Universidad de las Ciencias Informáticas
Facultad 2

Título: Identificación de Scripts Malignos

Trabajo de Diploma para optar por el título de

Ingeniero Informático

Autor: Irian Palmero Mainé

Tutor: Lic. Silvio René Morales Suárez

Julio 2007

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Irian Palmero Mainé

_____

Firma del Autor

Silvio René Morales Suárez

_____

Firma del Tutor

DATOS DE CONTACTO

Información del tutor: Especialidad de graduación: Ciencia de la computación, Categoría docente: Adiestrado, Años de experiencia en el tema: 1, Años de graduado: 1.

RESUMEN

Esta investigación consiste en extraer patrones de código de scripts malignos en ficheros batch y probar que pueden ser utilizados para identificar programas potencialmente malignos. Para esto se realizó un estudio de los principales métodos de detección de programas malignos, así como de las técnicas de ofuscamiento de código utilizadas en este tipo de programas para evitar ser detectados. Se realizó un estudio de 621 scripts malignos para obtener patrones teniendo en cuenta las características de los ficheros batch, las acciones de los comandos del sistema en general, la frecuencia de aparición de algunos comandos en este tipo de programas y los métodos de ofuscamiento de código en este tipo de programas. Se realizó un analizador para probar el reconocimiento de scripts potencialmente malignos en ficheros batch mediante los patrones obtenidos. Las pruebas de reconocimiento de los patrones se hicieron utilizando un método heurístico de análisis estático. El analizador se desarrolló en C++ utilizando la librería Boost-Regex para buscar en los programas los patrones obtenidos, representándolos mediante expresiones regulares.

PALABRAS CLAVE

Scripts malignos, ficheros batch, métodos de detección, ofuscamiento de código.

TABLA DE CONTENIDOS

INTRODUCCIÓN

La desventaja principal con los antivirus convencionales es que ellos sólo pueden detectar programas malignos conocidos. Por tal motivo los diseñadores de software antivirus han estado empleando métodos heurísticos durante algún tiempo, aumentando la posibilidad de detección de programas malignos desconocidos. El análisis heurístico es una técnica poderosa y puede ser particularmente útil junto con las técnicas convencionales, pero tiene algunas desventajas como tiempo y limitación de memoria.

El problema se magnifica si se tiene en cuenta la reciente aparición de virus scripts, que demuestran problemas para las compañías antivirus. Los virus scripts pueden aparecer por todas partes: páginas html, mensajes de correo electrónico, documentos, etc.… Nuevos virus de Windows también aparecen y desconciertan los emuladores y los métodos heurísticos. (1)

Como situación problémica se tiene que: Los programas antivirus de la empresa Segurmática necesitan identificar scripts para buscar código maligno dentro de ficheros texto. Esto es muy difícil puesto que habría que hacer un análisis sintáctico para diferenciar uno de otro. Por ejemplo, hay que saber si un fichero es html, o si contiene comandos del shell de Windows, etc. Pero no es necesario reconocer todo un lenguaje, sino patrones potencialmente peligrosos. La situación se complica con los scripts que contienen comandos corruptos que se ignoran durante el procesamiento pero que pueden confundir a un analizador sintáctico. Además, en un fichero puede haber varios lenguajes script.

El problema científico radica en ¿Cómo identificar scripts potencialmente malignos de ficheros batch mediante el reconocimiento de patrones? De ahí que el objeto de estudio sea el reconocimiento de patrones en programas malignos y el campo de acción sea el código de scripts malignos en ficheros batch.

El objetivo de la investigación es extraer patrones de scripts malignos de ficheros batch que puedan utilizarse para reconocer programas potencialmente malignos. Teniendo como hipótesis que: si se extraen patrones de scripts malignos de ficheros batch se pueden reconocer scripts potencialmente malignos a partir de estos patrones.

Para esto se han definido como tareas: Investigar acerca de los métodos de detección de programas malignos utilizados por programas antivirus fundamentalmente, investigar las técnicas usadas en los programas malignos para hacer más difícil su detección por programas antivirus, analizar el código de scripts malignos en ficheros batch para obtener patrones, es decir, los comandos, parámetros u operadores usados con frecuencia en secuencias de código de programas malignos, y probar que los patrones obtenidos pueden ser utilizados para identificar programas potencialmente malignos en ficheros batch. Estas tareas son tratadas respectivamente en cada uno de los capítulos de este trabajo.

De este trabajo se espera obtener como aporte práctico un conjunto de patrones de código de scripts malignos en ficheros batch.

CAPÍTULO 1. MÉTODOS DE DETECCIÓN

Desde el surgimiento del primer virus hasta la actualidad muchas personas se han dado a la tarea de combatirlos. Los programas malignos van evolucionando con el tiempo y con ellos los métodos utilizados por los antivirus y otros programas de seguridad para detectarlos. Actualmente existen una amplia variedad de programas malignos y su número se incrementa cada vez de forma más rápida. Por lo que desde hace algún tiempo se han venido desarrollando alternativas para hacerle frente a los nuevos retos. Los métodos más utilizados son los el análisis basado en firmas y los métodos preventivos.

1.1 Análisis basado en firmas

El análisis basado en firmas es uno de los métodos utilizados por los antivirus contemporáneos para la detección de programas malignos. Este consiste en buscar en los objetos analizados secuencias de código de software malignos conocidos. A estas secuencias de código se les conoce como firmas y son exclusivas, es decir, que son hechas para un programa específico. La firma para un virus solo sirve para reconocer específicamente a ese virus y, en algunos casos, a la familia del mismo según sea el caso. Por lo que se necesitan casi tantas firmas como programas malignos existan para que un sistema esté bien protegido. Todo esto implica un constante estudio por las compañías antivirus que consiste en registrar nuevos programas malignos, extraer sus patrones o descripciones y si es posible la forma de eliminarlos. Implica además agregar esta información a las bases de datos de los antivirus en forma de actualizaciones, que es un requerimiento fundamental de este método.

Las firmas de virus son una especie de huellas dactilares digitales que los motores antivirus pueden usar para identificar código maligno. Es una efectiva técnica con una destacada vulnerabilidad entre el tiempo en que es lanzada la amenaza, el tiempo en que es descubierta y el tiempo en que la firma es escrita y distribuida. (2)

La mayoría de los antivirus modernos usan expresiones regulares como firmas de virus (3). Estas no deben ser muy estrictas para evitar que un programa maligno con algunos cambios en su código deje de ser detectado con una firma determinada. A veces con cambiar un carácter de un script, sin afectar la lógica del programa, se puede frustrar la detección del mismo por algún antivirus, por ejemplo Kaspersky, que es uno de los mejores según AV-Comparatives.

En general se pueden mencionar como ventajas del método basado en firmas que pueden ser utilizados para detectar los programas malignos antes de que un programa infectado sea ejecutado examinando los nuevos archivos que entran en su sistema, y que normalmente no causan falsos positivos o por lo menos tienen un nivel mucho más bajo que el que presentan los métodos heurísticos.

Entre las desventajas se pueden mencionar que solo pueden detectar programas malignos para los cuales existan firmas; el tiempo de respuesta ante la aparición de nuevos virus es bastante prolongado, es decir, el tiempo que demora obtener nuevas firmas es a menudo un problema para la seguridad ya que los virus contemporáneos son capaces de infectar a millones de computadoras en un período muy corto; otra desventaja es que las bases de datos de los programas antivirus aumenta cada día debido al inevitable incremento de programas malignos, esto hace que el rendimiento y la velocidad se vean comprometidos seriamente. Por estas razones se buscan alternativas como los métodos preventivos.

1.2 Métodos preventivos

La técnica preventiva no involucra las distribuciones de firmas. En lugar de esto, los programas antivirus analizan el código de objetos escaneados y/o el comportamiento de las aplicaciones iniciadas y deciden si el software es maligno basados en un conjunto predefinido de reglas. Dichas reglas pueden estar constituidas por patrones genéricos de código de programas malignos o por patrones de comportamiento que incluyen chequeo de aspectos críticos del sistema, así como el monitoreo de determinadas actividades y el uso de ciertos privilegios; como por ejemplo escribir en zonas no permitidas de la memoria, en sectores de arranque, modificar el registro del sistema, controlar llamadas del sistema, la creación hilos en otros procesos, etc. Claro que estas reglas deben ser configuradas cuidadosamente ya que existe un alto riesgo de encontrar falsos positivos. Existen varios métodos que proporcionan protección preventiva, los más populares son los analizadores heurísticos y los bloqueadores de comportamiento. (4)

1.2.1 Método heurístico

El análisis heurístico está basado en expertos. Este determina la susceptibilidad de un sistema hacia una proporción particular de amenaza/riesgo usando varias reglas de decisión o métodos de ponderación.

Análisis estático

El método heurístico de análisis estático consiste en examinar el código de un programa para buscar comandos sospechosos característicos de programas malignos. Hay diferentes formas de analizar el código, por ejemplo algunos analizadores desensamblan el programa para obtener el código fuente y compararlo con códigos de programas malignos conocidos. Otros incluso lo hacen en combinación con el análisis dinámico, ya que en algún momento el código del programa aparece sin encriptar en memoria. El código es comparado con actividades típicas de programas malignos y si un porciento determinado del código se corresponde con el de virus, el fichero es marcado y se bloquea su acceso.

Cuando se analiza el código de un programa se va incrementando un contador por cada comando peligroso que se encuentre. Una vez terminado el análisis, el valor del contador se compara con un límite determinado según el nivel de seguridad que se quiera y si es mayor que dicho límite entonces el programa se considera sospechoso.

Generalmente los ficheros que contienen código maligno son marcados como peligrosos y se alerta al usuario. Muchas veces se envía una muestra del fichero a la compañía del antivirus para un análisis más profundo. Las ventajas de este método incluyen facilidad en la implementación y alto rendimiento. Sin embargo, la tasas de detección de códigos malignos nuevos son bajas, mientras las tasas de falsos positivos son altas.

