Sistema de ofuscación de malware para la evasión de NIDS

SISTEMAS INFORMÁTICOS

CURSO 2012-2013

Roberto Carrasco de la Fuente
Sergio Gumiel Erena
Adrián Vizcaíno González

Directores

Julián García Matesanz
Luis Javier García Villalba

Facultad de Informática
Universidad Complutense de Madrid

Madrid, Junio de 2013

Abstract

Nowadays, effectiveness of malware detection systems is below the expected level as they have many difficulties to defend against polymorphic type attacks and zero-day attacks. So that, systems protected behind them are exposed to an imminent risk.

Furthermore, it is increasingly common to use cryptography to protect communications which opens a new vector for unnoticed attacks. Avoidance system proposed herein, shows as a NIDS is unable to detect encrypted attacks without using any type of attack known for NIDS evasion (insertion, evasion, DoS, etc.).

Despite accuracy of NIDS in malware detection is improving moreover speed processing to detect these attacks, NIDS are still quite vulnerable to attacks whose content is wholly or partially encrypted.

Keywords
NIDS, OpenMP, Snort, malware, code obfuscation, encryption code

Resumen

La eficacia de los sistemas de detección de malware de nuestros días está muy por debajo del nivel esperado ya que encuentran muchas dificultades a la hora de defenderse sobre ataques de tipo polimórfico así como ataques de día cero. Esto hace que se exponga a un peligro inminente a los sistemas protegidos tras ellos.

Además, es cada vez más común el uso de criptografía para proteger las comunicaciones lo que abre un nuevo vector para ataques inadvertidos. El sistema de evasión propuesto en el presente documento, muestra como un NIDS es incapaz de detectar ataques cifrados sin necesidad de incluir algún tipo de ataque conocido para la evasión del NIDS (inserción, evasión, DoS, etc).

A pesar de que se está mejorando la precisión en la detección de malware de los NIDS y la velocidad de procesamiento para detectar estos ataques, los NIDS todavía son bastante vulnerables a ataques cuyo contenido esté total o parcialmente cifrado.

Palabras clave
NIDS, OpenMP, Snort, malware, ofuscación de código, cifrado de código

Autorizamos a la Universidad Complutense de Madrid difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Autor 1

Autor 2

Autor 3

Índice General

Índice de Tablas

Índice de Figuras

Capítulo 1

Introducción

La utilización de *Internet* [1] y de las redes domésticas ha aumentado de forma exponencial en los últimos años, tanto a nivel doméstico como a nivel empresarial. Esto ha provocado que haya también aumentado al mismo ritmo el número de ataques e intrusiones [2] en los sistemas informáticos de todo el mundo. Actualmente el campo de la investigación en seguridad informática es uno de los objetos de estudio en el entorno de la informática más importantes que existen, sobre todo en el ámbito empresarial.

"Test Your Network Intrusion Detection System" en adelante TYNIDS es un esfuerzo realizado por los alumnos de la Facultad de informática de la Universidad Complutense de Madrid, Roberto Carrasco de la Fuente, Sergio Gumiel Erena y Adrián Vizcaíno González como proyecto final de licenciatura dirigidos por los profesores Julián García Matesanz, Luis Javier García Villalba y dentro del departamento ISIA (Departamento de Ingeniería del *software* e Inteligencia Artificial) durante el curso 2012-2013.

TYNIDS es una herramienta que tiene como cometido probar la eficacia que tiene un NIDS (*Network Intrusion Detection System*) [3] a la hora de detectar *malware*. Las metas planteadas en el origen del proyecto fueron las siguientes: crear una herramienta que sirva para evadir un NIDS mediante el encapsulado y cifrado de código *malware* y una vez conseguido esto, su optimización aplicándole técnicas de polimorfismo para hacer más eficaz la técnica de evasión utilizada.

Antes de explicar cómo se han afrontado las metas anteriormente expuestas es conveniente establecer ciertos conocimientos relativos al dominio de los sistemas de detección de intrusos, los cuales permitirán un mejor entendimiento del trabajo realizado por los autores de este documento en la elaboración de nuestra propia herramienta para la evasión de estos últimos sistemas.

1.1. Conceptos previos

Se comienza explicando en este mismo apartado conceptos necesarios tales como el de amenaza, seguridad informática, *malware*, mecanismos de defensa, ofuscación de código, así como los distintos componentes de lo que ha venido a llamarse defensa en profundidad. Para analizar este apartado introduciremos los sistemas de detección de intrusos así como otras técnicas avanzadas de detección de intrusiones tales como los sistemas de prevención de intrusiones y los *HoneyPots*. [4]

A. Amenazas

Una amenaza es la posibilidad de ocurrencia de cualquier tipo de evento o acción que puede producir un daño (material o inmaterial) sobre los elementos de un sistema. En la última década el uso de sistemas de información se ha extendido entre la población, así mismo ha aumentado la exposición de los mismos ante ataques por el entorno en el que son utilizados (conexiones de acceso público, redes domésticas inadecuadamente protegidas).

Antes, los intrusos necesitaban de un conocimiento más profundo de las redes y los sistemas informáticos para poder lanzar sus ataques. Desgraciadamente, gracias al incremento del conocimiento sobre el funcionamiento de los sistemas, los intrusos están cada vez más preparados y lo que antes estaba accesible para sólo unos pocos (expertos), hoy en día cualquiera tiene herramientas accesibles con las que poder determinar las debilidades de los sistemas y explotarlas con el fin de obtener los privilegios necesarios para realizar cualquier acción dañina.

Existen amenazas que difícilmente se dejan eliminar (virus de ordenador) y por eso es parte de la tarea de la gestión de riesgo el preverlas, implementar medidas de protección para evitar o al menos minimizar los daños en caso de que se produzca una amenaza.

Muchos de los riesgos provienen de los propios programas o aplicaciones instaladas en nuestros sistemas, puesto que en la actualidad el ciclo de vida del *software* se ha visto drásticamente reducido. Esto es debido en parte a la adopción de metodología de desarrollo y a la consiguiente mejora en el tiempo necesario para realizar un desarrollo, sin embargo en demasiados casos es simplemente debido a la elevada competitividad del mercado, que ha llevado a un aumento en la complejidad del *software* y ha obligado a reducir el tiempo destinado a diseño y pruebas del producto. Esto conlleva la presencia de errores sin detectar que posibilitan la aparición de vulnerabilidades a ser explotadas por atacantes.

B. Seguridad Informática

La definición de lo que hoy conocemos como seguridad informática está basada en la obtención de: confidencialidad, integridad y disponibilidad en un sistema de información. La confidencialidad requiere que la información sea accesible únicamente por aquellos que estén autorizados, la integridad que la información se mantenga inalterada ante accidentes o intentos maliciosos, y disponibilidad significa que el sistema informático se mantenga trabajando sin sufrir ninguna degradación en cuanto a accesos y provea los recursos que requieran los usuarios autorizados cuando éstos los necesiten.

C. *malware*

Es un tipo de *software* que tiene como objetivo infiltrarse o dañar una computadora o sistema de información sin el consentimiento de su propietario. El *software* se considera *malware* en función de los efectos que, pensados por el creador, provoque en un computador. El término *malware* [5] incluye virus, gusanos, troyanos, la mayor parte de los *rootkits*, *scareware*, *spyware*, *adware* intrusivo, *crimeware* y otros *softwares* maliciosos e indeseables.

D. Mecanismos de defensa

Para la correcta protección de un sistema existen diversos mecanismos tales como el cifrado, la identificación y autentificación de usuarios para proveer confidencialidad e integridad. Por otro lado existen mecanismos que velan por la disponibilidad del sistema filtrando la información, ejemplos de estos mecanismos son las listas de acceso y cortafuegos. Como última línea de defensa de un sistema de información encontramos técnicas enfocadas a la detección de las amenazas tales como antivirus y sistemas de detección de intrusos. La característica distintiva de estas técnicas es que al contrario de las anteriores permiten detectar intentos de romper la seguridad de un sistema.

E. Ofuscación de código

El código ofuscado es aquel código que, aunque se tiene el código fuente, ha sido enrevesado específicamente para ocultar su funcionalidad (acto de hacerlo ininteligible). Por ejemplo la ofuscación binaria se realiza habitualmente para impedir o hacer más difícil los intentos de ingeniería inversa y desensamblado que tienen la intención de obtener una forma de código fuente cercana a la original.

F. Panales de miel (*HoneyPots*)

En el segundo caso nos encontramos con una técnica complementaria basada en la creación de elementos artificiales, aparentemente vulnerables con el objetivo de atraer a los atacantes. Estos elementos de red no pueden ser accedidos por ningún servicio legítimo del sistema por lo que todo acceso es inmediatamente catalogado como ataque. Esto resulta útil para reunir información sobre técnicas de ataque y su procedencia las cuales pueden ser usadas a posteriori para la mejora del resto de elementos de seguridad de la red.

G. Intrusión

Se puede definir intrusión como la violación de la política de seguridad de un sistema o como la materialización de una amenaza. Heady et al.[6] definen intrusión como cualquier conjunto de acciones de tratar de comprometer la integridad, confidencialidad o disponibilidad de un recurso.

Una de las definiciones más populares de intrusión es: fallo operacional maligno, inducido externamente [7], aunque bien es sabido que muchas de las intrusiones proceden del interior del sistema de información. Una vez definido el término de intrusión, podemos enunciar de una manera más formal el significado de las siglas IDS. El NIST (*National Institute of Standards and Technology*) define detección de intrusos como el proceso de monitorización de eventos que suceden en un sistema informático o red y análisis de dichos eventos en busca de signos de intrusiones.

H. Sistemas de detección de intrusos en una red (IDS)

Se trata de un programa que detecta intrusiones en un sistema a través de la red. Como definición inicial es suficiente, no obstante durante todo el documento se encuentran referencias constantes a este término y profundizaremos más. En el apéndice del documento

se ha incluido un artículo sobre los inicios de los IDS y como han ido evolucionando hasta la actualidad.

I. Sistemas de prevención de intrusos en una red (IPS)

Se trata de un dispositivo que ejerce el control de acceso en una red informática para proteger a los sistemas computacionales de ataques y abusos. La tecnología de Prevención de Intrusos es considerada por algunos como una extensión de los Sistemas de Detección de Intrusos (IDS), pero en realidad es otro tipo de control de acceso, más cercano a las tecnologías cortafuegos.

Utilizan las mismas técnicas de detección que los IDS, pero se diferencian principalmente en que realizan el análisis sobre el tráfico real de la red, en lugar de realizarlo sobre copias del mismo, esto es denominado modo in-line. Así mismo los IPS usualmente implementan mecanismos de respuesta activa. Los IPS fueron inventados de forma independiente por Jed Haile y Vern Paxon [8] para resolver la confusión en la monitorización pasiva de redes de computadoras, al situar sistemas de detecciones en la vía del tráfico. Los IPS presentan una mejora importante sobre las tecnologías de cortafuegos tradicionales, al tomar decisiones de control de acceso basados en los contenidos del tráfico.

J. Sistemas de detección de intrusos a nivel de Host (HIDS)

Este tipo de IDS fue el primero en ser desarrollado y desplegado. Típicamente están instalados sobre el servidor principal, elementos clave o expuestos del sistema. Los HIDS operan monitorizando cambios en los registros de auditoría del *host* tales como: procesos del sistema, uso de *CPU*, acceso a ficheros, cuentas de usuario, políticas de auditoría, registro de eventos. Si los cambios en alguno de estos indicadores exceden el umbral de confianza del HIDS o hace peligrar la integridad del sistema protegido el HIDS tomaría una acción en respuesta. La principal ventaja de estos detectores es su relativo bajo coste computacional puesto que monitorizan información de alto nivel, sin embargo esto también constituye su principal inconveniente puesto que sólo pueden lanzar una alerta una vez que el daño al sistema ya está hecho. Ejemplos de HIDS comerciales son: Symantec Host IDS, ISS BlackICE PC16, TCPWrappers, Enterasys Dragon Host Sensor.

K. *Shellcode*

Primero es necesario saber el significado etimologico de *shellcode* [9]. *Shell* es como se le dice en inglés a una terminal. Por lo tanto *shellcode* significa que es un código que nos crea una terminal para que la usemos.

Actualmente el término *shellcode* no solo se refiere al código que nos permite crear una *shell*, sino que se extiende a códigos con ás funciones como pueden ser crear usuarios, matar/crear un determinado proceso, dar permisos a una carpeta, etc. Técnicamente, una *shellcode* es una porción de código compilada para un determinado procesador que se ejecuta dentro de otro proceso previamente preparado para que la ejecute (ej: *stack overflow*).

Son pequeños programas escritos en ensamblador que, en la mayoría de los casos, ejecutan una línea de comandos propia del sistema operativo para el cual han sido escritos. Generalmente los *Shellcodes* son parte del código de un *exploit*.

