos pueden haber sido modificados después de la última copia de seguridad.
- Específico del virus: muchas veces, dependiendo del virus, la desinfección consiste en ejecutar una serie de subrutinas de desinfección con los parámetros adecuados. Estas subrutinas pueden ser ejecutadas automáticamente por los antivirus para virus simples.
- Específico del comportamiento del virus: en lugar de customizar la desinfección para virus individuales, se plantea una solución de acuerdo con el comportamiento de los mismos. Para los virus que toman control modificando el encabezado de un programa, la desinfección es una cuestión de lograr volver al encabezado original, moviendo el contenido de los archivos a su posición original. Los antivirus guardan información que les permita volver al estado original, la información necesaria es: el encabezado del programa, el largo del archivo, una suma de comprobación del contenido del archivo ejecutable sin encabezado (checksum).
- Usando el código del virus: hay dos opciones.
  La primera es usando a favor que los virus sigilosos suministran el contenido no infectado de un archivo. El software antivirus puede explotar esto para desinfectar un virus sigiloso simplemente pidiéndole al virus el contenido del archivo.
  Para la segunda opción, los métodos genéricos de desinfección asumen que el virus eventualmente se restaurará y saltará al código que infectó. Un desinfectador genérico ejecuta el virus bajo condiciones controladas, observando que el código original sea restaurado por el virus en nombre del desinfectador.

Técnicas Anti-sigilo

Hay dos opciones de técnicas contra virus sigilosos:

1) Detectar y deshabilitar el mecanismo de sigilo.

2) Omitir los mecanismos habituales para llamar al sistema operativo a favor de los inquebrantables.

Detección de Macro Virus

Los macro virus presentan algunos problemas interesantes para los software antivirus:

1) Creación accidental de nuevos macro virus por procesos de formateo automáticos o cambios accidentales o deliberados de macro virus.
2) Los bugs en la propagación de macro virus o una desinfección incompleta de un macro virus, puede generar nuevas variantes de macro virus.

3) Un macro virus puede "arrebatar" accidentalmente macros de un entorno que infecta, convirtiéndose en un nuevo virus.

A pesar de estos problemas, los macro virus tienen una característica favorable: operan en un dominio restringido, entonces es fácil para un antivirus poder determinar qué constituye un comportamiento “normal” con un alto índice de confianza, disminuyendo los falsos positivos.

Las técnicas de desinfección para macro virus están basadas en la eliminación:

- Eliminar todas las macros infectadas del documento, incluidas las macros desafortunadas, macros de usuario legítimas
- Eliminar las macros asociadas al virus encontrado. Esto requiere una base de datos de macro virus conocidos.
- Para los macro virus detectados usando heurísticas, eliminar los macros que contengan comportamiento ofensivo.
- La detección basada en emuladores puede rastrear las macros que el virus de macro utiliza y eliminarlas.

Defensa contra antivirus

Con la normalización de los antivirus, los virus empezaron a atacar a estos también para poder reproducirse sin trabas en el medio y sin ser detectados por el end-user o por el antivirus. En principio alguna de las técnicas utilizadas por los virus contra los antivirus realizan una de las siguientes tres cosas:

- Atacar agresivamente a los software de antivirus.
- Dificultar su análisis para los investigadores de antivirus.
- Intentar evitar ser detectados utilizando el conocimiento de cómo funcionan los antivirus.

Técnicas utilizadas contra los antivirus

Retrovirus

Cualquier virus que trate de deshabilitar el antivirus de su máquina se le llama retrovirus. Esta deshabilitación ronda desde recorrer un listado de procesos de antivirus conocidos y matándolos en tiempo de ejecución, hasta deshabilitarlos totalmente poniendo un return en la primera línea del ejecutable o tratar de hacer que estos procesos entren en *starvation*.

Entry Point Obfuscation (EPO)

Ofuscación de punto de entrada (o *entry point obfuscation*) es una técnica que hace que se cambie el punto de acceso al virus, por lo general modificando el exit point del código para hacer que eso inicialice el virus y así lograr que se dificulte la detección para algunos antivirus. Esto es en contraposición a la mayoría de virus que modifican la dirección de inicio de un ejecutable o código.

Anti emuladores

Outlast

Se busca que el virus sobreviva sin ser detectado por el emulador hasta que el mismo sea finalizado por el usuario que los comenzó. Esto se puede lograr haciendo que el virus no haga nada hasta que el emulador se cancele, o también haciendo que el virus actúe de forma benigna bajo una cierta probabilidad.

Outsmart

Busca modificar el código para que sea más difícil detectar el comportamiento malicioso.