Análisis dinámico

La idea de los emuladores surgió como una alternativa para analizar el código de programas malignos encriptados simulando su ejecución. Cuando estos programas se ejecutan el código que se carga en memoria está ya desencriptado, listo para ejecutarse; aprovechando esta brecha, se hace un análisis estático del mismo. De lo contrario habría que utilizar varios algoritmos especializados de detección que permitieran decodificar distintos tipos de encripciones, lo cual sería mucho más costoso en recursos del sistema. Pero ha sido demostrado que con un conjunto de reglas heurísticas de mayor nivel que sean capaces de agrupar los efectos de varias instrucciones pueden ser frustradas todas las técnicas de metamorfismo utilizadas por programas malignos para evitar ser detectados. Aunque en los programas antivirus de hoy en día, el análisis estático sigue siendo usado en combinación con el análisis dinámico estas reglas pueden ir más allá del análisis del código, por ejemplo monitoreando actividades comunes de virus tales como réplica, sobrescritura de archivos e intentos de ocultar la existencia de ficheros malignos.

Si una o más acciones similares a las de los virus son detectadas, el archivo es marcado como un virus potencial, y el usuario es alertado.

Para simular la ejecución de un programa, la mayoría de los antivirus que utilizan el método de análisis heurístico dinámico realizan esta función ejecutando los comandos de un script o un programa cuestionable dentro de una máquina virtual especializada, de ese modo permitiendo al programa antivirus simular internamente lo que sucedería si el archivo sospechoso fuera ejecutado, mientras tanto manteniendo el código sospechoso aislado de la máquina real. Este tipo de entorno virtual seguro es conocido también como sandbox o buffer de emulación.

El proceso de emulación de una aplicación es complicado, se deben utilizar trampas especiales o seleccionar la parte del código convenientemente; por ejemplo para programas que se ejecutan en una determinada fecha o bajo ciertas condiciones. En general este método requiere de mucho más recursos del sistema que el método estático pero por otro lado proporciona mayores tasas de detección de programas malignos que el método estático y menores tasas de detección de falsos positivos.

Ventajas y desventajas del análisis heurístico

La ventaja principal del método heurístico es que puede detectar programas malignos nuevos o desconocidos a diferencia del método basado en firmas. Aunque existan métodos preventivos como el chequeo de integridad que monitorea y evalúa los cambios en los programas y en las áreas del sistema, estos solo detectan un programa maligno después que ha entrado en el sistema (a menudo demasiado tarde). Solo el método heurístico puede detectar un nuevo programa maligno antes de su intrusión en el sistema. Otra ventaja es que por lo general es más rápido evaluar un conjunto simple de reglas heurísticas que usar grandes cantidades de firmas de programas malignos.

Como inconvenientes del método heurístico se puede decir que las reglas que utiliza pueden detectar código o comportamiento maligno con métodos de búsqueda y de monitoreo pero generalmente no permiten revertir cambios o desinfectar archivos dañados, ya que para esto es necesario conocer específicamente lo que hace cada programa maligno. Los métodos heurísticos tienen mayor riesgo de encontrar falsos positivos que los basados en firmas por lo que deben ser configurados cuidadosamente. Por esta razón los antivirus que usan esta técnica reducen la sensibilidad para disminuir el riesgo de falsas alarmas o presentan esa opción deshabilitada por defecto. Otra desventaja es que cuando la interacción con los usuarios para notificar sobre posibles virus es muy pronunciada, esto puede provocar

que con el tiempo estas advertencias sean ignoradas o puede llevar a los usuarios a establecer niveles de protección por debajo de los recomendados, incluso a desactivar completamente los antivirus. Pero su mayor desventaja es la detección de falsos positivos ya que por mucho que pueda reducirse su probabilidad no puede ser excluida. No obstante, esa desventaja puede ser compensada con los beneficios del análisis heurístico.

Efectividad

Aunque el análisis heurístico es capaz de detectar varios virus previamente desconocidos y nuevas variantes de los actuales virus, la efectividad es bastante baja con respecto a la exactitud y el número de falsos negativos. Esto se debe a que los virus informáticos, al igual que los biológicos, están constantemente cambiando y evolucionando. Debido a que el análisis heurístico mayormente opera sobre la base de la experiencia (comparando archivo con el código y funciones de virus conocidos), es probable que pase por alto varios nuevos virus que contienen código desconocido o métodos de operación que no se encuentren en ningún virus conocido. Afortunadamente, el análisis heurístico también está evolucionando junto con los virus. A medida que nuevos virus son descubiertos usando métodos alternativos de detección, se agrega información acerca de ellos a los motores de análisis heurísticos, de esta forma proporcionando modos para detectar cualquier virus nuevo basado en código previamente desconocido. Esto implica que al igual que los métodos basados en firmas, los preventivos también necesitan actualizaciones.

1.2.2 Bloqueadores de comportamiento

Los nuevos ataques frecuentemente pueden penetrar la seguridad de un buen motor antivirus dada la desventaja del método basado en firmas de detectar solamente programas malignos conocidos, y la detección basada en comportamiento pretende enfrentar los exploits día-cero[1] buscando y marcando comportamiento inapropiado. (2)

Un bloqueador de comportamiento es un programa que analiza el comportamiento de las aplicaciones ejecutadas y bloquea cualquier actividad peligrosa. A diferencia de los analizadores

[1] Los exploits día-cero son agujeros de seguridad o instancias que aprovechan un agujero de seguridad que son puestas en circulación antes o en el mismo día en que la vulnerabilidad se hace pública. El término día-cero proviene del número de días entre la publicación del agujero de seguridad y la liberación del exploit.

heurísticos, donde las acciones sospechosas son rastreadas en modo de emulación (análisis heurístico dinámico), los bloqueadores de comportamiento trabajan en condiciones reales.

Los bloqueadores actuales son capaces de monitorear un amplio rango de eventos en el sistema. Su propósito primario es controlar actividades peligrosas –eso es, analizar el comportamiento de todos los procesos corriendo en el sistema y guardar información acerca de todos los cambios en el sistema de ficheros y en el registro. Los bloqueadores pueden además detectar rootkits, que son conjuntos de herramientas usadas frecuentemente por los intrusos o crackers que consiguen acceder ilícitamente a un sistema informático. Estas herramientas sirven para esconder los procesos y archivos que permiten al intruso mantener el acceso al sistema, a menudo con fines maliciosos.

Una característica de los bloqueadores que vale la pena mencionar es su habilidad para controlar la integridad de aplicaciones y el sistema de registro de Microsoft Windows. En este caso, un bloqueador monitorea los cambios en las claves del registro y puede ser usado para definir reglas de acceso a ellas para diferentes aplicaciones. Esto hace posible revertir los cambios en caso de haber detectado actividades peligrosas, a fin de recuperar el sistema y retornarlo al estado antes de la infección, aun después de que programas desconocidos hayan realizado actividades malignas.

A diferencia del análisis heurístico, que es usado en casi todos los programas antivirus contemporáneos, los bloqueadores de comportamiento son mucho menos comunes.

Para resumir, los bloqueadores de comportamiento pueden prevenir que virus conocidos o desconocidos se diseminen, lo cual constituye una ventaja indiscutible de este tipo de método de protección. Por otro lado, aún las últimas generaciones de bloqueadores de comportamiento tienen una desventaja importante: acciones de programas legítimos pueden ser identificadas como sospechosas. Además, se requiere de la intervención del usuario para el veredicto final respecto a si la aplicación es maligna o no, lo que significa que el usuario necesita tener los conocimientos suficientes para esto. (4)

1.2.3 Métodos preventivos vs. basados en firmas

Los métodos de protección preventiva descritos arriba (heurísticos y bloqueadores de comportamiento) están basados en el conocimiento de acciones sospechosas características de programas malignos.

Los desarrolladores de antivirus no tienen otra opción que actualizar su conjunto de reglas de comportamiento y perfeccionar su heurística como respuesta a la aparición de nuevas amenazas. Estos tipos de actualizaciones son ciertamente menos frecuentes que en el caso de las firmas de virus, pero aún necesitan ser realizadas regularmente. Conforme aumente el número de nuevas amenazas, la frecuencia de tales actualizaciones inevitablemente aumentará también. Por consiguiente, la protección preventiva evolucionará en una variante del método de firmas, aunque basada más en el comportamiento que en los patrones de código. A pesar de las limitaciones, los métodos preventivos sí detectan algunas amenazas antes de que las relevantes firmas sean distribuidas. Se debe notar que las altas tasas de detección demostradas por los analizadores heurísticos tienen una desventaja: las tasas de falsos positivos son altas también. Para operar normalmente, un programa antivirus debe conseguir un equilibrio entre las tasas de detección y las de falsos positivos.

El resultado de los análisis dirigidos por AV-comparatives.org y AV-Test.org provee una sólida ilustración del hecho de que los métodos preventivos aisladamente son incapaces de proveer las tasas de detección necesarias. Las compañías de antivirus están perfectamente conscientes de esto, no obstante a toda la retórica acerca de las tecnologías preventivas, continúan usando los métodos clásicos de detección basada en firmas en sus soluciones. Naturalmente, los métodos basados en firmas tienen sus deficiencias también, pero hasta ahora, la industria de los antivirus ha sido incapaz de producir nada que pueda reemplazar estos métodos clásicos. Consecuentemente, el criterio primario para medir la efectividad de las soluciones de antivirus continuará incluyendo no solo la calidad de la protección preventiva, sino también tiempo de respuesta a nuevas amenazas de virus (el tiempo que lleva obtener la firma y añadirla a la base de datos, y distribuir actualizaciones para los usuarios).

En resumen, un número importante de conclusiones pueden ser hechas a partir de lo antes mencionado. Primero que todo, el método preventivo para combatir programas malignos es la respuesta de la industria de los antivirus a la cada vez mayor corriente de nuevos programas malignos y las crecientes tasas a las que se esparcen. Los métodos preventivos existentes son sin duda de gran ayuda en el combate de varias nuevas amenazas, pero la idea de que las tecnologías preventivas pueden reemplazar las actualizaciones regulares a la protección de los antivirus es una falacia. En realidad, los métodos preventivos requieren de actualizaciones tanto como los métodos basados en firmas, aunque en menor grado. Las técnicas preventivas existentes por si solas no pueden asegurar altas tasas de detección de programas malignos. Además de esto, las tasas altas de detección están acompañadas de

altas tasas de falsos positivos. En esta situación, el tiempo de respuesta a las nuevas amenazas sigue siendo una medida sólida de la efectividad de los programas antivirus. Para una protección óptima, los métodos preventivos y los basados en firmas deben ser usados juntos, las tasas más altas de detección solo pueden ser alcanzadas combinando estos dos métodos. (4)

Sería imprudente poner demasiada fe en la protección basada en comportamiento, a menudo llamada heurística. El análisis heurístico tiene sus propios inconvenientes. Algunas pruebas de la organización llamada AV-Comparatives (www.av-comparatives.org) parecen indicar que ningún método protege mejor que el antiguo método basado en firmas periódicamente actualizadas.