L. *Exploit*

Un *exploit* es un programa, escrito en cualquier lenguaje de programación, que pretende tomar ventaja sobre algun error o descuido en la programación de cualquier otro *software*. Generalmente están escritos con la finalidad de obtener privilegio sin autorización o inhabilitar cualquier equipo. Las vulnerabilidad que suelen explotar y que serán explicadas en capítulos siguientes son: *Buffer Overflow, Format String, Heap Overflow, Cross Site Scripting, Heap Spraying.*

1.2. Objeto de la investigación

El objetivo de esta investigación ha sido implementar técnicas de ofuscación [10] de *malware* para la evasión de Sistemas de Detección de Intrusos, más concretamente para Sistemas de detección de Intrusos en una Red (de ahora en adelante utilizaremos sus siglas en inglés NIDS para referirnos a ellos). En concreto se ha trabajado para construir una aplicación capaz de cifrar *malware*, pudiendo escoger entre varios algoritmos de cifrado de clave simétrica, y encapsularlo en un fichero perfectamente ejecutable en una máquina objetivo. Una vez cumplido este objetivo se han ido aplicando técnicas de mejora como el uso de polimorfismo [11] para hacer que el cifrado de *malware* sea aún más potente y que permita evadir NIDS previamente entrenados en la detección de dicho *malware*.

1.3. Estructura de trabajo

En el presente capítulo se han introducido conceptos tales como las amenazas, seguridad informática, *malware*, ofuscación de código y posibles mecanismos de defensa. Además se ha introducido el objeto de este trabajo que es la implementación de una herramienta que sirve para camuflar de tal manera el *malware* que pueda evadir a un sistema de detección de intrusos.

En el siguiente capítulo se tratarán los siguientes temas:

- Técnicas de ofuscación de *malware* (subcapítulo 2.1)
- Técnicas de evasión de NIDS (subcapítulo 2.2)
- Arquitectura x86 y técnicas de acceso a memoria privilegiada (subcapítulo 2.3)

A continuación se introducirá la herramienta "TYNIDS" que se ha implementado y una explicación de su funcionamiento (Capítulo 3). Seguidamente se muestran los resultados obtenidos con la ejecución de la herramienta (Capítulo 4).

Presentaremos las conclusiones y propuestas de trabajo futuro que consideramos de gran interés para la mejora de nuestra aplicación en el acto de evadir los sistemas de detección de intrusos. (Capítulo 5)

Por último, encontramos los apéndices que contienen información relacionada con el proyecto que no tiene cabida en los anteriores capítulos pero que se ha pensado que sí que tiene que quedar reflejado de alguna manera en este trabajo ya que nos ha parecido una información útil e interesante para el lector.

Capítulo 2

Estado del arte

En esta sección se muestran ciertos aspectos que motivan la investigación en torno al rol de los sistemas de detección de intrusos y cómo es posible evadirlos. Se da una perspectiva general de las técnicas de ofuscación de *malware* que existen en la actualidad, se explican distintas técnicas de evasión de NIDS y se hace una introducción a la arquitectura x86 y a las técnicas de acceso a memoria privilegiada.

2.1. Técnicas de ofuscación de *malware*

2.1.1. Introducción

En esta sección se enumeran y definen distintos métodos para transformar el código P de un *malware* [11] a un código P' sin que pierda funcionalidad y se mantenga el comportamiento. Con esta acción se pretende conseguir que el código malicioso no sea detectado por un sistema de detección de intrusos [12].

Se hace referencia a diversos ejemplos de ofuscación de código [13] para explicar este hecho y se da una perspectiva general de cómo han ido evolucionando las distintas técnicas a lo largo del tiempo. También se hace referencia a técnicas de ofuscación a nivel de capa de aplicación [14].

Y finaliza con la explicación de una posible aplicación de la ofuscación para prevenir el SPAM [15], además de una breve conclusión.

2.1.2. Técnicas de ofuscación de *malware* a nivel de *shellCode / opcode*

A. Definición de ofuscación

Término que fue formalizado por Barak et al.[16]. La ofuscación de *software* es una técnica para ocultar el flujo de control del *software*, así como las estructuras de datos que contienen información sensible y se utiliza también para mitigar la amenaza de la ingeniería inversa [17], también conocida como reingeniería.

En el año 1997 Collberg et al. [18] hizo pública una taxonomía completa de transformaciones del código para lograr su ofuscación. Para medir el efecto de una ofuscación, Collberg define tres parámetros: Potencia, resistencia y coste.

La potencia se describe como la dificultad que un ser humano tiene para entender un programa P' ya ofuscado. Existen distintas métricas de complejidad de *software* [19] para realizar estas mediciones, aunque es un indicador bastante subjetivo.

En contraste con la potencia que evalúa la fuerza de la transformación de ofuscación contra la capacidad de entendimiento de los seres humanos, existe la resistencia que define lo bien que aguanta el ataque de un desofuscador automático [20] un código previamente ofuscado.

Con el transcurso del tiempo han ido surgiendo nuevas técnicas de ofuscación de codigo malicioso (figura 2.1). Primero se hará mención a su nombre, en orden cronológico de su aparición y, a continuación, pasaremos a explicar en qué consisten.

Figura 2.1: Técnicas de ofuscación de código malicioso

B. Cifrado

El primer virus cifrado conocido apareció en 1987 con el nombre de "Cascade" [21]. En la figura 2.2 se muestra el antes y después del descifrado.

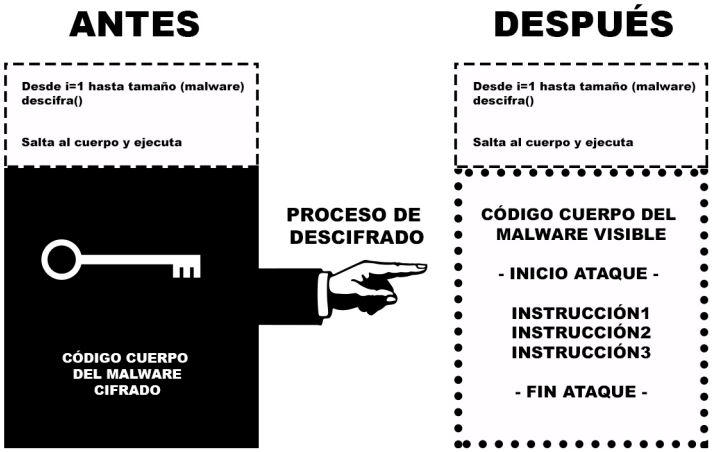

Figura 2.2: Virus *Cascade*

Un virus cifrado se compone de dos secciones básicas: un bucle de descifrado y el cuerpo principal. *Decryptor* [22], o bucle de descifrado, es un pequeño fragmento de código, que se encarga de cifrar y descifrar el código del cuerpo principal. El cuerpo principal es el código real del malware, cifrado, y no tiene sentido antes de ser descifrado por el bucle de descifrado. Cuando el virus empieza a actuar en el equipo destino, el primer bucle descifrador debe descodificar el cuerpo principal. Un antivirus no puede detectar inmediatamente el virus mediante firma ya que primero tiene que descifrar el cuerpo del virus para acceder

a todo el código. Sin embargo, si que puede detectar por firma la parte del módulo de descifrado.

C. Oligomorfismo

Un virus oligomórfico es una forma avanzada de cifrado. Contiene una colección de descifradores diferentes, que son elegidos al azar para cada nueva víctima. De esta manera, el código de descifrado no es idéntico en todos los casos. El primer virus conocido oligomórfico era "*Whale*" [23], un virus de DOS que fue presentado en 1990.

El oligomorfismo no supone un gran problema para el *software* antivirus, ya que sólo tiene un *malware* ligeramente más difícil de observar. A diferencia del virus cifrado, el motor del antivirus tiene que comprobar todas las instancias posibles del *Decryptor*, en lugar de buscar únicamente un descifrador, y necesita de más tiempo. (Figura 2.3)

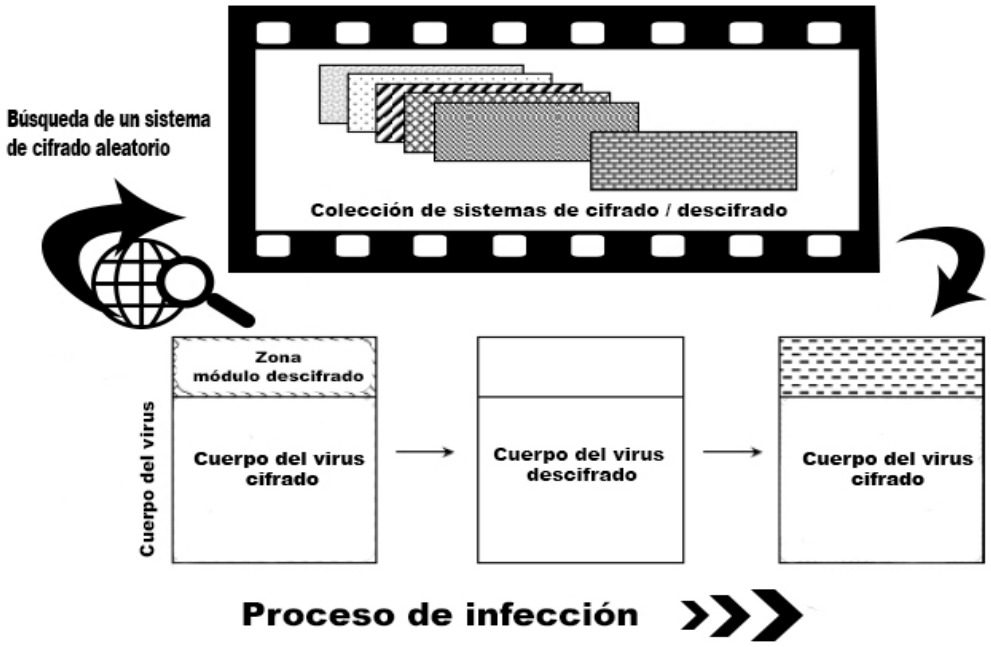

Figura 2.3: Proceso de infección en virus oligomórfico

D. Polimorfismo

Los virus polimórficos son similares a los virus cifrados y oligomorficos en cuanto al uso de un código de cifrado, pero la diferencia es que los polimórficos son capaces de crear un número ilimitado de nuevos descifradores diferentes.

El primer virus polimórfico llamado "1260" [24], un virus de la familia de camaleón que apareció en 1990, fue desarrollado por Mark Washburn. La regla principal es la de modificar el aspecto del código constantemente de una copia a otra.

Debe llevarse a cabo de tal forma que no haya cadena común permanente entre las variantes de un virus para que el motor del escáner del antivirus no lo detecte. Las técnicas

polimórficas [11] son bastante difíciles de implementar y administrar. (Figura 2.4)

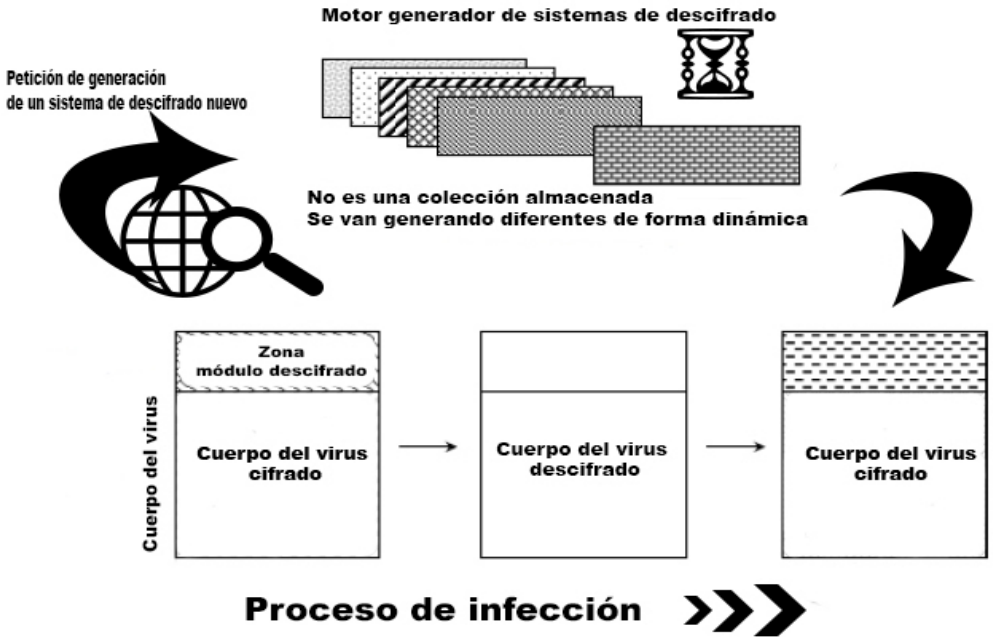

Figura 2.4: Proceso de infección en virus polimórfico

E. Metamorfismo

Un virus metamórfico tiene el motor metamórfico embebido dentro de sí mismo. Un ejemplo es el virus "*Zperm*" [25] que no usa un cifrado polimórfica tradicional. En su lugar, muta su código mediante la inserción de instrucciones de salto. El cuerpo completo del virus permuta, incluido el propio motor de permutación. Durante la infección, el virus metamórfico crea varias copias de sí mismo transformadas utilizando este motor incorporado.

A diferencia de las tres generaciones anteriores de camuflaje u ofuscado, el virus metamórfico (figura 2.5) no tiene parte cifrada. Por lo tanto, no necesita descifrador, pero como virus polimórfico que es, emplea un motor de mutación, así, en vez de modificar el bucle de descifrado solamente, lo que muta es todo su cuerpo.