Overextend

Hacer que el virus fuerce los límites del emulador para que el mismo no pueda tomar acciones sobre el virus. Este método se puede lograr a través de múltiples medios, a continuación se listan un par de ejemplos:

- Usando instrucciones no documentadas que sean aceptables para el lenguaje pero no para el emulador.

- Atacando la memoria del sistema para que ocurran errores de acceso de memoria.
- Otros métodos que atacan la forma de cómo actúa el emulador.

Armoring

Se dice que un virus está armado si usa técnicas que dificulten el análisis a los investigadores de antivirus.

Anti-debugger

Trata de dificultar el análisis dinámico para los humanos. No es muy funcional ya que una vez que se entra a este método, significa que el código está siendo analizado en un debugger y no le queda mucho tiempo hasta que descubran su finalidad.

Anti-disassembly

Tiene dos objetivos:

1. Desensamblar no debería ser fácilmente automatizado. Se logra con el uso de problemas que son computacionalmente muy difíciles de resolver, como por ejemplo mezclar data con instrucciones ya que para una computadora es muy difícil, en casos imposible, separarlos.
2. El código no debería estar disponible hasta que se deba correr. Esto se puede lograr bajo diferentes medios, como por ejemplo:
    - Generar el código a medida que el virus corre.
    - Modificar el código a medida que el virus corre.
    - Otros métodos más complejos pueden incluir encriptación y desencriptación o realizando cambios con el environment donde se ejecuta el virus.

Tuneleo

Este tipo de virus analiza el código por llamadas a APIs que el virus utiliza y se asegura que el mismo finalice en los lugares correctos, ya sea donde no va a ser monitoreado o donde el monitoreo pueda ser *bypasseado*. Como un punto importante a notar es que el virus es muy efectivo en sistemas operativos que no tengan una protección fuerte de memoria, ya que el mismo necesita poder acceder al código en memoria para poder detectar los puntos débiles.

Ataques contra los chequeos de integridad

Estos virus buscan atacar los antivirus de chequeos de integridad, esto se logra haciendo que el virus ya este en el archivo previamente a que se cargue en la computadora, y que se propague bajo circunstancias donde el chequeo de integridad no detecte los cambios en los archivos o que se hagan cuando el usuario haga alguna modificación en el archivo para que el asuma un falso positivo. Ejemplos de este tipo son los Stealth o companion virus.

Avoidance

Este tipo de virus es muy simple, ya que es simplemente los virus que atacan donde saben que el antivirus no chequea, por ejemplo si el antivirus solo chequea el disco duro, ataca todo lo que no sea disco duro. Si el antivirus no chequea archivos con un nombre específico entonces ataca esos archivos.

Prueba de Concepto

Se realizó la implementación de un virus en python que consiste en un script que es capaz de autoreplicarse cuando se ejecuta, leyendo los archivos .py que se encuentran en el directorio actual del mismo.

```python
def replicate():
    virus = read_virus_code()
    for file_path in find_files_to_infect("*.py", "."):
        infect(file_path, virus)
```

Para poder determinar qué porción de un programa infectado contiene el virus y replicarlo, se utilizan tags al inicio y final del código los cuales varían en función del nombre del archivo y determinan el código que va a ser replicado a otros programas portadores. Esto, con el fin de evitar que todos los archivos infectados puedan ser facilmente detectados. Adicionalmente, para ocultar el código malicioso se realizan transformaciones aleatorias, las cuales incluyen comentarios que no afectan la funcionalidad del malware pero que ayudan a aportar aleatoriedad al código fuente.

```python
def transform_virus_code(virus_code):
  transformed_virus_code = []
  for line in virus_code:
    transformed_virus_code.append("# " + str(random.randrange(1000000)) + "\n")
    transformed_virus_code.append(line + "\n")
  return transformed_virus_code
```

Finalmente, se comprime el código fuente y se codifica en base 64 para conseguir que cada archivo infectado tenga variaciones al momento de uno de ellos.

```python
def obscure_virus_code(virus_code):
    compress_level = 9
    data_bytes = bytes("".join(virus_code), "utf-8")
    return base64.urlsafe_b64encode(zlib.compress(data_bytes, compress_level))
```

El resultado de llevar a cabo estos pasos es que cada archivo tiene diferentes tags de inicio y final, y diferente código fuente por lo cual dificulta la labor de encontrar los archivos infectados, sumado a que el usuario no puede saber a simple vista que es esa porción de script que se encuentra en su programa y cada vez que lo ejecute infectara a otros archivos de su máquina.