En general el análisis heurístico por sí mismo no parece ser tan confiable como el método basado en firmas. La efectividad de un producto antivirus particular depende de la configuración con la cual opere. Es decir, en caso de incrementar las tasas de detección es probable que ocurra al costo de un incremento de falsos positivos o bajo rendimiento. Como en muchos casos en lo referente a la seguridad, es un trueque.

El análisis heurístico provee un nivel adicional de defensa contra los ataques día-cero, pero no es un método del cual se pueda depender exclusivamente. Las actualizaciones de firmas son la primera y mejor línea de defensa. (2)

1.3 Otros métodos

Todos los sistemas operativos trabajan para mejorar su seguridad con el transcurso del tiempo. Las aplicaciones de protección de tipo servidor y desktop también tratan de mitigar las vulnerabilidades día-cero. Generalmente estas tecnologías involucran análisis heurísticos de terminación, es decir, el análisis de un programa que intenta determinar si la evaluación de una expresión dada puede terminar definitivamente. Algunos sugieren que una solución de este tipo puede estar fuera de alcance porque es algorítmicamente imposible, en el caso general, analizar cualquier código arbitrario y poder determinar si es maligno, ya que tales análisis se reducen al problema halting sobre un autómata linealmente acotado, el cual irresoluble. Sin embargo, en la mayoría de los casos, no es necesario centrarse en el caso general (clasificar todos los programas en las categorías maligno y no maligno) para eliminar una amplia variedad de comportamientos malignos. Según se plantea es suficiente con reconocer la seguridad de un conjunto limitado de programas (ej. aquellos que pueden acceder o modificar solo un dado subconjunto de recursos

del sistema) mientras se rechazan algunos programas seguros y no seguros. Esto requiere mantener la integridad de los programas reconocidos como seguros, lo cual puede probar cierta dificultad ante un exploit a nivel de kernel[2]. Programas como la tecnología SONAR de Symantec intentan identificar software no maligno usando un algoritmo que detecta rasgos de programas no malignos conocidos. Cualquier nuevo programa instalado que no cumpla con los requisitos del algoritmo es marcado como potencial maligno.

El problema con este método es que no sería muy popular entre usuarios ya que no permitiría hacer mucho en un sistema fuera del conjunto de programas clasificados como seguros, o dependería del usuario para evaluar un programa al igual que los bloqueadores de comportamiento. Por lo general los usuarios prefieren trabajar con más libertad, incluso en detrimento de la seguridad, que en un sistema estrictamente restringido.

1.4 Resumen

Dado el problema del incremento constante de programas malignos las investigaciones se han dirigido hacia la búsqueda de métodos alternativos como los métodos preventivos, los que posiblemente evolucionen en una variante del método basado en firmas, pero centrado más en el comportamiento de los programas que en los patrones de código.

Aunque los métodos basados en firmas tienen desventajas, no se ha encontrado un método que los sustituya, ya que los métodos preventivos aisladamente son incapaces de proveer las tasas de detección necesarias para prescindir de ellos. Por tanto para una protección óptima, los métodos preventivos y los basados en firmas deben ser usados juntos y ambos deben ser actualizados periódicamente.

[2] El kernel o núcleo es el software responsable de facilitar a los distintos programas acceso seguro al hardware de la computadora o en forma mas básica, es el encargado de gestionar recursos, a través de servicios de llamada al sistema.

CAPÍTULO 2. OFUSCAMIENTO DE CÓDIGO MALIGNO

Los escritores de virus se están haciendo más astutos cada día. Están viniendo con nuevas e innovadoras vías para evadir la detección por firmas de los antivirus. Una técnica de evasión usada por virus polimórficos y metamórficos es su habilidad para cambiar código de tal modo que falle la detección basada en firmas. Estos virus cambian de forma tal que cada nuevo fichero infectado tiene cadenas de caracteres diferentes, haciendo que la detección de firmas basada en cadenas de caracteres sea prácticamente inservible ante tales virus. (5)

El ofuscamiento de código no es más que hacer que el código sea ininteligible o al menos difícil de entender. Este proceso consiste en aplicar transformaciones que cambien la apariencia física del código sin que cambie la funcionalidad del programa. El ofuscamiento no solo es usado para proteger la propiedad intelectual sino también para evitar la detección de código maligno, muchas veces cambiando constantemente la firma del código para evadir a los programas analizadores de virus.

2.1 Técnicas de Ofuscamiento

2.1.1 Inserción de basura

También conocida como inserción de código no funcional, consiste en la inserción en un punto particular del programa, de un conjunto de instrucciones válidas las cuales no alteran el comportamiento esperado del programa. Por ejemplo, dada la siguiente secuencia de instrucciones a = b / d, b = a * 2; cualquier instrucción que modifique a b, puede ser insertada entre la primera y la segunda instrucción; además, las instrucciones que reasignan cualquier otra variable sin cambiar su valor realmente pueden ser insertadas en cualquier punto del programa (ej. a = a + 0, b = b * 1,…). (6) En algunos casos las instrucciones insertadas no tienen que ser necesariamente válidas.

El ejemplo más simple es la inserción de instrucciones nop en el caso de los ejecutables. Ofuscaciones más interesantes incluyen secuencias de código desafiantes que modifican el estado del programa, pero solo para restaurarlo inmediatamente. Otra variante pudiera ser insertar saltos de modo que las nuevas instrucciones no se ejecuten. El análisis del código puede detectar ofuscamientos simples como inserciones de instrucciones nop, usando expresiones regulares en lugar de firmas

predeterminadas. Para capturar inserciones de instrucciones nop la firma debe permitir cualquier número de instrucciones nop en los extremos de cada instrucción.

El código basura no hace el programa más complejo, y en dependencia de cómo se realice el algoritmo de detección, no hay diferencia entre las instrucciones nop y códigos basura más complejos.

2.1.2 Transposición de código

La transposición de código desordena las instrucciones de modo que el orden en la imagen binaria sea diferente del orden de ejecución, o del orden asumido en las instrucciones de la firma usada por el antivirus.

La mayoría de las herramientas automáticas de análisis usan una representación intermedia, tal como la gráfica de flujo de control (CFG) o la gráfica de dependencia del programa (PDG), que no es sensible a los cambios superfluos en el flujo de control. (3)

El orden de las instrucciones, así como la estructura del programa, es alterado introduciendo falsos saltos condicionales e incondicionales de forma tal que en tiempo de ejecución el orden en que son ejecutadas no es modificado. Además, las llamadas a función y los saltos directos pueden ser traducidos en indirectos cuyas direcciones de memoria son camuflajeadas en otras instrucciones a fin de evitar una reconstrucción exacta del flujo de control.

También se hace uso de instrucciones independientes, ej. Instrucciones cuyo procesamiento no depende del resultado de la instrucción previa. Éstas son arbitrariamente permutadas sin alterar la semántica del programa. Por ejemplo, las tres declaraciones a = b * c, d = b + e y c = b & c pueden ser ejecutadas en cualquier orden, siempre y cuando el uso de la variable c preceda su nueva definición. (6)

2.1.3 Reasignación de registros

Las transformaciones de reasignación de registros reemplazan el uso de un registro con otro en un campo de acción específico. Por ejemplo, si el registro ebx está completamente fuera dado el campo de acción de el registro eax, puede sustituir eax en ese campo de acción. En ciertos casos, las reasignaciones de registros requieren de inserción de código prólogo y epílogo alrededor del campo de acción para restablecer el estado de varios registros. (3)

En general el uso de una variable (registro, dirección de memoria o elemento de pila) es cambiada por otra variable perteneciente a un conjunto de candidatos válidos, preservando el comportamiento del programa. (6)

2.1.4 Sustitución de instrucciones

Esta técnica de ofuscación usa un diccionario de secuencias de instrucciones equivalentes para reemplazar una secuencia de instrucciones con otra. Ya que estas transformaciones están basadas en el conocimiento humano de las instrucciones equivalentes, representan un reto difícil para la detección automática de código maligno.

Para tratar con ofuscación basada en sustitución de instrucciones, una herramienta de análisis debe mantener un diccionario de secuencias de instrucciones equivalentes, similar al diccionario usado para generarlas. Esta no es una solución integral pero puede lidiar con los casos comunes. En el caso de IA-32[3], el problema puede ser ligeramente simplificado usando un lenguaje intermedio simple que descompone las operaciones complejas correspondientes a cada instrucción IA-32. (3)

Una secuencia de instrucciones es asociada a un conjunto de secuencias alternativas de instrucciones las cuales son semánticamente equivalentes a la original. Cada ocurrencia de la secuencia original puede ser reemplazada por un elemento arbitrario de este conjunto. Por ejemplo, en lo que respecta a los registros eax y ebx, los siguientes fragmentos de código son equivalentes. (6)

Instrucciones	**Formas equivalentes**
mov **eax**, **ecx**	xor **ebx**, **eax**
mov **ebx**, **eax**	xor **eax**, **ebx**
mov **ecx**, **ebx**	xor **ebx**, **eax**

[3] **IA32** es la arquitectura de microprocesadores de 32 bits de Intel (*Intel Architecture* 32). Son los microprocesadores más usados en los ordenadores personales (PC).

2.2 Código polimórfico

Código polimórfico es código que muta manteniendo el algoritmo original intacto. Esta técnica es usada a veces por virus, shellcodes[4] y gusanos para ocultar su presencia.

La mayoría de los antivirus y sistemas de detección de intrusos tratan de localizar el código maligno buscando en los archivos de la computadora y en los paquetes enviados por la red. Estos usan análisis sofisticados de patrones para encontrar patrones subyacentes dentro de las diferentes mutaciones del motor de desencriptamiento, esperando detectar tales programas malignos fiablemente.