La definición más concisa de los virus metamórficos es introducida por Igor Muttik. Def. "*metamorphics are body-polymorphics*". [25] Cada nueva copia puede tener diferente estructura, secuencia de código, tamaño y propiedades sintácticas, pero el comportamiento del virus no cambia. El primer virus conocido metamórfico que se produjo para DOS fue ACG, en 1998.

Un motor metamórfico [26] debe incluir las siguientes partes:

1. Desensamblador
2. Código analizador
3. Código transformador

4. Ensamblador

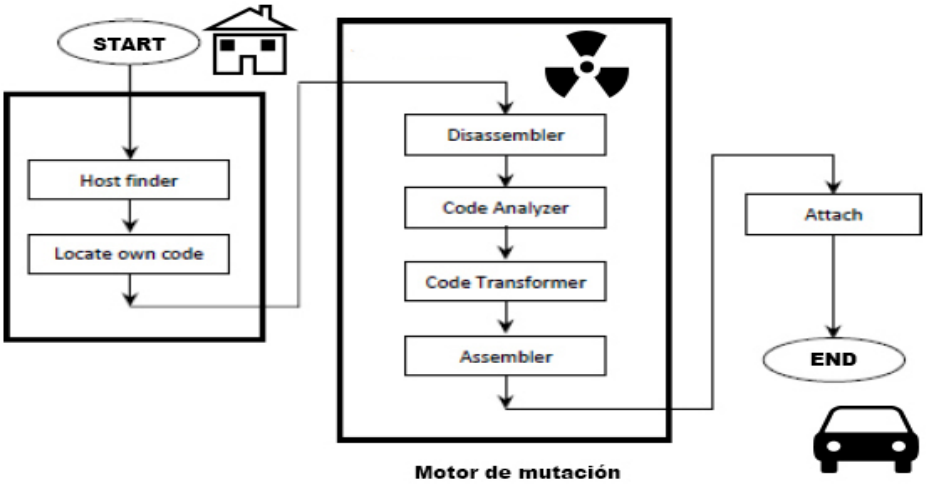

Figura 2.5: Proceso de infección virus metamórfico

Después de que el virus encuentre la ubicación de su propio código, se realiza un desensamblado de forma interna. El analizador de código es el responsable de proporcionar información para el módulo transformador. El transformador de código puede necesitar cierta información tal como la estructura y el diagrama de flujo del programa, subrutinas, período de vida de variables y registros, y así sucesivamente. Esta información ayuda al transformador de código para que funcione adecuadamente. El código transformador u ofuscador es el corazón del motor de la mutación. Es el responsable de ofuscar el código y cambiar la secuencia binaria del virus. Se pueden utilizar todas las técnicas de ofuscación que se han mencionado anteriormente para los virus polimórficos. El último módulo, ensamblador, convierte el nuevo código mutado en código máquina binario.

Los virus metamórficos que estén bien construidos no contendrán firmas que coincidan con transformaciones anteriores. Esto significa que un *malware* metamórfico profesional es capaz de producir un número ilimitado de variantes similares en comportamiento y que no contengan la vulnerabilidad de producir un patrón único que pueda ser detectado. Por lo tanto, los motores de análisis antivirus deben usar heurística y técnicas de análisis basadas en comportamientos para poder detectar este tipo de *malware*.

Así mismo esta categoría de *malware* se puede subdividir en dos subcategorías.

La primera sería la hipotésis del mundo abierto la cual permitiría que el *malware* se pueda comunicar con otros sitios en Internet y pueda descargar actualizaciones. Un ejemplo es el gusano "*Conficker*" [27], 2008, comprometió muchos PC's usando una vulnerabilidad RPC sobre MS Windows.

La segunda subcategoría se basa en la hipótesis de un mundo cerrado, lo que implica que no haya comunicación ninguna con el exterior durante la mutación. El primer virus que uso esta técnica fue Win32/Apparition [25]. El motor metamórfico sigue un algoritmo general para generar copias metamórficas del archivo de virus base. El algoritmo en

lenguaje de alto nivel se resume en la figura 2.6. que se muestra en adelante.

Con posterioridad a la generación de virus metamórfico, han surgido nuevas técnicas de ofuscación, como por ejemplo la ofuscación en JavaScript o la ofuscación de código dinámica. Estas están todavía en vías de desarrollo. El *malware* sobre la *web* es una grave amenaza para los usuarios de Internet y cada vez supone un reto mayor para los fabricantes de antivirus.

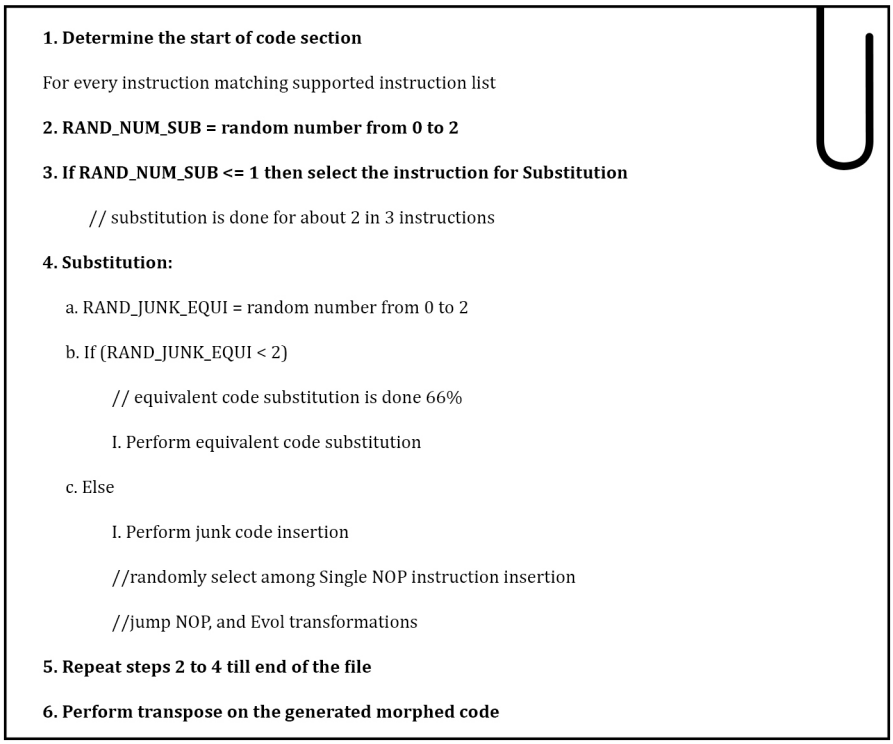

Figura 2.6: Algoritmo general virus metamórfico

F. Operación de Mutación

Entre estas técnicas podemos encontrar las siguientes:

- Inserción de código basura o código muerto.
- Sustitución de variables / registros.
- Reemplazo de instrucciones.
- Permutación de instrucciones.
- Transposición de código.

I. Inserción de código basura:

La inserción de código muerto o código basura es la solución más fácil para modificar la secuencia binaria del programa de un virus sin ningún efecto sobre la funcionalidad de

código y su comportamiento. Existen diversos tipos de códigos de basura. En la figura 2.7. y 2.8. se muestran algunos ejemplos de estas instrucciones. No cambian el contenido de los registros de la CPU ni de la memoria, en inglés son conocidas como "No-operation" (NOP).

Instruction		Operation
ADD	Reg,0	Reg←Reg+0
MOV	Reg,Reg	Reg←Reg
OR	Reg,0	Reg←Reg | 0
AND	Reg,-1	Reg←Reg&-1

Figura 2.7: Ejemplo de instrucciones de código basura (NOP) I

Instruction		*Comments*
PUSH	***CX***	*It push value of AX into stack, later it must be turned back to AX before any effects on AX or stack memory*
...	...	
POP	***CX***	
INC	***AX***	*The value of DX increases by 14, and later before any usage of DX, its value must be changed back to its previous value*
...	...	
SUB	***AX, 1***	

Figura 2.8: Ejemplo de instrucciones de código basura (NOP) II

II. Sustitución de variables registros:

Otro método utilizado por los motores de mutación es el intercambio de registros o variables de memoria en diferentes instancias del virus. Con esta técnica, el virus trata de derrotar a la detección por firmas, mediante la conversión de los *bytes* idénticos en las diferentes generaciones. W95.Regswap fue uno de los primeros virus que hace uso de este enfoque para producir diversas variantes del virus, en diciembre de 1998. Obviamente, esto no cambia el comportamiento del código, pero modifica la secuencia binaria del código. Este método relativamente simple en combinación con otras técnicas pueden producir variantes bastante complicadas de un virus que haga que no puede ser descubierto fácilmente y hacer que la detección basada en firmas sea muy poco práctica.

III. Reemplazo de instrucciones:

Esta técnica intenta reemplazar algunas instrucciones por instrucciones equivalentes. Por ejemplo, todas las instrucciones siguientes son diferentes y sirven para configurar el registro eax con un cero. (Figura 2.9) Los programadores de virus aprovechan esta habilidad para añadir la ofuscación a los códigos. Es similar al uso de diferentes sinónimos en lenguaje natural.

mov	eax, 0
xor	eax, eax
and	eax, 0
sub	eax, eax

Figura 2.9: Instrucción inofensiva de relleno

IV. Permutación de instrucciones:

En muchos programas, el programador tiene la capacidad para reordenar una secuencia de instrucciones de forma segura. Es decir, aunque se cambie el orden de las instrucciones el resultado del programa tiene que ser el mismo (figura 2.10). Este caso se da cuando algunas instrucciones son independientes entre sí. Cambiar su orden, no afectará al resultado final.

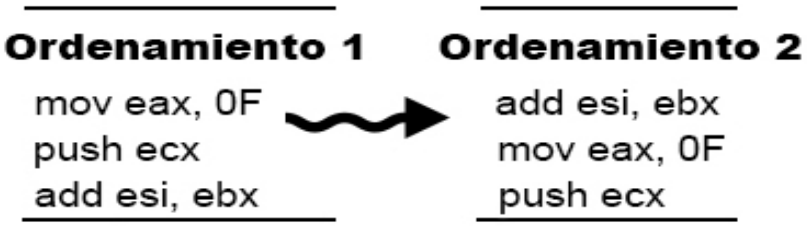

Figura 2.10: Ejemplo de permutación de instrucciones

V. Transposición de código:

Este tipo de ofuscación permite cambiar la estructura del programa, de tal manera que aunque reordenemos el código, el flujo del código sigue siendo el mismo (figura 2.11). Las ramas condicionales o incondicionales se siguen comportando igual. La transformación se puede realizar a nivel de instrucción o bloque de código.

2.1.3. Ofuscación de *malware* a nivel de capa de aplicación

Como se ha mencionado anteriormente existen numerosos caminos o métodos que se pueden seguir para intentar ofuscar el código malware. Uno de los más simples puede ser cambiar el nombre a las variables por nombres (figura 2.12) que no aporten significado al código.

Pero al igual que evoluciona la tecnología, también evoluciona los objetivos de los atacantes. Cada vez se usan más los dispositivos móviles, los navegadores *web* para visitar las páginas del banco, casas de apuestas, sitios de compras, etc. En definitiva, medios de información que manejan datos muy sensibles. Por ello se está haciendo más hincapie en el desarrollo de técnicas de ofuscación de *malware* para dispositivos móviles o bien *malware*

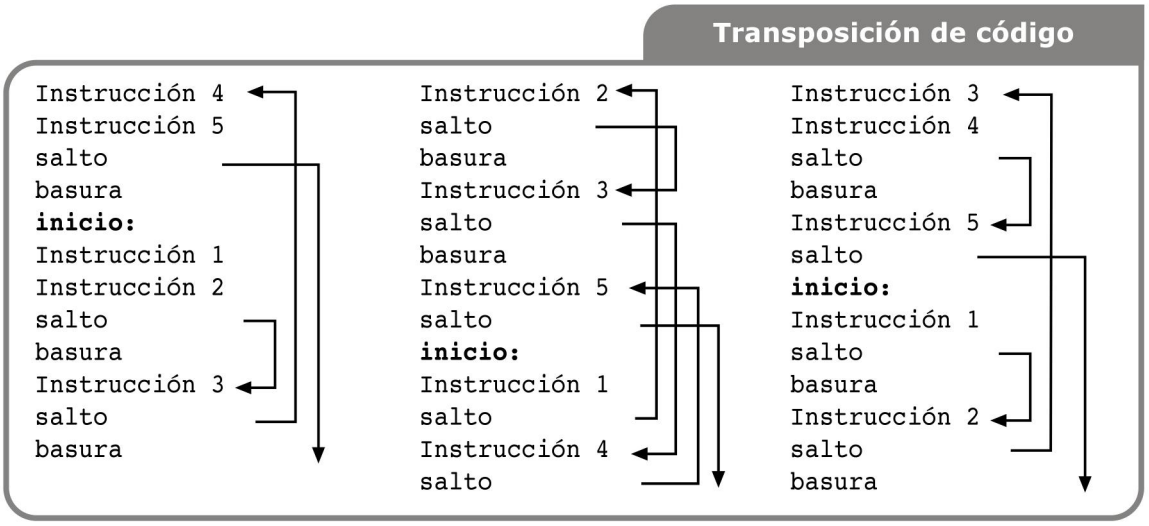

Figura 2.11: Ejemplo de transposición de código en el Virus Zperm

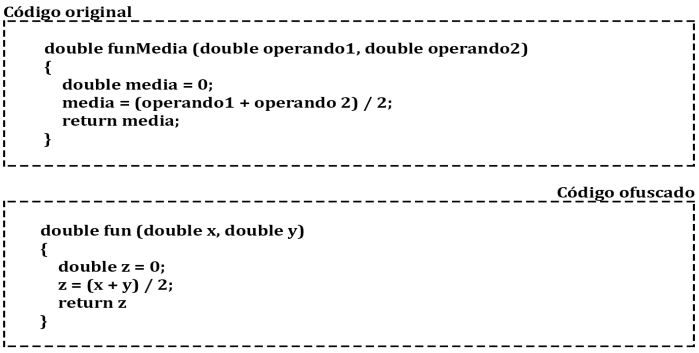

Figura 2.12: Ofuscación por cambio de nombre de variables

que tiene como objetivo ejecutarse en navegadores de clientes. Esto se consigue por ejemplo mediante la ofuscación de código Javascript.

A. Ofuscando código JavaScript

Una de las técnicas más usadas por los atacantes para explotar vulnerabilidades en los navegadores es la ofuscación de *scripts* [28]. Los *scripts* que son ejecutados en la parte del cliente suelen estar escritos en Perl, JavaScript, VBScript, ofuscando el código se consigue complicar la tarea de comprender, estudiar o analizar la funcionalidad del mismo.

Esto ha conseguido que sea una de las técnicas más utilizadas cuando se habla de *malware* desarrollado específicamente para objetivos *web*. El objetivo es transformar el código en algo totalmente incomprensible mientras se transfiere desde el servidor *web* hasta el navegador del cliente, pero siempre manteniendo su funcionalidad. A continuación un código sin ninguna transformación.

```
<script language=javascript>
    function factorial( val, is_transient){
        var ret = val ==  1 ? 1 : factorial( val-1, 1) * val;
        if (!is_transient) {
           document.write("factorial of" + val + "is: " + ret + "<br>");
        };
    };
    factorial(6, 0) ;
    factorial(3, 0) ;
    factorial(9, 0) ;
</script>
```

Para poder ofuscar el código correctamente se puede seguir los siguientes pasos:

Cambiar el nombre de las variables utilizadas en el código por caracteres sin sentido por ejemplo "factorial" puede llamarse "ZmFjdG9yaWFs". Reemplazar constantes o valores numéricos por expresiones, así "485" puede ser "0x16c1+3640-0x2314". Reemplazar los caracteres de los strings por sus valores hexadecimales correspondientes, así "javascript" será:

```
\x6a\x61\x76\x61\x73\x63\x72\x69\x70\x74
```

Eliminar cualquier espacio o indentación existente en el código, de forma que este quede más compacto y sea más complicado de leer. Eliminar cualquier anotación o comentario en el código que pueda dar pistas al usuario de la funcionalidad del mismo. Ejemplo de como quedaría ofuscado.

```
<script language=javascript>
function z60b72bb3fe (z387ef0e78e, zd06054e7d1)
{
   var z205fd1aa25 = z387ef0e78e == (0x397+8978-0x26a8) ?
    (0xd81+6110-0x255e) : z60b72bb3fe( z387ef0e78e - (0x1083+838-0x13c8),
    (0x463+3498-0x120c)) * z387ef0e78e;
   if (!zd06054e7d1) {
     document.write("\x66\x61\x63\x74\x6f\x72\x69\x61\x6c\x20\x6f\x66\x20"
       + z387ef0e78e + "\x20\x69\x73\x3a\x20" + z205fd1aa25 +
       "\x3c\x62\x72\x3e");
    };
}
    z60b72bb3fe((0x11e8+2586-0x1bfc), (0xa63+7224-0x269b));
    z60b72bb3fe((0xfc5+2132-0x1816), (0x1119+3554-0x1efb));
    z60b72bb3fe((0x10b3+1338-0x15e4), (0x846+7200-0x2466));
</script>
```

Como se aprecia, el código queda completamente ilegible, y esta es una técnica muy usada para evadir las firmas de detección utilizadas por aplicaciones de seguridad como los IPS (Intrusion Prevention Systems), escáneres de *malware* o las aplicaciones de filtrado, por mencionar algunos ejemplos.

B. Ofuscación mediante el uso de iFrames

El iframe [29] no deja de ser una página *web* HTML, con la peculiaridad de que permite insertar una página dentro de otra. Esto ha provocado que se convierta en una de las técnicas preferidas por los atacantes para inyectar *malware* en las aplicaciones *web*.

Un ejemplo de las funciones sería:

```
escape("URL a codificar") se convierte \%22URL\%20a\%20codificar\%22
unescape(\%22URL\%20a\%20codificar\%22) se queda "URL a codificar"
```

Su funcionamiento consiste en encontrar parejas de 2 o 4 caracteres hexadecimales e intercambiarlos por su representación en Latin-1, de esta forma %20 es el equivalente al espacio, %21 una exclamación, %22 es el equivalente a la comilla doble, etc.

Código antes del proceso de ofuscación

```
<iframe src="http://URL-MALICIOSA" width="0" height="0"
frameborder="0"></iframe>
```

Código después del proceso de ofuscación

```
\%3Ciframe\%20src\%3D\%22http\%3A\%2F\%25\%2FURL-MALICIOSA\%22
\%20width\%3D\%220\%22\%20height\%3D\%220\%22\%20frameborde\%72
\%3D\%220\%22\%3E\%3C\%2Fiframe\%3E
```

C. *Cross Site Scripting* (XSS)

Si se ha de hablar de una de las mayores vulnerabilidades en aplicaciones *web* usadas para propósitos maliciosos, esa es *Cross Site Scripting* [30] o también conocido como XSS.

Permite a un atacante alterar o modificar el código fuente de la aplicación, resultando en una ampliación del ataque y en la infección de un mayor número de usuarios potenciales. Una vez que ha tenido éxito el ataque, la fase de propagación puede realizarse tan rápida como lo hizo el gusano Blaster, Slammer, etc. [31] La detección y explotación de una vulnerabilidad no conocida puede desencadenar en consecuencias desastrosas. Propagando la infección de *malware* en sitios de confianza. Es común que sitios como redes sociales, foros, chats, correos, sean los objetivos escogido para perpretar un ataque debido al alto tráfico que reciben.

Se trata de una vulnerabilidad capaz de causar un impacto considerable y perjudicial sin necesidad de estar presente en un determinado navegador o sistema operativo.

Estas se producen cuando la entrada dada por un usuario es aceptada y no se valida, transmitiéndose al lado del cliente con una codificación incorrecta. Esta situación provoca que se ejecuten secuencias de comandos malintencionados.

Normalmente suele aparecer en las páginas dinámicas que no suelen tener un control exhaustivo sobre sus salidas ni sobre cómo procesan internamente la entrada de datos. Para entenderlo supongamos que un atacante envía el siguiente código malicioso.

```
<a href=http://www.tuenti.com/#m=Profile&func=index&user_id=<script>
alert("XSS");</script></a>
```

Cuando la víctima haga *click* sobre el vínculo se abrirá un enlace hacia la dirección apuntada junto con el código contenido entre las etiquetas *script*.

D. Ofuscación de Código Dinámica

I. Mecanismo de Protección o Amenaza

La ofuscación de código dinámica es una nueva forma de enmascarar el código, veremos las ventajas y riesgos que implican el descubrimiento de esta nueva técnica, así como algunos de las métodos que existen tanto para ofuscar el código, como para lograr la desofuscación del mismo, además de las principales formas y lugares donde esta técnica puede ser aplicada. La ofuscación de código dinámica es una técnica que transforma el código en garabatos incomprensibles, esta técnica puede ser utilizada con dos propósitos principales, el primero como una herramienta de protección de derechos de autor del código de los programas, y el segundo como una amenaza a los mecanismos actuales de seguridad en la red.

La mayoría de las empresas tratan de evitar que su código fuente pueda ser obtenido por alguien más, es por esto que hacen uso de herramientas que les permitan evitar que alguien lo pueda obtener. Sin embargo la ofuscación también sirve como un mecanismo para generar muchas versiones del mismo código, logrando de esta manera sobrepasar los mecanismos usuales de análisis de los sistemas de seguridad, como los antivirus.

II. La amenaza oculta

En la actualidad, a pesar de que los antivirus son cada día más potentes y hacen uso de técnicas mucho más complejas para lograr evitar las amenazas, nace una nueva técnica que logra poner en dificultad a estos sistemas actuales de defensa, esta nueva técnica es llamada ofuscación de código dinámica, y se basa en la transformación o enmascaramiento del código para evitar que se pueda determinar la forma que tenía originalmente.

La ofuscación de código dinámica empezó a funcionar con Javascript un poco después de que fuera liberado este lenguaje en 1995, se hizo por la principal razón de proteger

el código que tanto trabajo les costaba desarrollar a los programadores, y que cualquier usuario podía ver usando un explorador con esas características.

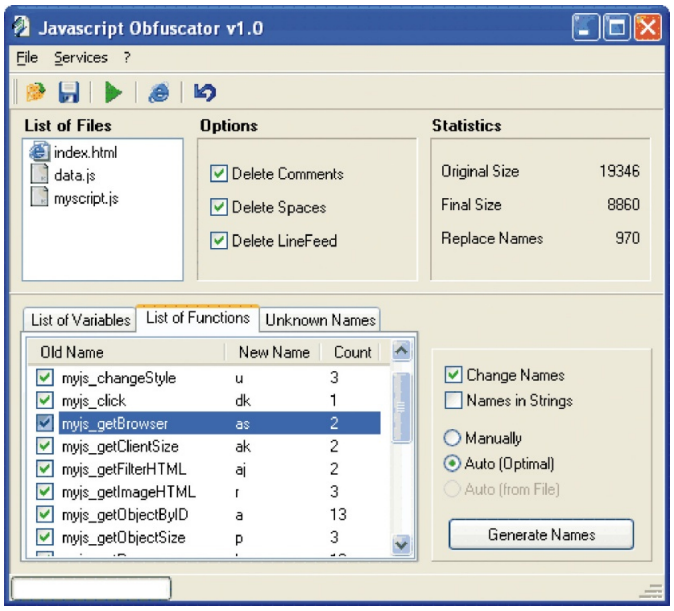

Figura 2.13: Ejemplo de aplicación de ofuscación de código

Existen muchas herramientas (figura 2.13) que ayudan a la fácil ofuscación de código, las cuales permiten proteger los códigos dinámicos o de *scripting*, aunque también se puede utilizar con fines malignos.

III. Mecanismo de protección

El código fuente de un programa es muy valioso, siendo una creación intelectual, es susceptible de ser utilizado sin haber tenido que pasar por un proceso creativo nuevamente, es decir, una vez que se desarrollóa el código, puede volver a ser utilizado sin haber tenido que emplear tiempo en su desarrollo, es por esto que las empresas y desarrolladores protegen tan celosamente su código. Una manera de entre muchas que existen en la actualidad para proteger su código, en otras palabras, su propiedad intelectual, es la ofuscación, que se ofrece como una manera de poder ejecutar el código por otras personas, sin que estas puedan llegar a entender cómo funciona en realidad.

La ofuscación de código ha sido propuesta por un gran número de investigadores como un medio para dificultar la realización de ingeniería inversa sobre el código. Pero las transformaciones típicas de ofuscación se basan en razonamientos estáticos sobre ciertas propiedades de los programas, lo cual da como resultado que la ofuscación del código pueda ser fácilmente desentrañada mediante una combinación de análisis estáticos y dinámicos.

Conceptualmente se pueden distinguir dos tipos de ofuscación de código: la ofuscación superficial y la ofuscación profunda. La primera siendo solamente un reacomodo de la sintaxis del programa, ya sea cambiando nombres de variables o algún otro método similar, y la segunda que intenta cambiar la estructura actual del programa, cambiando su control de flujo o modo de referenciar sus datos, esta última teniendo menos efecto en el proceso

de estar oculto a la ingeniería inversa, ya que no se ocupa de disfrazar la sintaxis del código.

IV. Técnicas de ofuscación

Una de las técnicas de ofuscación es la llamada "aplanamiento de control de flujo básico", la cual intenta esconder la lógica de flujo del programa para que simule que todos los bloques básicos tienen el mismo antecesor y el mismo predecesor.

El control de flujo actual es guiado mediante un despachador de variables, el cual en tiempo de ejecución asigna a cada bloque básico un valor variable, que indica cual es el siguiente bloque básico que debe ser ejecutado a continuación. Entonces un bloque *switch* utiliza el despachador de variable para saltar de manera indirecta, a través de una tabla de saltos hacia el sucesor de control de flujo que debe ser.

Existen varias mejoras en esta técnica de ofuscación, siendo una de ellas el flujo de datos interprocedimental, las variables que asignaba el despachador se encontraban disponibles dentro de la función misma, debido a lo cual aunque el comportamiento del control de flujo del código ofuscado no sea obvio, puede ser reconstruido mediante un análisis de constantes asignadas por el despachador de variables.