Un virus polimórfico usa múltiples técnicas para prevenir el análisis por firmas. Primero, el código del virus es encriptado, y solo una pequeña rutina en claro es designada para desencriptar el código antes de ejecutar el virus. Cuando un virus polimórfico se replica él mismo infectando otro programa, encripta el cuerpo del virus con una nueva clave generada, y cambia la rutina de desencriptar generando nuevo código para esta. Para ofuscar la rutina de desencripción, son aplicadas varias transformaciones a esta. Éstas incluyen: inserción de instrucciones nop, transposición de código (cambiando el orden de las instrucciones y colocando instrucciones jump para mantener la semántica original), y reasignaciones de registros (combinando asignaciones de registros). Estas transformaciones cambian efectivamente la firma de los virus, inhibiendo la detección por el scanner de un antivirus. (3)

2.3 Código metamórfico

En términos de virus informático, código metamórfico es el que puede reprogramarse a sí mismo. A menudo, lo hace convirtiendo su propio código en una representación temporal, y luego lo convierte de vuelta al código normal. Esto es usado en algunos virus cuando están a punto de infectar nuevos archivos, y el resultado es que los “hijos” nunca se parecerán a sus padres. Los virus informáticos que usan esta técnica hacen esto para evitar el reconocimiento de patrones de los antivirus: el algoritmo actual no cambia, pero todo lo demás pudiera cambiar.

[4] Un shellcode es un pedazo ejecutable de código de máquina que es utilizado como payload o carga en el aprovechamiento de un error de software o bug de un programa. Los shellcode pueden ser cargados en el espacio de memoria de otro programa y ser ejecutados subsecuentemente.

El código metamórfico es más efectivo que el código polimórfico. Esto es porque los antivirus tratarán de encontrar código de virus conocidos aún durante la ejecución del código. Los virus metamórficos, al igual que los polimórficos, encriptan su código pero a diferencia de estos cambian su código usando técnicas de ofuscamiento. Además el código de los polimórficos aparece desencriptado en memoria durante su ejecución mientras que en los metamórficos el código nunca es revelado completamente de una sola vez.

2.4 Resumen

Las técnicas de ofuscamiento de código como inserción de basura, transposición de código, reasignación de registros y sustitución de instrucciones están dirigidas fundamentalmente a evitar que los programas malignos sean detectados por los métodos tradicionales basados en firmas y a los métodos basados en análisis de código en general. Los programas malignos más recientes usan polimorfismo y metamorfismo. Aunque la mayoría de estas técnicas no le hacen frente a las técnicas de ingeniería inversa, representan un reto incluso para los métodos preventivos ya que las reglas heurísticas utilizadas deben ser capaces de agrupar mucho más instrucciones en el caso de análisis estático. También se hace más complicado el proceso de desinfección de archivos.

CAPÍTULO 3. SCRIPTS MALIGNOS EN FICHEROS BATCH

Las técnicas tratadas anteriormente son relativas a los virus en general y principalmente a los virus que hacen uso de ficheros binarios ejecutables para realizar su acción. Algunas de estas técnicas son usadas por los virus scripts tales como la inserción de basura, la transposición de código y la sustitución de instrucciones. Los virus scripts a diferencia de los demás pueden ser analizados directamente pues son ficheros texto que presentan el código fuente en un lenguaje interpretado[5], y generalmente está en claro, por lo que no es necesario utilizar ingeniería inversa ni ningún método para desensamblar el código como en el caso de los programas compilados o pre-compilados, y no siempre se requiere el uso de emuladores.

3.1 Scripts malignos

Se pueden encontrar varios tipos de programas malignos que usan lenguajes script, entre ellos están virus, gusanos, troyanos, spoofers, entre otros. Es común que varias de estas modalidades se encuentren en un mismo programa. Los virus script son una sub-categoría escrita en una variedad de lenguajes de script (VBS, JavaScript, BAT, PHP etc.). Ellos infectan otros scripts, por ejemplo, archivos de instrucciones o servicios de Windows o Linux, o bien son parte de virus de varios componentes. Los virus de script pueden infectar otros formatos de archivo, como HTML, si el formato permite la ejecución de scripts. (7)

Los virus script (a veces llamados virus macros) generalmente viajan insertados en correos y en documentos de ofimática, aunque pueden ser encontrados también en páginas Web. Los virus tradicionales son usualmente implementados en código ejecutable del sistema, mientras que los virus scripts son usualmente escritos en un potente lenguaje de alto nivel que es compilado y ejecutado al mismo tiempo sobre la marcha. A menudo tienen funcionalidad sofisticada e interfaces directas con aplicaciones de alto nivel tales como procesadores de texto, hojas de cálculo, correo electrónico y programas Web, y pueden ocasionar daños considerables. También son llamados macro virus ya que inicialmente cubrían los programas de ofimática. Los virus script también pueden propagarse mediante los protocolos IRC. (8)

[5] Los lenguajes interpretados son los que se ejecutan instrucción por instrucción sin ninguna traducción previa del código fuente; a diferencia de los lenguajes compilados, donde el programa se traduce a partir de su código fuente por medio de un compilador en un archivo ejecutable para una determinada plataforma.

3.1.1 Patrones generales de scripts malignos

Estas son algunos casos sospechosos que pueden activar banderas heurísticas:

- Uso de directorios especiales (Windows, Windows\system, Windows\temp).
- Creación de archivos en los directorios WINDOWS o WINDOWS\SYSTEM.
- Creación de uno o más archivos con el mismo tamaño que el analizado.
- Uso de Outlook u otras aplicaciones.
- Obtención del número de líneas (*CountOfLines*) del archivo analizado indica que el *script* se puede copiar (o incluso mutar) hacia otros lugares.
- La creación del archivo SCRIPT.INI puede indicar que el virus también tiene *scripts* de MIRC.
- Código que produce el envío de correo electrónico con el archivo anexado.
- Búsqueda de recursos de red.
- Escribir en el registro. (1)

A continuación se muestran ejemplos de patrones de scripts malignos, en este caso de código del lenguaje VBScript.

```
Set fso = CreateObject(“Scripting.FileSystemObject”)
Set dirwin = fso.GetSpecialFolder(0)
Set dirsystem = fso.GetSpecialFolder(1)
Set dirtemp = fso.GetSpecialFolder(2)
Set c = fso.GetFile(Wscript.ScriptFullName)
c.Copy(dirsystem&”\KERNEL32.DLL.VBS”)
set out=WScript.CreateObject(“Outlook.Application”)
set mapi=out.GetNameSpace(“MAPI”)
for ctrlists=1 to mapi.AddressLists.Count
for ctrentries=1 to a.AddressEntries.Count
set a=mapi.AddressLists(ctrlists)
malead=a.AddressEntries(x)
set male=out.CreateItem(0)
male.Recipients.Add(malead)
Obj.CodeModule.CountOfLines
```

- `Application.ScreenUpdating = 0`
- `Application.EnableCancelKey = wdCalcelDisabled`
- `Application.DisplayAlerts = wdAlertsNone`
- `Set b = fso.CreateTextFile("c:\ircap\events.ini", True)`
- `Options.VirusProtection = False`
- `Options.ConfirmConversions = False`
- `Options.SaveNormalPrompt = False`
- `WSHShell.Run ("C:\Some.URL")`
- `wo.NormalTemplate.VBProject.VBComponents.Item(1).Codemodule.AddFromFile c:\filename.ext"`
- `Mirc=fso.CreateTextFile("c:\mirc\scrit.ini")`
- `HMMXS.Attachments.Add Wscript.ScriptFullName`
- `HMMXS.DeleteAfterSubmit = True`
- `HMMXS.Send`
- `Set HMMXS = WG.48.EnumNetworkDrives`
- `Regedit.RegWrite "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices\WinLoader", "c:\windows\system\Kernel32.DLL.vbs" (1)`

Dado que los scripts malignos son a menudo combinaciones de varios tipos, por ejemplo, un mismo script puede ser un gusano que realiza también acciones de un caballo de Troya o una puerta trasera, se presentan también patrones generales de otros tipos de programas malignos. Algunos troyanos y gusanos no infectan otros archivos, pero ellos necesitan ejecutarse automáticamente al inicio de Windows, y lo logran modificando las llaves del registro. Un buen analizador heurístico debe investigar algunas de estas llaves:

- `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`
- `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce`
- `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnceEx`
- `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices`
- `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServicesOnce`
- `HKEY_CLASSES_ROOT\exefile\shell\open\command`

También pueden instalarse modificando los archivos de configuración del sistema:

- El comando ‘run’ en la sección [windows] del archivo WIN.INI
- El comando ‘shell’ del archivo SYSTEM.INI (1)

3.2 Acciones comunes de los scripts malignos en ficheros batch

Los ficheros batch no son más que ficheros texto sin un formato determinado que contienen un conjunto de instrucciones formadas por los comandos del sistema que se ejecutan secuencialmente. Además permiten correr una serie de ejecutables de forma inmediata. Generalmente utilizan para esto el intérprete de comandos CMD.EXE que es un programa separado que provee comunicación directa entre el usuario y el sistema operativo. Los programas batch les permiten a los usuarios de MS-DOS y Microsoft Windows automatizar tareas habituales o procesos muy tediosos y sus ficheros tienen extensión .cmd o .bat generalmente. Se puede incluir cualquier comando en el fichero batch. Ciertos comandos, tales como **if**, **for**, y **goto**, permiten procesar condicionalmente los comandos en el fichero. Otros comandos permiten controlar la entrada y salida de datos y llamar a otros ficheros batch. Incluso algunos pueden realizar operaciones delicadas que pueden comprometer el funcionamiento del sistema. Aunque estos programas no son adecuados para la programación de propósito general, a menudo son utilizados con fines malignos.

Los scripts malignos realizan varias acciones triviales u objetivas según la intención del que escribe el mismo, como por ejemplo permitirle al autor acceso a varios sistemas o simplemente diseminarse en las redes. Pero además de su payload o carga, estos programas pueden realizar acciones que de cierto modo se repiten en los nuevos programas malignos que surgen o en las nuevas versiones de los ya conocidos.