La ofuscación del código puedes ser más difícil de reconstruir si además de lo anterior agregamos el paso de variables entre los procedimientos, la idea es utilizar un arreglo global para pasar los valores asignados por el despachador de variables, así cada vez que se llame a la función se pasará el valor al arreglo de variables global, logrando de esta manera que los valores asignados cada vez que se llama la función no sean constantes, sino que al examinar el código ofuscado no sea evidente el proceso que se realizó en la llamada a la función.

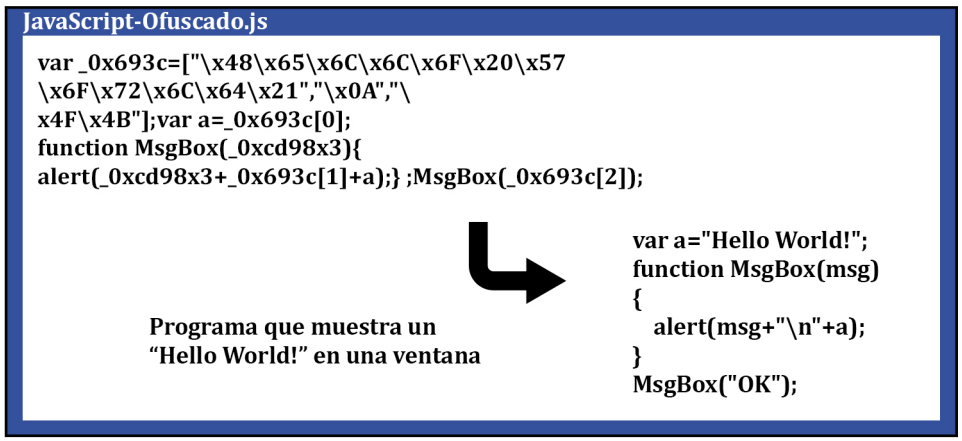

Figura 2.14: Análisis de un código ofuscado JavaScript

Análisis de un código ofuscado en Javascript. Se puede notar que son pocas las cadenas de texto legibles por humanos (figura 2.14). Determinar la semántica de un código (el que realiza) es sumamente complicado, por este motivo la mayoría de las herramientas de protección bloquean en su totalidad programas de Javascript. [32]

Otra mejora que se puede agregar a la técnica de ofuscación mencionada es la adición de bloques básicos al de flujo de control del programa, algunos de los cuales nunca van a ser ejecutados, pero lo cual es muy difícil de determinar mediante análisis estáticos, debido a que las llamadas se generan de manera dinámica durante la ejecución del programa. Entonces se agregan funciones de cargas hacia esos bloques inalcanzables a través de punteros, lo cual tiene el efecto de confundir a los análisis estáticos sobre el posible valor que asignó el despachador de variables.

V. Técnicas de desofuscación

Así como existen varias técnicas y mejoras para la ofuscación del código, también existen técnicas para lograr desofuscar el código, o mejor dicho, técnicas de ingeniería inversa sobre el código ofuscado algunas de las cuales se describen a continuación.

Clonación: Muchas de las técnicas de ofuscación se basan en la creación de rutas de ejecución falsas para engañar a los programas de análisis estáticos, debido a que esas rutas nunca van a ser tomadas en el momento de ejecución del programa, mas sin embargo, causan información engañosa que se propagará a lo largo del análisis y reducirá la precisión de la información obtenida, lo cual hará más difícil de entender la lógica del programa analizado.

Esa información defectuosa tiene la característica de introducir imprecisiones donde los caminos de ejecución se cruzan. Una forma de resolver este problema o desofuscar el código es mediante la clonación de porciones de código, de tal manera que las rutas de ejecución falsas no se unan con la ruta de ejecución original.

Esta transformación tiene que ser aplicada con sumo cuidado, debido a que si no se hace de esta manera, puede incrementar el tamaño del código, y más aun que lograr desofuscar el código, empeore el problema de desofuscación, sin embargo como la meta de la desofuscación del código es la de remover código ofuscado, la clonación implica que se tenga un conocimiento previo de la ruta de ejecución actual, lo cual podría lograrse mediante la clonación selectiva en ciertos puntos donde las rutas de ejecución se unen.

Análisis de factibilidad de ruta estática: Este es un análisis estático basado en cadenas, que evalúa la factibilidad de una ruta de ser ejecutada.

En este análisis primeramente se debe crear una cadena que corresponda con la ruta de ejecución, la construcción de esta cadena se puede realizar de varias maneras, pero la meta, es que tome en cuenta los efectos de las operaciones aritméticas sobre los valores de las variables de los bloques de código, y que nos represente la propagación de constantes a través de una ruta de ejecución del programa, y después se evalué si esa ruta de ejecución es factible o no.

Algunas de las formas de crear las cadenas anteriormente descritas son las siguientes: Asignación, aritmética, indirección y bifurcaciones.

Combinación de análisis estáticos y dinámicos: Análisis estáticos convencionales son inherentemente conservadores, por lo tanto al aplicar técnicas de desofuscación meramente estáticas al código ofuscado, el conjunto de resultados son en realidad un su-

perconjunto de ese código ofuscado, inversamente a lo que resulta si se aplican técnicas de desofuscación dinámica, que no pueden tomar todos los posibles valores de entrada, por lo cual este análisis solamente nos regresa un subconjunto del código ofuscado.

La naturaleza dual de estos dos tipos de análisis sugieren que se debe intentar realizar un análisis que abarque estas dos aproximaciones de solución para desofuscar un código, lo cual se puede realizar de dos maneras, primero realizando un análisis dinámico y después uno estático para agregar partes al control de flujo del programa, o se puede realizar de manera contraria, empezando primero con el análisis estático y después el dinámico, lo que nos servirá para reducir partes del control de flujo del programa.

De cualquier manera, cuando se realiza una combinación de estos dos tipos de análisis no se puede garantizar la precisión, sin embargo si lo vemos desde el punto de vista de la ingeniería inversa, al unir estos dos tipos, nos podemos sobreponer a las limitaciones que nos causan el hecho de realizar un análisis estático o uno simplemente dinámico.

2.1.4. Aplicaciones de ofuscación. Caso práctico.

Técnica de ofuscación de *email* para prevenir SPAM

A continuación también se explica como usar la ofuscación para evitar que los *spiders* o rastreadores de la red recojan los *emails* de las páginas *webs*. Existen diversas técnicas ya sea mediante CSS [33] o con código Javascript.

CSS Método #1

Una técnica sencilla pero que da buenos resultados es escribir la dirección *email* al revés y luego con estilos mostrarlo en la dirección correcta utilizando la propiedad *direction*, por ejemplo:

```
<span style="direction:rtl; unicode-bidi:bidioverride;">
    moc.oinimod@otcatnoc
</span>
```

Entonces creamos una función que se encargue de utilizar este método para mostrar los *emails* en nuestra *web*:

```
function hideEmail(\$mail){
    \$mail = strrev(\$mail);
    return "<span style=\"direction:rtl;
    unicode-bidi:bidi-override;\">" . \$mail."</span>";
}
```

CSS Método #2

Otro método consiste en agregarle un texto oculto dentro de la dirección *email*, de esta forma la dirección se visualizará correctamente pero si un *spammer* copia el *email* esta

contendrá un dirección inválida. Este texto se oculta con estilos utilizando la propiedad *display*.

```
contacto@<span style="display:none">null</span>dominio.com
```

Luego creamos la función en PHP para ofuscar los *emails*:

```
function hideEmail(\$mail) {
    \$temp = explode("@", \$mail);
    return
    \$temp[0]."@<span style=\"display:none;\">null</span>" . \$temp[1];
}
```

Método ROT13

Este método consiste en codificar la dirección *email* con el algoritmo ROT13 [34] que consiste en trasladar 13 posiciones las letras del alfabeto, dejando los caracteres no-alfabéticos sin modificar. Para ello escribimos el *email* codificado y lo mostramos correctamente con una función Javascript.

```
<script type="text/javascript">
    document.write(("pbagnpgb@qbzvavb.pbz").replace(/[a-z]/gi,
    function(s){
        return String.fromCharCode(s.charCodeAt(0) +
        (s.toLowerCase()<'n?'a':-13));
    }));
</script>
```

Luego escribimos una función en PHP que se encargue de codificar la dirección *email* y que imprima el código Javascript para decodificarlo:

```
function hideEmail(\$mail) {
  \$temp ="<script type=\"text/javascript\">\n";
  \$temp.="document.write((\"".str_rot13(\$mail)."\").replace(/[a-z]/gi,
  function(s){ \n"; \$temp.="return String.fromCharCode(s.charCodeAt(0)+
  (s.toLowerCase()<'n?'a':-13));\n"; \$temp.= "}));\n";
  \$temp.= "</script>"; return \$temp;
}
```

2.1.5. Contramedidas

Anteriormente en los conceptos previos, se ha visto la definición de *shellcode*, pero aún no se ha visto como intentar mitigar su maleza.

Contramedidas para la mutación de *shellcode*

Existen varios métodos que pueden detectar *shellcode* polimórfico. únicamente mencionaremos los más conocidos junto con alguna referencia donde se pueda encontrar más información al respecto.

Por ejemplo "Buttercup" se trata de un método de detección de polimorfismo a nivel de red. Analiza el rango que es usado en la dirección de retorno para detectar *shellcode* polimórfico que afecte a la vulnerabilidad de desbordamiento del *buffer*. No es escalable, a pesar de que su precisión es alta - el falso positivo es de tan sólo el 0,01 % y la tasa de falsos negativos es del 0 %.

También hay métodos que usan redes neuronales para clasificar instrucciones desensambladas, intentando analizar las posibles rutas de ejecución de la carga útil a la vez que la tasa de repetición de las instrucciones seleccionadas. La tasa tanto de falso-positivo como de falso-negativo es muy baja en este método, puede ser incluso incapaz de detectar un código *shell* de día cero si el código *shell* no sigue estrategias de evasión comunes, tales como bucles de descifrado, instrucciones basura, etc.

La detección de *shellcode* polimórfico a nivel de red simula la ejecución de *shellcode* en un emulador de CPU. Este enfoque trata el flujo de *bytes* de entrada como los códigos de instrucción e intenta ejecutarlos. La detección es factible debido a la ejecución de *bytes* aleatorios en la entrada y por lo general se detiene, enviando una alerta, antes que otros métodos de detección (por ejemplo, debido a que encuentra un código de instrucción ilegal).

Un posible mecanismo de detección es si se encuentra un excesivo número de lecturas de memoria dentro de una pequeña región (esto podría significar haber encontrado el proceso de descifrado). Por otra parte, la emulación puede ser incapaz de analizar en profundidad *shellcodes* complicados ya que supone un incremento en el consumo de rendimiento y quizás el sistema de detección no pueda soportarlo.

El método, Spector también emula simbólicamente la ejecución de *shellcode* e investiga su comportamiento, analizando así, la ejecución de ciertas instrucciones o llamando a ciertas interfaces de programación de aplicaciones (API) que ayuden en el análisis. A pesar de que se realiza un exhaustivo análisis en la emulación, los atacantes pueden evadir la detección con algunas técnicas avanzadas, como el ataque de escaneo de memoria "memory scanning".

2.1.6. Técnicas de detección

Para la detección de *malware* hay varias técnicas que se pueden aplicar e incluso la combinación de varias de ellas. En esta sección haremos un resumen de las estrategias que por ejemplo un antivirus o detector de *malware* suele emplear.

- Basado en firmas
- Uso de heurística
- Emulación de código

Empezaremos por el método de detección basado en firmas (también conocido como vacunas). Este emplea una base de datos generada por el fabricante que permite determinar al *software* si un archivo es o no una amenaza. El sistema es sencillo: se coteja cada archivo a analizar con la base de datos y, si existe coincidencia (es decir, existe en la base una firma que se corresponde con el archivo), se identifica como código malicioso.

Además de la necesidad de mantener actualizada la base de datos, este método posee otras dos desventajas:

- El programa no puede detectar *malware* que no se encuentre en la base de datos
- El sistema debe contar con una firma por cada variante de un mismo código malicioso

En conclusión, la detección por firmas es un método de protección reactivo: primero se debe conocer el *malware* para que luego sea detectado. Para dar solución a esta problemática aparecen los métodos de detección proactivos basados en heurística, como complemento de la detección basada en firmas. Esto quiere decir que la detección proactiva es un agregado a la detección por firmas y para una óptima protección son necesarios ambos métodos, tal como trabajan las soluciones anti-*malware* en la actualidad.

Por lo general la programación heurística es considerada como una de las aplicaciones de la inteligencia artificial y como herramienta para la resolución de problemas. Tal como es utilizada en sistemas expertos, la heurística se construye bajo reglas extraídas de la experiencia, y las respuestas generadas por tal sistema mejoran en la medida en que "aprende" a través del uso y aumenta su base de conocimiento. La heurística siempre es aplicada cuando no puedan satisfacerse demandas de completitud que permitan obtener una solución por métodos más especificos (por ejemplo la creación de una firma para un *malware* determinado).

De igual forma, con las tecnologías reactivas es impsoible cubrir la protección necesaria para las condiciones actuales de evolución de amenazas, ya que no es posible contar con todos lso códigos maliciosos que circulan por Internet, y tampoco se puede disminuir los tiempos de creación de firmas lo suficiente para asegurar protección total al usuario. Ante estas imposibilidades la aplicación de heurísticas en tecnologías antivirus ofrece cobertura y protección inteligente ante códigos maliciosos.

El análisis heurístico posee un comportamiento basado en reglas para diagnosticar si un archivo es potencialmente ofensivo. El motor analítico trabaja a través de su base de reglas, comparando el contenido del archivo conc reiterios que indican un posible *malware*, y se asigna cierto puntuaje cuando se localiza una semejanza. Si el puntuaje iguala o supera un umbral determinado, el archivo es señalado como amenaza y procesado de acuerdo con ello.

Existen varios tipos de heurística:

- **Heurística genérica:** Se analiza cuán similar es un objeto a otro, que ya se conoce como malicioso. Si un archivo es lo suficientemente similar a un código malicioso previamente identificado, este será detectado como "una variante de..."

- **Heurística pasiva:** Se explora el archivo tratando de determinar qué es lo que el programa intentará hacer. Si se observan acciones sospechosas, éste se detectará como malicioso.
- Por último existe una **heurística activa**, relacionado con el tercer método de detección conocido como emulación de código y se trata de crear un entorno seguro y ejecutar el código de forma tal que se pueda conocer cuál es el comportamiento del código. Otros nombres para la misma técnica son "sandbox" o "emulación".

Los algoritmos heurísticos de los antivirus deben ser evaluados, no sólo por sus capacidades de detección, sino también por su rendimiento y la cantidad de falsos positivos detectados. La mejor heurística es aquella que combina los niveles de detección con bajos (o nulos) falsos positivos.

Como resumen de lo visto anteriormente se extrae que hay que intentar contar con métodos reactivos combinados con la heurística antivirus, que a su vez debe proporcionar buenos índices de rendimiento y una baja tasa de falsos positivos. La combinación de todas estas variantes permitirá al usuario confiar en una solución efectiva para las necesidades actuales en materia de protección contra *malware*. El desafio es poder adelantarse a los creadores de códigos maliciosos, que están en constante movimiento y evolución.

2.1.7. Conclusión

En el presente trabajo, se ha explicado en que consiste la ofuscación y en las diversas técnicas que existen para lograrla. Se ha explicado por qué los *malware* más sofisticados hacen uso del polimorfismo y metamorfismo para intentar evadir los escáneres de los antivirus y que como tendencia futura se prevé que los atacantes concentren sus esfuerzos en el intento de conseguir atacar vulnerabilidades en dispositivos móviles y navegadores *web*. Los cuales hacen uso de información sensible como puede ser datos bancarios y direcciones de *email*.

2.2. Técnicas de evasión de NIDS

2.2.1. Introducción

La evasión de NIDS fue tratada por primera vez por Ptacek y Newsham en 1998 [35]. En este artículo, los autores resaltan dos grandes problemas en el procesamiento del tráfico de red por parte de los NIDS. En primer lugar, un NIDS no siempre es capaz de conocer con exactitud la manera en que los paquetes serán procesados en el sistema final, debido a los conflictos existentes en los protocolos TCP/IP. El segundo problema resaltado es que es posible realizar ataques de Denegación de Servicio (DoS) sobre los NIDS. Esta técnica se explicará posteriormente en la parte de las tipos de ataques.

Con el fin de explotar y estudiar los conflictos mencionadas anteriormente, se han diseñado algunas herramientas específicas, como Fragroute, la cual intercepta, modifica, y reescribe el tráfico destinado a un sistema [36], o IDSprobe, que genera tráfico modificado a partir de trazas originales [37]. Las modificaciones que estos sistemas proponen están orientadas a la generación de tráfico que provoque evasiones. De esta manera, se pueden realizar auditorías sobre NIDS haciendo uso de técnicas evasivas existentes. Alternativamente a estas herramientas, se pueden utilizar algunas funcionalidades de programas más

conocidos como Nikto [38] o Nmap [39], cuyos parámetros de ejecución permiten modificar los paquetes que se envían para intentar evitar la detección por parte de los NIDS.

La literatura existente sobre técnicas que prevengan a los NIDS ante posibles evasiones se centra en el tratamiento del tráfico que le llega al NIDS, con el fin de paliar los conflictos en la interpretación de los protocolos de comunicaciones y establecer un procesamiento común. Watson et al. [40] proponen un sistema para depurar paquetes de datos, de manera que se eliminen los posibles intentos de evasión de los NIDS, e implementan un depurador (*scrubber*) sobre el protocolo TCP para generar tráfico bien formado a partir de uno de entrada que pueda contener conflictos. Por su parte, Handley et al., proponen el concepto de normalizadores de tráfico [37]. Éstos son elementos que se sitúan como intermediarios en la red y que toman el tráfico antes de que llegue al NIDS con el fin de eliminar los posibles conflictos que puedan ocurrir. Dado que algunos de los problemas que pueden llevar a evasiones se dan en el reensamblado de los paquetes, estos normalizadores deben almacenar el estado y los paquetes previos de todas las conexiones entrantes, lo que requiere una gran capacidad de almacenamiento, ya que deben verificar la consistencia de los paquetes subsecuentes.

Esta tarea consume una gran cantidad de recursos (tanto de procesamiento como de almacenamiento) y puede convertirse en un cuello de botella cuando se trabaja con redes de alta velocidad [41]. También se han propuesto algunas soluciones alternativas que no precisan de la modificación del tráfico. Varghese et. al. [42] plantean dividir la firma que busca el NIDS para detectar el ataque (NIDS *signature*) en pequeños fragmentos, de manera que cualquier paquete que contenga cualquiera de los fragmentos sería detectado de forma rápida, pasando después el conjunto de paquetes a un análisis más lento pero más profundo y por lo tanto más eficaz. Shankar y Paxson [43] proponen un sistema que notifica al NIDS la topología de red y el comportamiento (entendiéndose como tal la implementación de los protocolos) del sistema final que se monitoriza, de manera que pueda modificarse la configuración del NIDS para ajustarse a la información proporcionada por su sistema. A este respecto Snort [44], uno de los NIDS de código abierto más utilizado en la actualidad, implementa una técnica similar a esta en el preprocesador IP (frag3).

Finalmente, Antichi et al. [35] proponen el uso de *Bloom Filters*, una suerte de filtros distribuidos que consiguen que la firma del NIDS sea detectada sin necesidad de un reensamblado previo de los paquetes. Con este sistema se minimizan los falsos negativos pero se incrementa sobremanera la carga computacional del NIDS. En lo referente a la búsqueda de evasiones en NIDS, cabe destacar el trabajo de Mutz et al. [45], en el cual los autores proponen realizar ingeniería inversa (tomando un conjunto de entradas/salidas y analizando el comportamiento de los ficheros binarios de un NIDS comercial) con el fin de obtener información acerca de las firmas que aplican los NIDS cuyo comportamiento no es conocido al ser el código propietario. Una vez analizado su funcionamiento, los autores presentan un ejemplo de evasión con un ataque a servidores Apache. Su trabajo se centra en la evasión de las firmas, las cuales son el último escalón en la arquitectura de un NIDS, sin tener en consideración el preprocesamiento que pueda realizarse anteriormente. Sin embargo, las técnicas evasivas propuestas por Ptacek y Newsham [46] se basan precisamente en conflictos de este preprocesamiento.

2.2.2. Evasión de la detección de patrones

Gran cantidad de IDS comerciales emplean bases de firmas como motor de análisis para la detección de patrones. Es decir, cuando se recibe un evento en el motor de análisis se compara contra toda la base de firmas. Si alguna de ellas coincide, se disparan las alarmas y se genera un evento que será almacenado para su posterior revisión.

La tecnología de firmas ha sido heredada de los motores antivirus que las llevan empleando desde sus inicios. No obstante ha quedado demostrada la facilidad que existe a la hora de burlar dichos sistemas de firmas. Simples modificaciones en los contenidos de los paquetes o la fragmentación de los mismos sirven para que numerosos ataques pasen desapercibidos al motor de firma, y es que a pesar de ser una herramienta esencial dentro de los sistemas de seguridad presentan importantes limitaciones y problemas.

Uno de los inconvenientes más importantes es el de las falsas alarmas: falsos positivos y falsos negativos. Los falsos positivos consisten en aquellas alarmas que se generan cuando en realidad no se está produciendo ningún ataque, intrusión o abuso. Dentro de los sistemas de detección por firmas o patrones, es frecuente que una firma (que en muchos casos no es más que una sucesión de *bytes*) aparezca dentro del flujo de tráfico lícito y levante una alarma cuando no existe el ataque. Por otro lado los IDS basados en anomalías presentan también un problema similar. Pueden detectar como tráfico hostil la aparición de un nuevo tipo de tráfico generado, por ejemplo, por la instalación de un nuevo servicio.

El otro tipo de falsa alarma son los falsos negativos y tienen que afectan precisión del propio IDS, ya que se trata de ataques reales que no han podido ser detectados. Otro problema importante que se deriva, es la excesiva cantidad de eventos o alarmas (en un porcentaje alto de falsos positivos) que se pueden generar en una red de mediano tamaño. Lo más negativo de esta cuestión es que la continua aparición de falsos positivos puede hacer que un administrador acabe ignorando las alarmas.

Así mismo, cada vez es más común el uso de la criptografía para proteger las comunicaciones. Esto establece un nuevo problema para los IDS puesto que se imposibilita la detección de los paquetes cifrados y por tanto se abre un nuevo vector para ataques inadvertidos.

2.2.3. Tipos de ataques para la evasión de NIDS

A. Ataques de Inserción

Debido a la carencia de información de la que dispone, un NIDS puede aceptar un paquete pensando erróneamente que el paquete ha sido aceptado y procesado en el sistema final cuando realmente no lo ha hecho. Un atacante puede explotar esta condición enviando paquetes al sistema final que rechazará, pero que el IDS pensará que son válidos. Haciendo esto, el atacante estará "insertando" tráfico en el IDS.

Para entender porque la inserción hace fallar el análisis de firmas, es importante entender la técnica empleada por los IDS. El análisis de firmas se usan algoritmos de patrones para detectar cierta cadena de *bytes* en un flujo de datos. Por ejemplo, para evitar un ataque de "*transversal directory*" un IDS intentará detectar una cadena como "../" en la URL de una petición http. De esta forma, es normal que el NIDS genere una alerta cuando

encuentre una petición *web* como "GET /carpeta/../../../etc/passwd". ¿Pero qué ocurriría si el intruso consigue inyectar paquetes que no serán aceptados en la máquina atacada pero que el IDS si procesará?. Pues conseguiría que el NIDS ve una petición diferente y no detectara el ataque:

```
GET /carpeta/ ..AAAA/..AAAA/..AAAA/etc/pasAAAswd
```

El intruso ha usado un ataque de inserción para añadir los caracteres 'A' al flujo original. El IDS ya no detectaría el ataque porque no encuentra literalmente la cadena "../" que era su patrón para encontrar este tipo de ataques.

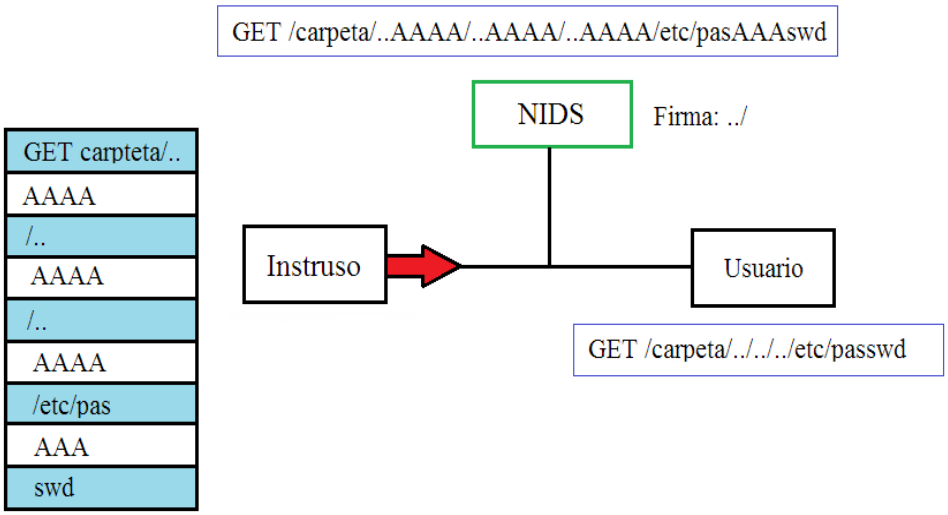

Figura 2.15: Ejemplo de ataque de inserción

Como vemos en el diagrama anterior (figura 2.15), la petición del intruso se fragmenta en trozos más pequeños e intercala paquetes especialmente formados (azul oscuro) que serán procesados por el IDS pero no por la máquina objetivo. Con éste ataque de inserción el NIDS es incapaz de encontrar la firma en la petición http y por tanto no detecta el ataque.