Algunos scripts malignos tratan de eliminar o modificar ficheros, generalmente ejecutables o librerías de enlace dinámico, de antivirus y otros programas de seguridad conocidos de modo que queden inutilizables, disminuyendo así la probabilidad de ser detectados, en dependencia de los programas de seguridad especificados en el script y la cantidad de personas que los utilicen. Ej.

```
del C:\progra~1\norton~1\*.exe /q
```

Muchas veces es necesario utilizar scripts en otros lenguajes como el VBS en el caso de los gusanos de correo electrónico para propagarse enviando una copia del programa en un correo dirigido todos en la lista de contactos del usuario, o los comandos del IRC para propagarse utilizando los canales de este protocolo de comunicación en tiempo real basado en texto. Para esto se crean ficheros en otros lenguajes y se escribe en ellos línea por línea utilizando el comando **echo** y operadores de re-direccionamiento '**>**' y '**>>**'. Ej.

```
echo [script] >  c:\mirc\script.ini
echo n0=on 1:JOIN:#:{ >> c:\mirc\script.ini
...
echo n3=} >> c:\mirc\script.ini
```

Una forma de un programa maligno de asegurar su permanencia es copiándose directamente en el sistema creando carpetas ocultas, sobrescribiendo archivos del sistema, etc. También se copian directamente en las carpetas de inicio o modificando el registro haciendo que el programa maligno se ejecute cuando se inicie el sistema. Ej.

```
echo REGEDIT4 > c:\X.reg
echo[HKEY_LOCAL_MACHINE\...\Windows\CurrentVersion\Run] >> c:\X.reg
...
...
regedit /s c:\X.reg
del c:\X.reg
```

Otras acciones, que aunque no son evidentemente malignas o que pueden ser utilizadas en programas no malignos o que incluso no se usan en los sistemas operativos más recientes, también se deben tener en cuenta por su índice de aparición en scripts malignos como por ejemplo desactivar la salida de la consola y re-direccionar la consola a nul para evitar interrupciones por parte del usuario.

Las acciones en el caso de los troyanos por lo general consisten en inutilizar el sistema o hacer que algunas funciones del mismo queden inservibles… para esto eliminan, cambian nombres o extensiones de ficheros, modifican el contenido de los mismos sobrescribiendo o agregando información en archivos de inicio, por ejemplo. Muchas veces su payload o carga (a menudo lo único que tienen)

consiste en destruir información borrando archivos aleatoriamente, formateando unidades de disco, o en hacer desorden creando directorios de forma masiva en cualquier parte del disco.

Muchos de estos programas están constituidos por unas pocas instrucciones, lo que hace más difícil su detección por métodos heurísticos de análisis estático porque la suma de los pesos de los patrones que se identifiquen puede no alcanzar el umbral de decisión definido, por lo que es necesario hacer reconocedores de patrones más específicos para los comandos utilizados en este tipo de programas malignos y dar más peso a los mismos en la decisión.

3.3 Ofuscamiento de código en scripts batch

Los scripts malignos usan comandos corruptos, operaciones superfluas o textos no válidos para dificultar el análisis, ya que estos hacen más difícil la detección por métodos heurísticos de análisis estático o por los métodos basados en firmas. Ej.

```
set var=s
set var=d
goto var
set var=s
:var
...
%var%ebug < file
```

En el ejemplo anterior se hacen varias declaraciones **set** innecesarias para una misma variable de entorno. Muchas veces simplemente se agregan en cada línea, por ejemplo al principio o al final de cada una, variables de entorno cuyo valor son caracteres triviales como el de espacio o tab. Ej.

```
%space%@ren bat enrg.bat%space%
%space%@deltree/y bat >nul%space%
%space%@deltree/y enrg >nul%space%
%space%@enrg.bat%space%
```

En el caso de los virus ejecutables se tiene más cuidado con la inserción de código basura porque un determinado error puede hacer que la ejecución del programa infestado sea terminada o interrumpida, pero con los scripts no ocurre lo mismo ya que estos no son compilados y se ejecutan línea por línea; de

modo que si ocurre un error en alguna instrucción, las demás se siguen ejecutando. Esto solo afectaría el funcionamiento del programa si las instrucciones posteriores dependieran de las anteriores y aún así se llegarían a ejecutar todas aunque no se lograra el objetivo del script. Por consiguiente, en un script arbitrario se puede insertar cualquier secuencia de caracteres sin que se afecte el funcionamiento del mismo siempre y cuando que no se modifiquen sus instrucciones. Ej.

```
uÿMüˆ …øþÿÿSPÿ°@ …Àt°éÓ
€½øþÿÿ\u"€½ùþÿÿ\u …úþÿÿj\PèE'
echo off
å÷ ¡a}ØÚS¼q ®<-90ßžÉnØÞ&Ë¬Ü\
copy %0 C:\prob.bat
...
```

Un método efectivo para evadir el análisis estático o basado en firmas es el encriptamiento del código. Existen scripts malignos como el BAT.Calhob que usan otros programas para encriptar su código, éstos se mantienen encriptados y se desencriptan solamente cuando se ejecutan y luego se vuelven a encriptar. También se realiza utilizando variables de entorno para sustituir una letra o parte del código. Ej.

```
set piece=ipt
cscr%piece% file.vbs
```

Muchas veces se crea un alfabeto y se convierte prácticamente todo el código del programa de modo que se hace difícil análisis del código, ya que el script se ejecuta sin decodificarse completamente, a diferencia de los scripts que utilizan programas para encriptarse, y sus códigos completos aparecen en claro al menos en el momento de ejecutarse o cuando son cargados en memoria. Ej.

```
@%sm%%ak%%aq%%kk% %kk%%ww%%ww%
%ak%%ec%%ec%%nx% %qf%%xj%%uw%
%ec%%to%%qz%%id%%id%%ut%%uw%%uw% /%ww% /%ut%%bn% WFINDV32.EXE
%ec%%to%%qz%%id%%id%%ut%%uw%%uw% /%ww% /%ut%%bn% WEBSCANX.EXE
%ec%%to%%qz%%id%%id%%ut%%uw%%uw% /%ww% /%ut%%bn% VSSTAT.EXE
```

Para detectar este tipo de programa maligno habría que realizar un análisis dinámico o tener previamente la firma específica para ese programa. Otra forma de frustrar la detección por los métodos

convencionales basados en firmas es transposición de código, es decir, cambiar el orden en que está escrito, sin cambiar el orden en que se ejecutan las instrucciones. Con el comando **goto** y el uso de etiquetas se puede modificar el orden en que se ejecutan las instrucciones haciendo un salto incondicional hasta la etiqueta indicada. Se pueden hacer saltos condicionales utilizando el comando **if** junto con **goto**. El comando **call** también se puede utilizar para hacer saltos. Ej.

```
goto payload
...
:spread
...
goto end
:payload
...
goto spread
:end
...
```

Otro modo de cambiar el orden de ejecución de un programa es marcar cada instrucción del fichero utilizando variables de entorno de modo que las líneas de código de cada bloque del programa estén marcadas con una variable diferente. Esta variante es usada cuando un programa va a difundirse, creando un fichero y copiando cada bloque utilizando el comando **find**, de modo que los bloques pueden copiarse en el nuevo fichero en un orden diferente al anterior sin afectar la lógica del programa. Ej.

```
if [condición] goto 1 %AAAA%
if [condición] goto 2 %AAAA%
:1              %AAAA%
find "AAAA"<%0 >> C:\my.bat %AAAA%
find "CCCC"<%0 >> C:\my.bat %AAAA%
find "DDDD"<%0 >> C:\my.bat %AAAA%
find "BBBB"<%0 >> C:\my.bat %AAAA%
goto code       %AAAA%
:2              %AAAA%
find "AAAA"<%0 >> C:\my.bat %AAAA%
find "DDDD"<%0 >> C:\my.bat %AAAA%
find "BBBB"<%0 >> C:\my.bat %AAAA%
find "CCCC"<%0 >> C:\my.bat %AAAA%
goto code       %AAAA%
...             %AAAA%
:code           %AAAA%
REM Mass mailer %BBBB%
```

```
...             %BBBB%
REM Payload     %CCCC%
...             %CCCC%
REM Registry chg     %DDDD%
...             %DDDD%
```

Es usual ver en scripts malignos el uso de código en otro lenguaje que se escribe en un fichero y se ejecuta con el comando **start** o **cscript**. De hecho los gusanos de correo o los flooder (programas que envían el mismo mensaje o texto de forma reiterada y masiva, pretendiendo así producir un efecto de saturación, colapso o inundación) en ficheros batch crean un fichero script en VBS y luego ejecutan dicho fichero. Con el código en VBS se crea el correo, se adjunta una copia del programa maligno u otros ficheros, se busca direcciones en las listas de contactos y se envía a todos los contactos, generalmente usando Microsoft Outlook. Ej.

```
...
echo set a = Wscript.CreateObject("Wscript.Shell") >> c:\X.vbs
echo set b = CreateObject("Outlook.Application") >> c:\X.vbs
echo for y = 1 To c.AddressLists.Count >> c:\X.vbs
...
echo e.Attachments.Add ("c:\virus_copy.bat") >> c:\X.vbs
echo e.Send >> c:\X.vbs
echo next >> c:\X.vbs
start c:\X.vbs
```

También se crear ficheros binarios escribiendo en lenguaje de máquina y luego depurando el fichero creado para obtener el código script. Ej.

```
echo e 0170  65 20 6B 65 79 2C 64 69 73 61 62>>temp
echo e 0180  6C 65 25 64 76 6C 72 67 25 63 74>>temp
echo e 0190  74 79 20 63 6F 6E 0D 0A 25 6E 72>>temp
echo e 01A0  67 25 63 6C 73 25 64 76 6C 25 00>>temp
echo rcx>>temp
echo AA>>temp
echo n file.bat>>temp
echo w>>temp
echo q>>temp
debug < temp
```

3.4 Patrones

Las técnicas explicadas anteriormente fueron obtenidas a partir del estudio del código de varios programas malignos en ficheros batch para obtener patrones que permitan reconocerlos. Se analizaron de forma manual alrededor de 600 scripts malignos para estudiar sus acciones más comunes y registrar instrucciones claves en su comportamiento o que son muy utilizadas en este tipo de programas. Para extraer los patrones se tuvo en cuenta las características de los ficheros batch, las posibles acciones de los comandos del sistema en general, la frecuencia de aparición de algunos comandos en este tipo de programas.

Por ejemplo, en el caso de los comandos, los que realizan acciones como eliminar archivos, formatear unidades, administrar usuarios, acceder y modificar el registro del sistema se consideraron como de mayor peligrosidad, de modo que si un programa presenta alguna de estas instrucciones entonces tiene una alta probabilidad de ser maligno, y por tanto tienen mayor peso en la decisión de si un script es maligno o no. Pero no solo por las determinadas acciones que realiza un comando han sido identificados los patrones sino también por la frecuencia con que son usados en los programas malignos. Hay varias instrucciones que se repiten en muchos scripts malignos pero no son imprescindibles ya que no realizan ningún cambio significativo y por lo general no representan ningún comportamiento evidentemente maligno como en el caso de los comandos cuya función es solamente mostrar información. Este tipo de instrucciones fueron utilizadas también para identificar programas malignos pero con un peso menor según las operaciones que realizan. De la misma manera, los scripts que están formados por pocas instrucciones (algunos solamente tienen una) pueden no alcanzar el umbral de decisión debido a que en ellos se reconocen pocos patrones. Los patrones más comunes en este tipo de scripts tienen mayor peso como por ejemplo el que incluye el comando **format**, que está presente en varios troyanos.

Por otro lado, no se tuvo en cuenta otros tipos de lenguajes scripts como VBScript o los comandos del IRC que son frecuentemente utilizados en este tipo de programas malignos aunque no de forma directa sino creando un fichero con el código correspondiente y ejecutándolo. Es decir, los patrones obtenidos no reconocen otros tipos de script. No obstante en algunos patrones pueden haber involucrados otros lenguajes scripts con el objetivo de evitar ambigüedades. Por ejemplo el comando **set** puede ser confundido con una declaración **set** de VBScript, ya que tienen sintaxis muy parecidas. En la siguiente tabla se muestra un ejemplo de análisis de un script maligno.

Email-Worm.BAT.Alcobul.a	Patrones
se propaga escribiendo en un fichero .ini del mIRC	`echo…>>…`
crea un directorio oculto	`md, attrib +h +r`
se copia él mismo un fichero dentro del directorio	`copy %0`
salto condicional	`if exist… goto`
modifica el registro y hace que el script se ejecute al inicio del sistema, escribiendo en un fichero y ejecutándolo	`echo…>>…, start`
intenta sobrescribir los archivos .dat del antivirus McAfee	`for… do copy %0`
crea un fichero .vbs (mass-mailer) escribiendo en él, luego lo ejecuta	`echo…>>…, start`
oculta el archivo creado	`attrib +h +r`

Se debe aclarar que aunque varios de los patrones son simplemente comandos, la mayoría de ellos están formados por un comando asociado a ciertos operadores o parámetros, incluso algunos contienen más de un comando. Así mismo hay comandos que están incluidos en más de un patrón. Ej.

Patrón general: `DEL|ERASE...`

Patrón específico: `DEL|ERASE...WINDOWS|WINDIR|SYSTEM|AUTOEXEC`

Esto ocurre porque algunos patrones, que son más específicos, identifican una operación con una alta probabilidad de ser maligna pero al mismo tiempo ese comando en un patrón no tan específico puede contribuir a clasificar un script pero en menor grado, por ejemplo con un peso menor. Esto implica que si en un programa se encuentra el patrón específico de un comando determinado entonces se encontrará también el patrón menos específico o más general del mismo. Aunque esto pudiera parecer un problema no es así porque los patrones repetidos se hacen con el objetivo de identificar una operación más peligrosa con un mayor peso. Es decir, que en el caso que ambos patrones sean reconocidos en una misma instrucción, el patrón más general simplemente contribuirá a aumentar el grado de peligrosidad del programa que es en principio lo que se pretende. Si solamente se utilizara el patrón más general asociado a un comando, se identificarían ambas, las operaciones evidentemente malignas y las no tan malignas, pero contribuirían con un mismo grado en la clasificación de un programa y por tanto con menor exactitud. A continuación se muestran todos los patrones obtenidos del estudio de scripts malignos:

```
FORMAT...C:
ECHO|REN|COPY...AUTOEXEC
DEL|ERASE...WINDOWS|WINDIR|SYSTEM|AUTOEXEC|.(*|INI|SYS|EXE|COM|DLL)...
```

```
NET USER|LOCALGROUP.../ADD
COMMAND /F /C COPY A:\...
COPY %0...
PING...
TASKKILL|TSKILL...
RUNDLL USER|SHELL32|MOUSE|KEYBOARD...
REG ADD|IMPORT...
REGEDIT.../S...
ECHO OFF
ECHO...>...
START...
CSCRIPT...
WSCRIPT...
DEL|ERASE...
DEBUG <...
COPY...
ATTRIB...\+H...
RENAME...
FIND...>...
MD|MKDIR...
FOR...%...DO
CTTY NUL
NET STOP...
IF...EXIST...
IF...ERRORLEVEL...
SET...=...
```

3.5 Resumen

En resumen se puede decir que, al igual que los demás programas malignos, los scripts malignos en ficheros batch utilizan técnicas de ofuscamiento de código como inserción de basura, transposición de código, sustitución de instrucciones, y pueden existir también algunas formas de polimorfismo y metamorfismo. No obstante, dado que los programas interpretados no se compilan a lenguaje de máquina o algún lenguaje intermedio, es posible utilizar métodos de análisis de código para detectar scripts malignos siempre que no estén encriptados. Y si se realiza un análisis del código de varios de estos se pueden encontrar varios patrones que pudieran ser utilizados para identificar este tipo de programas utilizando métodos heurísticos de análisis estático.

CAPÍTULO 4. PRUEBA DE CONCEPTO

4.1 Expresiones regulares

Para verificar el reconocimiento de scripts potencialmente malignos en ficheros batch mediante los patrones obtenidos se realizaron varias pruebas usando un método heurístico de análisis estático. No se utilizó el análisis dinámico ya que el objetivo es reconocer programas potencialmente malignos a partir del análisis del código de los mismos y no de su comportamiento.

¿Por qué expresiones regulares y no análisis sintáctico?

Aunque es posible, no se utilizó un analizador sintáctico para el reconocimiento de patrones ya que los ficheros batch no tienen una estructura definida como en un programa en el lenguaje Pascal, que se divide en un encabezamiento y un bloque, y a su vez el bloque se subdivide en una parte declarativa y en secuencias de instrucciones enmarcadas por **begin** y **end**, por ejemplo. Y dado que los ficheros batch están constituidos por secuencias de instrucciones de los comandos del sistema que por lo general son independientes entre sí, es decir que pueden ejecutarse en cualquier orden, es mejor entonces reconocer los patrones utilizando expresiones regulares.

Boost Regex

El reconocimiento se hizo mediante expresiones regulares utilizando la librería Regex de Boost (www.boost.org). Boost provee de forma gratuita bibliotecas portables de C++, haciendo énfasis en las que trabajan bien con la Biblioteca Estándar de C++. Las librerías de Boost tienen como fin ser útiles y usadas en un amplio espectro de aplicaciones. La licencia de Boost estimula tanto el uso comercial como el no comercial. 10 librerías de Boost ya están incluidas en el reporte técnico del comité de estándares de C++ y muchas están propuestas para el próximo reporte. Boost-Regex está hecho con la intención de cumplir con la propuesta de estandarización de expresiones regulares, la cual aparecerá en un futuro reporte técnico de C++ estándar y es posible que en una versión futura de este estándar.

Boost Regex tiene varias funciones como regex_match que permiten validar expresiones regulares, pero como el objetivo es buscar si aparecen patrones de instrucciones y no reconocer completamente un programa maligno (como se haría en caso de utilizar el método basado en firmas), se utiliza la función regex_search, que busca ocurrencias de una expresión regular en un texto dado. Para

cada fichero script que se analiza se hace una llamada a la función regex_search con cada una de las expresiones de los patrones definidos.

Representación de los patrones

Se hizo una expresión para cada patrón obtenido del estudio de los scripts malignos de muestra. Para hacer las expresiones regulares no se representaron las sintaxis completas de los comandos involucrados en los patrones con el objetivo que no aumente mucho la complejidad computacional y para que sean suficientemente generales.

Para reconocer las expresiones en cualquier parte de un texto se necesitan al menos dos símbolos, uno que marque el comienzo de la expresión y otro el final. Aunque para algunos patrones se utiliza el símbolo de principio de línea ´^´ de forma alternativa, básicamente se busca un comando seguido de cualquier cadena de caracteres hasta encontrar el fin de línea, cuya representación en Regex es el símbolo ´$´. Pero para los patrones que están formados solamente por un comando seguido de cualquier cadena de caracteres (la ruta de un archivo, un nombre, etc.) se reconocen primero los símbolos que pueden aparecer delante del comando como por ejemplo los caracteres de espacio, los símbolos ´@´, ´\´ , ´|´, y el símbolo de principio de línea de Regex ´^´. También se reconocen los caracteres de espacio y los de los parámetros al final del nombre de un comando; todo esto se hace para evitar reconocer falsas instrucciones, por ejemplo, una palabra que contenga el nombre de un comando. Cuando hay símbolos indeterminados en una instrucción no se reconocen los argumentos o parámetros opcionales a menos que se requiera específicamente para reconocer un determinado patrón. De lo contrario el proceso de reconocimiento tendría una mayor complejidad. Para esto Regex utiliza el carácter ´.´ que reconoce cualquier símbolo, excepto el fin de línea ´$´ en este caso (especificado por directivas en Regex), y el operador ´*´ que significa que el carácter o grupo que aparece delante del mismo puede ser reconocido cero o varias veces. Así mismo, los operadores ´+´ y ´?´ reconocen la expresión precedente una o varias veces y cero o una vez respectivamente. Las secciones encerradas en paréntesis son sub-expresiones marcadas que tienen como efecto secundario un campo más en el resultado del análisis y las que están encerradas de la misma forma pero con los símbolos “?:” después del primer paréntesis son sub-expresiones no marcadas. A continuación se muestran las expresiones regulares de los patrones obtenidos.

```
"(FORMAT)(?:\\.COM)?(?:\\x20|\\t|/)(?:.*(?:\\x20|\\t))?[[:alpha:]]:.*$"
```

```
"(?:^|@|\\x20|\\t|\\|)(ECHO|REN(AME)?|COPY)(?:\\x20|\\t|/|\\.).*AUTOEXEC.*$"

"(?:^|@|\\x20|\\t|\\|)(DEL(TREE)?|ERASE)(?:\\t|\\x20|/).*(?:WINDOWS|WINDIR|SYS
TEM|AUTOEXEC|\\.(?:\\*|INI|SYS|EXE|COM|DLL)).*$"

"(DEL(TREE)?|ERASE)(?:\\t|\\x20|/).*(?:WINDOWS|WINDIR|SYSTEM|AUTOEXEC|\\.(?:\\
*|INI|SYS|EXE|COM|DLL)).*$"

"(NET)(?:\\.EXE)?(?:\\x20|\\t)+(?:USER|LOCALGROUP)(?:\\x20|\\t).+(?:/ADD).*$"

"(COMMAND)(?:\\.COM)?.+(?:/F).+(?:/C).+(?:COPY).+(?:A:\\\\)?.*$"

"(COPY)(?:\\x20|\\t|/).*(?:%0).+$"

"(?:^|@|\\x20|\\t|\\\\|\\|)(PING)(?:\\.EXE)?(?:\\x20|\\t).+$"

"(?:^|@|\\x20|\\t|\\\\|\\|)(TASKKILL|TSKILL)(?:\\.EXE)?(?:\\x20|\\t|/).+$"

"(RUNDLL)(?:32)?(?:\\.EXE)?(?:\\x20|\\t).*(?:USER|SHELL32|MOUSE|KEYBOARD).+$"

"(REG)(?:\\.EXE)?(?:\\t|\\x20)+(?:ADD|IMPORT)(?:\\x20|\\t).+$"

"(REGEDIT)(?:\\.EXE)?.*(?:/S).+$"

"(?:ECHO)(?:\\x20|\\t).*(OFF).*$"

"(ECHO)(?:\\x20|\\t|\\.).+(?:>).+$"

"(?:^|@|\\x20|\\t|\\|)(START)(?:\\x20|\\t|/).+$"

"(?:^|@|\\x20|\\t|\\\\|\\|)(CSCRIPT)(?:\\.EXE)?(?:\\x20|\\t|/).+$"

"(?:^|@|\\x20|\\t|\\\\|\\|)(WSCRIPT)(?:\\.EXE)?(?:\\x20|\\t|/).+$"

"(DEBUG)(?:\\.EXE)?(?:\\x20|\\t)*(?:<).+$"

"(?:^|@|\\x20|\\t|\\|)(MOVE)(?:\\x20|\\t|/).+$"

"(ATTRIB)(?:\\.EXE)?(?:\\x20|\\t).*(?:\\+H).*$"

"(?:^|@|\\x20|\\t|\\|)(REN)(AME)?(?:\\x20|\\t).+$"

"(FIND)(?:\\.EXE)?(?:\\x20|\\t|/).+(?:>).+$"

"(?:^|@|\\x20|\\t|\\|)(MD|MKDIR)(?:\\x20|\\t).+$"

"(FOR)(?:\\x20|\\t).*(?:%).+(?:DO).+$"
```

```
"(CTTY)(?:\\x20|\\t)+(?:NUL).*$"

"(IF)(?:\\t|\\x20)+(?:NOT)?(?:\\t|\\x20)*(?:ERRORLEVEL)(?:\\t|\\x20).+$"

"(?:^|@|\\x20|\\t|\\|)(DEL(TREE)?|ERASE)(?:\\t|\\x20|/).+$"

"(?:^|@|\\x20|\\t|\\|)(COPY)(?:\\x20|\\t|/).+$”

"(NET)(?:\\.EXE)?(?:\\x20|\\t)+(?:STOP)(?:\\x20|\\t).*$"
```