En general, los ataques de inserción ocurren cuando el IDS es menos estricto en el procesamiento de paquetes que la máquina atacada. Por lo tanto una primera solución para evitar este tipo de ataques podría ser hacer el IDS lo más estricto posible en el procesamiento de paquetes, pero como veremos más adelante, esta opción genera otro tipo de problemas.

B. Ataques de Evasión

Son muy similares a los anteriores ataques de inserción, pero sucede la situación contraria: Una máquina de la red puede aceptar un paquete que el NIDS desecha erróneamente perdiendo todo su contenido. Información que será vital para la correcta detección de la alerta. Esto es debido a que el procesamiento de paquetes del NIDS es más restrictivo que

el qué realiza la máquina final.

Estos ataques de evasión pueden tener un impacto de seguridad importante ya que sesiones enteras pueden ser forzadas a evadir un NIDS y por consiguiente que no se detecte ni la más mínima alerta a nivel de red del ataque.

Como vemos en la figura siguiente, los ataques de evasión burlan la búsqueda de patrones de una manera similar a los ataques de inserción. De nuevo, el atacante que consigue que el IDS vea un flujo de datos diferente al del sistema final.

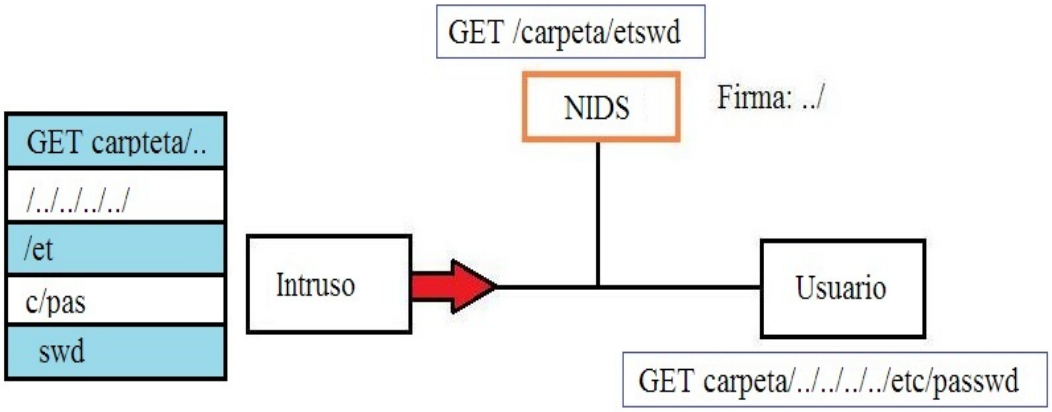

Figura 2.16: Ataque de evasión

La petición HTTP, de nuevo, la fragmentamos en distintas partes. Los paquetes azul oscuro evaden el NIDS y no son procesados por el monitor, en cambio la máquina objetivo procesa todos los paquetes normalmente. (Figura 2.16)

C. Denegación de Servicios (DoS)

Otro de los ataques más populares, son los ataques de denegación de servicio (DoS) que intenta comprometer la disponibilidad del un recurso. Entre los más conocidos se encuentran los SynFloods, MailBomb, PingFlod y el famosísimo "Ping de la Muerte". Todos estos ataques intentan consumir desproporcionadamente montones de recursos para que los procesos legítimos no los puedan utilizar. Otros ataques de DoS están orientados a *bugs* en el *software* que al explotarlos pueden llegar a bloquear el sistema.

Cuando un dispositivo de red recibe un ataque de denegación de servicio, pueden ocurrir dos cosas: que el dispositivo deje de dar servicio bloqueando la red, que sería el caso más común de un *router* o un buen *firewall*. O bien, la otra posibilidad es que el dispositivo de red deje de dar servicio pero permita completamente que cualquier tipo de trafico pase a través de él, este es el caso de los NIDS. Son sistemas pasivos que no pueden actuar sobre la red y por lo tanto en caso de recibir un ataque de DOS actuarían en modo "*fail-open*". Si un atacante consigue bloquear el IDS o sus recursos, tiene total libertad para realizar ataques a la red sin peligro de ser detectado por los NIDS.

Algunas denegaciones de servicio surgen de un error en el *software*: Un IDS falla al recibir cierto paquete o una serie de mensajes. Afortunadamente, este tipo de errores son rápida y fácilmente corregidos por los fabricantes. Es también interesante hablar de los IPS, que puede ser inundados con falsos positivos (paquetes maliciosamente malformados) para que reaccionen contra ataques que nunca han ocurrido y bloqueen el tráfico de ciertas máquinas consiguiendo un ataque de DoS.

Hay muchos tipos diferentes de DoS que afectan a los sistemas de detección. Por lo general estos ataques involucran el agotamiento de algún recurso. El atacante identifica algún punto de la red que requiere la asignación de algún recurso y causa una condición que consume todo el recurso:

Agotamiento de CPU: Consiste en el agotamiento de las capacidades de procesamiento de un IDS, haciéndole "perder el tiempo" en trabajo inútil y así evitar que realice otras operaciones. Un ejemplo concreto es la fragmentación: un atacante puede forzar un IDS a procesas gran cantidad de fragmentos incompletos y lo más pequeños posibles para consumir ciclos de CPU.

Agotamiento de la Memoria: Un atacante puede determinar que operaciones requieren una mayor utilización de la memoria y forzar al IDS para asignar toda la memoria a información inútil para facilitar algún tipo de ataque. Un IDS necesita memoria para una gran cantidad de cosas: almacenar el estado de conexiones TCP, reensamblado de fragmentos, *buffers* necesarios para la detección de firmas, etc.

Agotamiento de Disco: En algún punto, ya sea en fichero, BD o en algún otro formato, el IDS necesitará almacenar los logs y las alertas del tráfico de red. Un atacante puede crear un flujo de eventos sin importancia que continuamente sean almacenados en el disco y poco a poco ir completando el espacio del disco necesario para almacenar eventos reales.

Agotamiento de Ancho de Banda: Los sistemas de IDS deben llevar un seguimiento de toda la actividad de las redes que monitoriza. Pocos IDS pueden hacerlo cuando las redes están extremadamente llenas de carga. Un IDS, a diferencia que una máquina final, deben leer todos los paquetes de la red, no solo los destinados a ella. Un atacante puede sobrecargar la red y evitar que el IDS detecte correctamente todo lo que ocurre en la red. Este tipo de ataques son extremadamente difíciles de defender. La sobrecarga de recursos no es fácil de solventar y hay muchos puntos diferentes en los que los recursos de un IDS pueden ser consumidos.

D. Inundación de Alertas

Otro tipo de ataque contra la precisión y eficacia de los NIDS, es la inundación de alertas con el objetivo de enmascarar y ocultar el verdadero ataque. Un intruso puede conseguir fácilmente que un NIDS genere gran cantidad de alertas de falsos ataques mientras se realiza el verdadero ataque que solo genera unas pocas alertas. El objetivo es que la verdadera alerta pase inadvertida para el administrador o la tome como un falso positivo al igual que el resto de alertas falsas. Otra implicación que puede tener la inundación de alertas es la del agotamiento del ancho de banda, antes comentado, que puede mermar la capacidad del NIDS para detectar el verdadero ataque.

E. Ataques de *spoofing*

El término *spoofing* hace referencia al uso de técnicas de suplantación de identidad generalmente con objetivos maliciosos Existen diferentes tipos de *spoofing* que dependerán del entorno donde se produzca. Algunos ejemplos pueden ser el IP *spoofing* (quizás el más conocido), ARP *spoofing*, DNS *spoofing*, *web spoofing* o e-mail *spoofing*, etc.

Como se ha dicho el IP-*spoofing* es una de las técnicas más conocidas, debido al desafortunado hecho de que el protocolo Ipv4 no tiene forma de autenticación: cualquier paquete puede provenir de cualquier host. Esto supone algunos problemas para el IDS que puede ser engañado al analizar solamente la información basada en las cabeceras IP y creer algo que realmente no está pasando en la red. Este tipo de tipo de ataques tiene un especial impacto en los protocolos no orientados a conexión donde no existe ninguna clase de código de control de secuencia.

Hoy en día hay que tener en cuenta que los *routers* actuales no admiten el envío de paquetes con IP origen no perteneciente a una de las redes que administra (los paquetes spoofeados no sobrepasarán el *router*) ni tampoco aceptan paquetes entrantes con IP origen alguna de alguna de las redes internas. Estas dos características mitigan notablemente la posibilidad de realizar ataques de IP-*spoofing*.

2.2.4. Ataques en la capa de red

A. Cabeceras malformadas

Para realizar ataques de inserción simples el intruso debe generar paquetes que se rechacen en la máquina objetivo pero que sean procesados por el NIDS. La forma más sencilla de que un datagrama sea descartado es que tenga una cabecera no válida. Como a priori el objetivo del NIDS no es el validar los campos de un paquete, puede ser relativamente sencillo que procese paquetes malformados sin percatarse de que no son del todo correctos.

La definición de la cabecera IP y de todo el protocolo lo podemos encontrar en el RFC791 (http://www.ietf.org/rfc/rfc0791.txt).

El problema de intentar usar paquetes con cabeceras malformadas en ataques de inserción, es que dichos paquetes no van a ser reenviados por los *routers*, que detectarán los paquetes como erróneos. Por esta razón se hace complicado utilizar éste tipo de técnica para engañar a los NIDS a menos que nos encontremos en la misma red local que el analizador de red.

Un ejemplo de ataque de inserción consistiría, por ejemplo, en intercalar paquetes con el *checksum* erróneo en el flujo de datos. Si el NIDS no comprueba la validez de este campo procesará más datagramas que la máquina destino viendo un flujo de datos distinto.

B. Ataques basados en TTL

El campo TTL dentro de la cabecera IP indica cuantos saltos puede realizar el paquete para llegar a su destino. Cada vez que el paquete pasa a través de un *router*, éste

decrementa en uno el valor del campo TTL. Si el valor llega a cero el paquete es descartado.

Existe un tipo de ataque basado en el TTL, que puede ser tanto de inserción como de evasión, e intenta aprovechar el problema de la diferente localización en la red del NIDS y la máquina atacada. Se verá más claro con el siguiente ejemplo de red (figura 2.17):

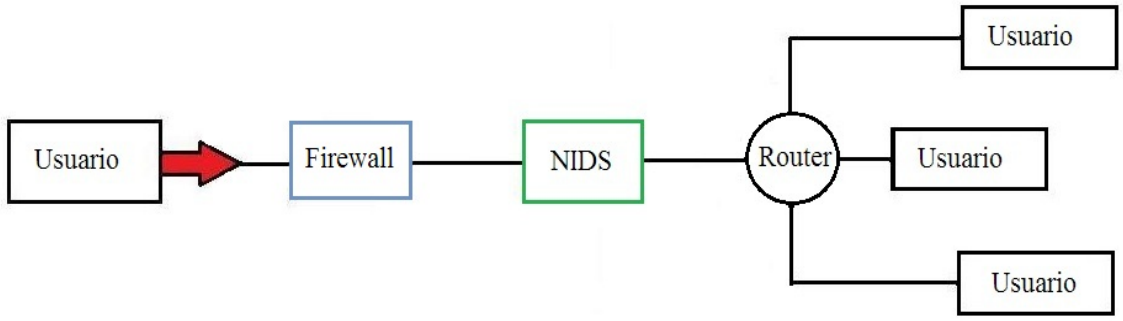

Figura 2.17: Ataque basado en TTL

Por supuesto para realizar este ataque es necesario un conocimiento previo por parte del atacante, de la topología de red; aunque pueden ser especialmente útiles herramientas básicas como traceroute o algo más complejas como paratrace para extraer información acerca de la topología.

Supongamos un caso ficticio de la empresa SA. De dicha empresa sabemos que tiene un *firewall* protegiendo la red interna del exterior y que solo presenta dos tipos de servicios a Internet: Servicio *web* y servicio DNS (*Domain Name Service*) de la zona "sa-enterprise.com". Por tanto los únicos puertos abiertos al exterior serán el 53 UDP y el 80 TCP. Sabemos también que utiliza direccionamiento público para todas las máquinas de su red.

El objetivo del ataque será el servidor *web* y se intenta explotar una vulnerabilidad en la aplicación *web* www.sa-enterprise.com, por lo que dicho ataque podrá ser realizado a través del puerto 80 sin que el *firewall* sea un impedimento. El único problema que encuentra el intruso, es que desconoce si existe algún otro dispositivo de análisis de tráfico como un NIDS que pueda detectar el ataque. El intruso antes de arriesgarse a atacar y ser detectado intenta recabar más información acerca de la red.

Esta información puede ser obtenida de muy diferentes fuentes, pero en el ejemplo existe un problema grave en el servicio DNS (ya sea por configuración, *software* no actualizado o vulnerabilidades) que le permite dicha información. El ejemplo más catastrófico sería que el servidor DNS permitiera al atacante realizar una transferencia de zona y obtener todos los nombres de dominio de la zona "sa-enterprise.com".