4.2 Pruebas

Primero se hizo un análisis de todos los scripts malignos y no malignos de muestra, para determinar el índice de ocurrencia de cada patrón. En las pruebas realizadas no se utilizaron todos los patrones mencionados en la sección 3.1.1 Patrones generales de scripts malignos, ya que estos pudieran ser válidos para cualquier tipo de lenguaje script y en este caso solo se estaban probando los patrones de los programas malignos en ficheros batch específicamente.

Se usaron como muestra 621 scripts malignos en ficheros batch y 418 scripts en ficheros batch no malignos de programas arbitrarios. Inicialmente se hizo una prueba para observar el índice de detección en general y de cada patrón. Como resultado se obtuvo mayores tasas de detección de positivos que de falsos positivos para la mayoría de los patrones y también en general. En la siguiente tabla se muestran los índices generales de detección de positivos y falsos positivos. Los scripts detectados son los programas en los que se reconoce al menos uno de los patrones definidos, y el total de scripts son los programas malignos y no malignos utilizados como muestra.

	Positivos	Falsos positivos	% positivos	% falsos positivos	Diferencia
Total de scripts	621	418			
Scripts detectados	607	336	97.74557	80.38278	17.3628

Para decidir si un patrón se puede utilizar o no, o en qué medida contribuye a identificar un programa maligno se tuvo en cuenta los porcentajes de scripts positivos y falsos positivos producidos por cada comando. La diferencia entre los índices de detección de positivos y falsos positivos determina si un

patrón puede ser utilizado o no y en qué medida contribuye a esto, es decir, qué peso tiene en la decisión. De modo que mientras mayor sea la diferencia, mayor peso tendrá a la hora de decidir si un programa es potencialmente maligno siempre que el índice de detección de positivos sea mayor que el de falsos positivos. En la siguiente tabla se muestran los índices de detección por cada patrón. Los patrones marcados con el símbolo (*), son los que involucran un comando que ya está incluido en otros; estos reconocen instrucciones más específicas, y aunque a veces no sean los más frecuentes, identifican código potencialmente maligno con mayor exactitud y con menor probabilidad de reconocer falsos positivos.

Patrones	Positivos	Falsos positivos	% positivos	% falsos positivos	Diferencia
FORMAT	125	0	20.12882	0	20.12882
ECHO, RENAME, COPY *	109	0	17.55233	0	17.55233
DEL,ERASE *	263	7	42.35105	1.674641	40.67641
NET *	5	0	0.805153	0	0.805153
COMMAND	7	0	1.127214	0	1.127214
COPY *	104	0	16.74718	0	16.74718
PING	10	0	1.610306	0	1.610306
TASKKILL	3	0	0.483092	0	0.483092
RUNDLL32	68	0	10.95008	0	10.95008
REG ADD	8	0	1.288245	0	1.288245
REGEDIT	27	0	4.347826	0	4.347826
ECHO OFF	468	84	75.36232	20.09569	55.26663
ECHO	248	31	39.93559	7.416268	32.51932
START	73	35	11.75523	8.373206	3.382028
CSCRIPT	24	0	3.864734	0	3.864734
WSCRIPT	5	0	0.805153	0	0.805153
DEBUG	27	0	4.347826	0	4.347826
MOVE	23	0	3.703704	0	3.703704
ATTRIB	46	0	7.407407	0	7.407407
REN	91	2	14.65378	0.478469	14.17532
FIND	43	1	6.924316	0.239234	6.685081
MD, MKDIR	97	15	15.61997	3.588517	12.03145
FOR	80	4	12.88245	0.956938	11.92551

CTTY	170	0	27.3752	0	27.3752
IF EXIST	127	137	20.45089	32.77512	-12.3242
IF ERRORLEVEL	59	28	9.500805	6.698565	2.802241
DEL,ERASE	337	31	54.26731	7.416268	46.85104
COPY	202	55	32.52818	13.15789	19.37029
NET	8	1	1.288245	0.239234	1.04901
SET	131	190	21.09501	45.45455	-24.3595

En este caso los patrones para los comandos **if** + **exist** y **set** tienen una influencia negativa. Esto significa que aun teniendo altos índices de detección positivos tienen mayores índices de detección de falsos positivos por lo que no se pueden utilizar como patrones de reconocimiento directo de programas malignos. En la siguiente tabla se muestran los resultados de un segundo análisis sin estos patrones.

	Positivos	Falsos positivos	% positivos	% falsos positivos	Diferencia
Total de scripts	621	418			
Scripts detectados	601	130	96.77939	31.10048	65.67891

4.2.1 Umbral

La variable que se utilizó para decidir si un determinado script es maligno o no, es la suma de los pesos de los patrones identificados en un script. El umbral es un valor predefinido con el cual se compara dicha suma. Si la suma sobrepasa el umbral entonces el script analizado se considera potencialmente maligno. De modo que el umbral que se seleccione es quien determina los índices de detección, es decir, mientras más bajo sea el umbral mayor será el número de programas identificados como potencialmente malignos, pero se debe tener en cuenta que al disminuir el valor del umbral aumenta también el número de falsos positivos.

En las primeras pruebas que se hicieron para obtener los índices generales simplemente se consideraban como potencialmente malignos los programas en los que se encontraba al menos uno de los patrones definidos. Esto es lo mismo que establecer un umbral con valor igual a 1.

4.2.2 Pesos

Otro aspecto que se tuvo en cuenta fueron los pesos; estos son los que permiten hacer un análisis heurístico. También en las primeras pruebas se utilizaron pesos unitarios, es decir, simplemente se contaba la cantidad de patrones reconocidos en un script y se comparaba con el umbral establecido. Se realizaron varios análisis con diferentes valores de umbral pero los índices de detección no fueron muy favorables. La cantidad de patrones identificados en scripts malignos no siempre es lo que define si un programa es maligno ya que varios scripts malignos están formados por solamente unas pocas instrucciones y el resto del programa es simplemente código basura. Esto hace que no se puedan establecer umbrales muy altos porque sino este tipo de scripts no se detectarían. Además, en varios scripts no malignos se reconocen algunos de los patrones de programas malignos por lo que tampoco puede ser muy bajo el umbral de decisión. Por eso fue necesario establecer pesos distintos para cada patrón según la peligrosidad de la instrucción que representa y la frecuencia de aparición en la muestra analizada.

En la siguiente tabla se observan los pesos definidos para cada patrón. La diferencia es igual al porciento de positivos menos el de falsos positivos para cada patrón. Este indicador fue utilizado anteriormente para determinar si un patrón se podía usar o no en el análisis, en dependencia de su signo, es decir, se podía usar si era positivo. El peso relativo se obtuvo a partir de esta diferencia, sumándole 1 a la división entera de la misma entre 11, de forma tal que mientras mayor sea la diferencia mayor deberá ser el peso. Esto es igual a hacer grupos de 11 y enumerarlos empezando por 1 y el peso relativo será el número del grupo en el que se encuentra el valor de la diferencia. Es posible que aun cuando la diferencia sea grande, también lo sea el índice de detección de falsos positivos por lo que el peso relativo de los patrones que tienen índices de detección de falsos positivos mayores que 3 no fueron tenidos en cuenta. Esto se hizo para reducir el probabilidad de detectar un alto número de falsos positivos. Pero no fueron desechados del todo sino que se les asignó un peso igual a 1. Luego para determinar el peso final se tomaron los mayores valores entre el peso relativo y la peligrosidad. Esta última fue determinada según las operaciones que realizan las instrucciones reconocidas por cada uno de los patrones, así como el papel que desempeñan en los programas malignos.

Patrones	% positivos	% falsos positivos	Diferencia	Peso relativo	Ajuste	Peligrosidad	Peso
FORMAT	20.12882	0	20.12882	2		3	3

ECHO, RENAME, COPY *	17.55233	0	17.55233	2		3	3
DEL, ERASE *	42.35105	1.674641	40.67641	4		3	4
NET *	0.805153	0	0.805153	1		3	3
COMMAND	1.127214	0	1.127214	1		3	3
COPY *	16.74718	0	16.74718	2		3	3
PING	1.610306	0	1.610306	1		2	2
TASKKILL	0.483092	0	0.483092	1		2	2
RUNDLL32	10.95008	0	10.95008	1		2	2
REG ADD	1.288245	0	1.288245	1		2	2
REGEDIT	4.347826	0	4.347826	1		2	2
ECHO OFF	75.36232	20.09569	55.26663	6	1	1	1
ECHO	39.93559	7.416268	32.51932	3	1	1	1
START	11.75523	8.373206	3.382028	1		1	1
CSCRIPT	3.864734	0	3.864734	1		1	1
WSCRIPT	0.805153	0	0.805153	1		1	1
DEBUG	4.347826	0	4.347826	1		1	1
MOVE	3.703704	0	3.703704	1		1	1
ATTRIB	7.407407	0	7.407407	1		1	1
REN	14.65378	0.478469	14.17532	2		1	2
FIND	6.924316	0.239234	6.685081	1		1	1
MD, MKDIR	15.61997	3.588517	12.03145	2	1	1	1
FOR	12.88245	0.956938	11.92551	2		1	2
CTTY	27.3752	0	27.3752	3		1	3
IF ERRORLEVEL	9.500805	6.698565	2.802241	1	1	1	1
DEL,ERASE	54.26731	7.416268	46.85104	5	1	1	1
COPY	32.52818	13.15789	19.37029	2	1	1	1
NET	1.288245	0.239234	1.04901	1		1	1

Una vez establecidos los pesos para cada uno de los patrones se realizaron pruebas con diferentes valores del umbral de decisión para obtener los índices de detección. Esto es importante ya que según el umbral que se escoja se obtienen determinados índices de detección. En la siguiente tabla se muestran estos valores para un rango de umbrales de decisión.

Umbral	Detectados	Scripts malignos	Porcentaje	Detectados	Scripts no malignos	Porcentaje	Diferencia
2	578	621	93.07568	69	418	16.50718	76.56851
3	531	621	85.50725	50	418	11.96172	73.54552
4	490	621	78.90499	41	418	9.808612	69.09638
5	428	621	68.9211	20	418	4.784689	64.13641
6	374	621	60.22544	6	418	1.435407	58.79004
7	302	621	48.63124	1	418	0.239234	48.39201

Se debe aclarar que los valores de los umbrales están estrechamente relacionados con los pesos para este tipo de método heurístico, de modo que si se definen valores de pesos más altos, entonces los umbrales para los que se obtienen mejores índices de detección lo serán también.

4.3 Resumen

En este capítulo se realizó un analizador para probar el reconocimiento de scripts potencialmente malignos en ficheros batch mediante los patrones obtenidos. Los cuales fueron obtenidos a partir del estudio del código de varios scripts malignos teniendo en cuenta técnicas y métodos de ofuscamiento de código. Las pruebas de reconocimiento de los patrones se hicieron utilizando un método heurístico de análisis estático. El analizador se desarrolló en C++ utilizando la librería Boost-Regex para buscar en el código de los programas instancias de los patrones obtenidos, representándolos mediante expresiones regulares.

Los resultados de las pruebas fueron positivos, obteniéndose índices de detección de scripts malignos por encima del 90% con índices de detección de 17% de falsos positivos, y de 48% con índices de detección de falsos positivos de 0.2%.

CONCLUSIONES

Las investigaciones se han dirigido hacia la búsqueda de métodos alternativos como los métodos preventivos, que están centrados más en el comportamiento de los programas que en los patrones de código. Aunque los métodos basados en firmas tienen desventajas, no se ha encontrado un método que los sustituya, ya que los métodos preventivos aisladamente son incapaces de proveer las tasas de detección necesarias para prescindir de ellos. Por tanto para una protección óptima, los métodos preventivos y los basados en firmas deben ser usados juntos y ambos deben ser actualizados periódicamente.

Al igual que los demás programas malignos los scripts malignos en ficheros batch utilizan técnicas de ofuscamiento de código como inserción de basura, transposición de código, sustitución de instrucciones, así como polimorfismo y metamorfismo. No obstante, dado que los programas interpretados no se compilan completamente a lenguaje de máquina o algún lenguaje intermedio hasta su ejecución, es posible utilizar métodos de análisis de código para detectar scripts malignos. Por tanto, si se realiza un análisis del código de varios de estos se pueden encontrar patrones que pueden ser utilizados para identificar programas potencialmente malignos que no estén encriptados, utilizando métodos heurísticos de análisis estático.

RECOMENDACIONES

Se deben seguir de cerca los programas malignos en ficheros batch ya que estos evolucionan, y reaparecen utilizando nuevas técnicas para burlar los métodos de detección, por lo que es necesario actualizar los motores de búsqueda con nuevas reglas. En caso de realizarse un analizador heurístico estático se debe tener en cuenta también la evolución de los programas no malignos en ficheros batch, ya que la validez de los patrones depende también de los índices de detección de falsos positivos. Aunque es posible que se encuentren más patrones para este tipo de programas usando una muestra más grande, el número de patrones puede ser bastante acotado. Estos están basados en los mismos comandos del sistema, y aunque cambiaran, la cantidad de patrones siempre será finita. Otra barrera es que la factibilidad de su uso depende de si los programas analizados están encriptados o no. Esto impone algunos límites a la investigación de este tipo de programas con métodos de análisis estático. No obstante, esta es una investigación larga y puede incluir otros lenguajes como Perl, Python, JavaScript, VBScript, AWK, XML, en fin, cualquier fichero script que pueda contener código maligno; aunque, es muy probable que para estos otros lenguajes script haya que utilizar un procedimiento diferente como análisis sintáctico en vez de expresiones regulares para reconocer los patrones por ejemplo.

REFERENCIAS BIBLIOGRAFICAS

1. **González, Francisco Avila.** Detección heurística de programas malignos. [En línea] Febrero de 2002. http://espejos.unesco.org.uy/simplac2002/Ponencias/Segurm%E1tica/VIR004.doc.

2. **Jackson, William.** Cybereye. *GCN (Goverment Computer News).* [En línea] 1105 Media, Inc., 16 de Abril de 2007. http://www.gcn.com/print/26_08/43494-1.html.

3. **Christodorescu, Mihai y Jha, Somesh.** Static Analysis of Executables to Detect Malicious Patterns. *Scientific and Technical Information Network.* [En línea] http://stinet.dtic.mil/oai/oai?&verb=getRecord&metadataPrefix=html&identifier=ADA449067.

4. **Gudilin, Oleg.** Proactive Protection: a Panacea for Viruses? *Viruslist.com.* [En línea] Kaspersky Lab, 28 de Enero de 2006. http://www.viruslist.com/en/analysis?pubid=189801874.

5. **Karnik, Abhishek, Goswami, Suchandra y Guha, Ratan.** Detecting Obfuscated Viruses Using Cosine Similarity Analysis. *IEEE Computer Society.* [En línea] 2007. http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/proceedings/&toc=comp/proceedings/ams/2007/2845/00/2845toc.xml&DOI=10.1109/AMS.2007.31.

6. **Bruschi, Danilo, Martignoni, Lorenzo y Monga, Mattia.** Using Code Normalization for Fighting Self-Mutating Malware. *VX Heavens.* [En línea] 2006. http://vx.netlux.org/lib/adb00.html.

7. Script Viruses. *Viruslist.com.* [En línea] http://www.viruslist.com/en/virusesdescribed?chapter=153313914.

8. Script & Macro Viruses. *Living Internet.* [En línea] http://www.livinginternet.com/i/is_vir_mac.htm.

BIBLIOGRAFÍA

Balakrishnan, Arini y Schulze, Chloe. Code Obfuscation Literature Survey. *Computer Sciences.* [En línea] 19 de Diciembre de 2005. http://www.cs.wisc.edu/~arinib/writeup.pdf.

Bruschi, Danilo, Martignoni, Lorenzo y Monga, Mattia. Using Code Normalization for Fighting Self-Mutating Malware. *VX Heavens.* [En línea] 2006. http://vx.netlux.org/lib/adb00.html.

Christodorescu, Mihai y Jha, Somesh. Static Analysis of Executables to Detect Malicious Patterns. *Scientific and Technical Information Network.* [En línea] http://stinet.dtic.mil/oai/oai?&verb=getRecord&metadataPrefix=html&identifier=ADA449067.

Elaine. HOW HEURISTIC ANALYSIS WORKS. *AVG FREE FORUM.* [En línea] Grisoft s.r.o., 20 de Junio de 2005. http://forum.grisoft.cz/freeforum/read.php?4,39695,39695.

González, Francisco Avila. Detección heurística de programas malignos. [En línea] Febrero de 2002. http://espejos.unesco.org.uy/simplac2002/Ponencias/Segurm%E1tica/VIR004.doc.

Gudilin, Oleg. Proactive Protection: a Panacea for Viruses? *Viruslist.com.* [En línea] Kaspersky Lab, 28 de Enero de 2006. http://www.viruslist.com/en/analysis?pubid=189801874.

Heuristic analysis. *Viruslist.com.* [En línea] http://www.viruslist.com/en/viruses/glossary?glossid=189210535.

Jackson, William. Cybereye. *GCN (Goverment Computer News).* [En línea] 1105 Media, Inc., 16 de Abril de 2007. http://www.gcn.com/print/26_08/43494-1.html.

Jordan, Myles. Anti-Virus Research - Dealing with Metamorphism. *ca.* [En línea] Enterprise IT Management (EITM™), 1 de Octubre de 2002. http://www.ca.com/us/securityadvisor/documents/collateral.aspx?cid=48051.

Karnik, Abhishek, Goswami, Suchandra y Guha, Ratan. Detecting Obfuscated Viruses Using Cosine Similarity Analysis. *IEEE Computer Society.* [En línea] 2007. http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/proceedings/&toc=comp/proceedings/ams/2007/2845/00/2845toc.xml&DOI=10.1109/AMS.2007.31.

Script Viruses. *Viruslist.com.* [En línea] http://www.viruslist.com/en/virusesdescribed?chapter=153313914.

Script & Macro Viruses. *Living Internet.* [En línea] http://www.livinginternet.com/i/is_vir_mac.htm.

Using batch files. *Microsoft.* [En línea] Microsoft Corporation. http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/batch.mspx?mfr=true.