Una vez obtenida esta información el intruso observa un nombre que llama la atención "ids.sa-enterprise.com" Para encontrar la situación exacta de estas dos máquinas en la red, una posibilidad que tiene es realizar un trazado de ruta utilizando paquetes UDP con puerto destino 53 (como el servicio DNS) ya que en el *firewall* está permitido el tráfico de estas características. Se podría maquillar aun más el trazado, utilizando paquetes con

peticiones DNS o HTTP reales pero para el ejemplo no será necesario.

Para realizar el trazado el intruso utiliza una potente herramienta de generación de paquetes: HPING (http://www.hping.org/). Los comandos ejecutados serían:

```
hacker# hping ids.hall-enterprise.com --traceroute --udp --destport 53
hacker# hping www.hall-enterprise.com --traceroute --udp --destport 53
```

Una vez realizadas distintas pruebas, el intruso llega a la conclusión que el NIDS y la máquina objetivo no se encuentran en la misma red y que hay uno o varios *routers* situados en medio. Es posible entonces generar paquetes que solo verá el NIDS usando el valor TTL justo para que sea procesado por el analizador y que llegue a cero antes de alcanzar la máquina objetivo. (Figura 2.18)

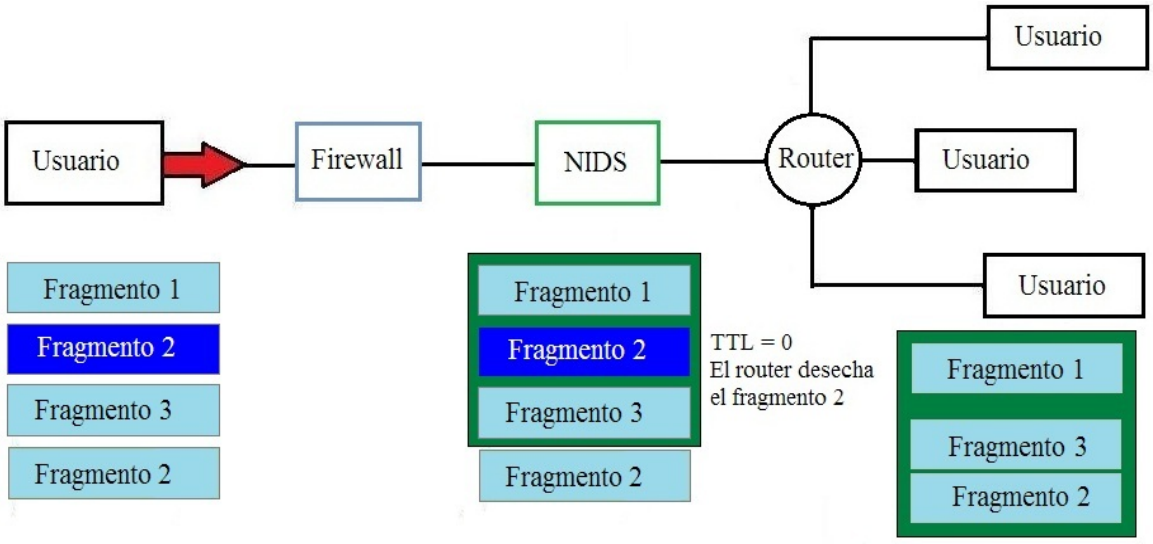

Figura 2.18: Ataque de inserción usando TTL

Por supuesto, esta ha sido una situación ficticia donde han coincidido varios problemas graves de seguridad que han permitido el ataque. Con una correcta política de cortafuegos, una buen 'configuración del NIDS, un diseño de red seguro y un *software* actualizado y bien configurado serían prácticamente imposibles ataques de este tipo.

C. Opciones IP

Otro tipo de problema es el relacionado con las opciones de la cabecera IP. El intruso utilizando una combinación adecuada de estas opciones, es capaz de nuevo, de realizar ataques de inserción y evasión. Para evitar estos ataques, el NIDS debería analizar las opciones de cada paquete. Éste proceso adicional no es tan sencillo como verificar un *checksum* y supone un coste adicional de procesamiento por paquete, además de que no soluciona el problema en todos los casos. Aquí vemos algunas de las posibilidades que tiene un paquete para ser descartado antes de ser procesado por la máquina atacada:

Cualquier mala opción de longitud.
Opción de ruta origen: desplazamiento menor que 4.
Ruta origen estricta: Este 'host' no es uno de los saltos en lista.
Ruta origen: Este 'host' está configurado para dejar fuente encaminada.
Ruta origen: No hay rutas para el siguiente salto.
Opción registro de rutas: el desplazamiento es menor que 4.
Registro de ruta: No hay rutas en el registro de rutas para el siguiente salto.
Opción marca temporal: Demasiado corto.
Marca temporal: No hay suficiente espacio para guardar marca temporal y dirección IP.
Marca temporal: El registro de marca temporal está lleno y el contador de filas ha vuelto a 0.
Marca temporal: Marca temporal mal determinada.

Tabla 2.1: Posibilidades de que un paquete sea descartado

Un ejemplo es el uso de la opción *strict source routing* (encaminamiento estricto desde el origen) que especifica la ruta exacta salto a salto. Si la ruta no es posible se produce un error y el *router* desecha el paquete. Por lo tanto se pueden llegar a realizar ataques muy similares a los basados en TTL. Examinar las opciones de source route en paquetes IP parece ser la solución obvia, pero hay más opciones que hay que tener en cuenta y que son tan obvias de procesar.

Otro ejemplo de ataque puede ser la opción *timestamp* que indica que cada *router* guarde en el paquete el instante en el que el paquete pasa por él. Si no hay suficiente espacio para añadir ese dato de 32 *bits*, el paquete será descartado por el *router*.

Un problema similar ocurre con el bit DF de la cabecera IP (*Don't fragment*). El campo DF dice a los dispositivos de enrutado que no deben fragmentar dicho paquete, cuando el paquete es demasiado grande para ser reenviado por la red, simplemente es desechado. Si el MTU en la red del NIDS es mayor que en el sistema destino, un atacante puede insertar paquetes haciéndolos demasiados grandes para llegar a la máquina objetivo pero lo suficientemente pequeños para viajar por la red del NIDS y que los procese. (Figura 2.19)

Como contramedida muchos de estos ataques se basan en la emisión de mensajes ICMP de error, informando al emisor de lo que acaba de ocurrir. Un IDS puede potencialmente analizar de dichos mensajes para intentar detectar estos ataques. Esto no siempre es posible e implica mantener el "estado" de cada paquete IP, lo cual supone un consumo de recursos muy importante y un consiguiente ataque de denegación de servicio contra el IDS. Además de que el NIDS en muchos casos no sabrá diferenciar entre un ataque y un error normal en la red.

D. Ataques de fragmentación

El tamaño máximo de un paquete viene definido por diseño de la red. Cuando un paquete supera la MTU de la red (maximium transfer unit) la única posibilidad para que el paquete viaje por la red es fragmentarlo en trozos más pequeños que serán reensamblados en el destino. A pesar de ser un mecanismo esencial del protocolo IP son bien conocidas las implicaciones de seguridad que tiene la fragmentación para dispositivos de análisis de

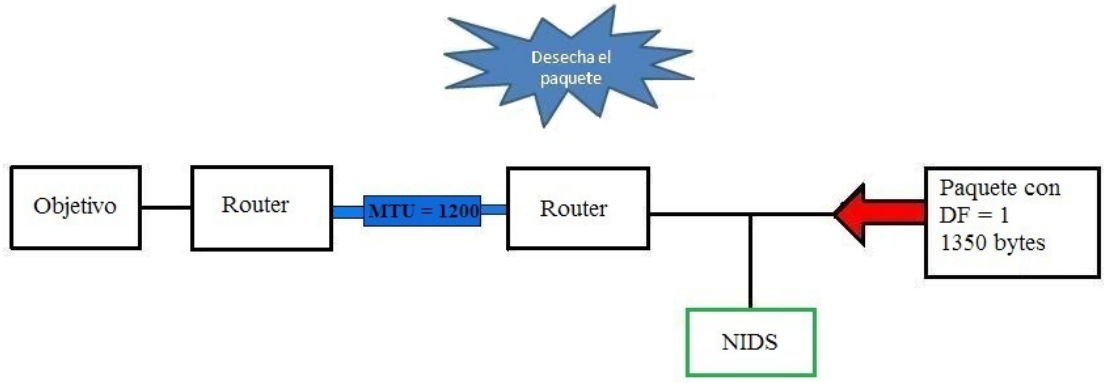

Figura 2.19: Ataque de inserción usando opciones IP

tráfico como *firewalls* o IDS.

A continuación se dan algunos ejemplos de ataques posibles contra NIDS basándonos en la fragmentación.

El ataque más básico que existe para evadir la detección de firmas por parte de un IDS consiste en fragmentar todo el tráfico en trozos muy pequeños asegurando que en cada datagrama individualmente no exista la cadena de *bytes* que identifica el ataque. Por supuesto, la utilidad de los NIDS se pondrían en duda si fueran vulnerables a un ataque tan básico. El analizador de red, debe ser capaz de reensamblar correctamente todos los fragmentos tal y como haría la máquina destino para poder analizar correctamente el flujo de datos.

Para poder realizar el reensamblado el IDS (o cualquier máquina) debe esperar para recibir todos los fragmentos de un paquete. Un NIDS puede ser atacado, enviándole infinidad de fragmentos parciales, que nunca llegarán a completar un paquete. El IDS esperando recibir todas las partes, almacenará en memoria cada fragmento que reciba, lo cual se convertirá en un potencial ataque de denegación de servicio (DoS) por consumo de memoria.

E. Timeout de fragmentación

Para evitar la situación anterior y que no se produzca un consumo de memoria, muchos sistemas desechan fragmentos basándose en sus tiempos de llegada. Un IDS eventualmente puede desechar fragmentos viejos e incompletos pero lo debe hacer de una forma consistente con las máquinas que vigila, o será vulnerable a ataques de inserción o evasión como veremos a continuación.

Imaginemos una red, donde el NIDS tiene un *timeout* de fragmentación más pequeño que el servidor que vigila. El intruso puede enviar una serie de fragmentos y esperar que el NIDS los deseche por tiempo límite de reensamblado, consiguiendo un ataque de evasión. Podemos encontrar también la situación contraria:

El tiempo de reensamblado en el NIDS es mayor que en la máquina objetivo, por lo que se pueden conseguir ataques de inserción esperando el tiempo exacto entre fragmentos

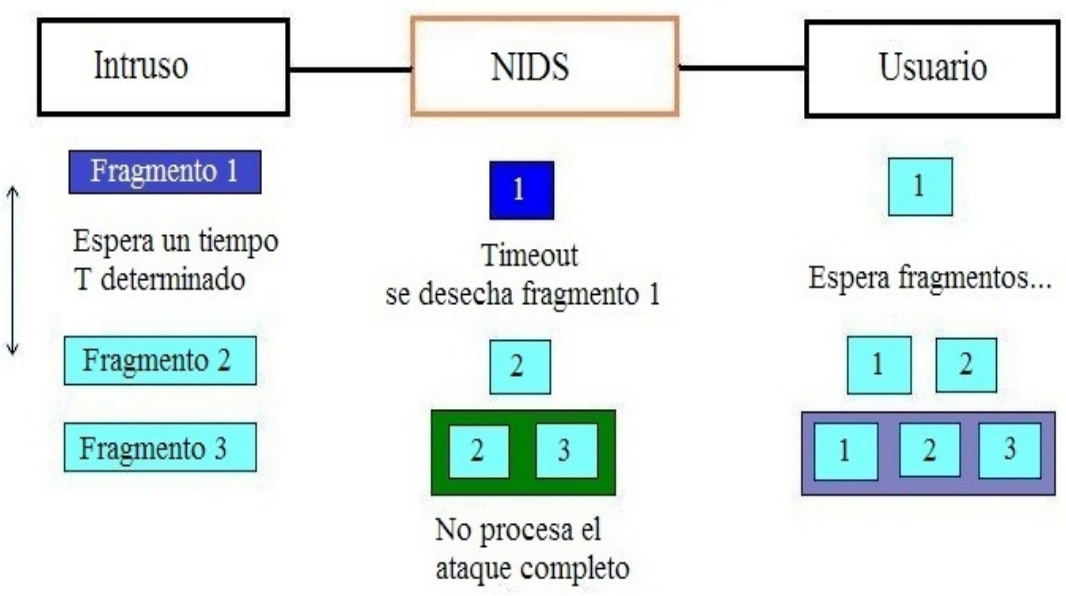

Figura 2.20: Ejemplo de Timeout de fragmentación

para que el servidor deseche algunos fragmentos. (Figura 2.20)

F. Superposición de fragmentos

Es segundo problema y que puede presentar un mayor impacto en los IDS de red, es la superposición de fragmentos. Cuando a un sistema le llega un fragmento IP con datos que ya tenía, pueden ocurrir dos cosas según la implementación del algoritmo de reensamblado que tenga. Una posibilidad es que la máquina sobrescriba con los nuevos datos que acaba de recibir, la otra es que, deseche esos nuevos datos y conserve los antiguos. Como decíamos esto presenta un problema importante ya que si el NIDS no maneja la superposición de fragmentos de la misma forma que las máquinas que vigila podrá ser vulnerable a ataques de evasión.

La solución a este problema es bastante complicada, cada sistema manejará la superposición de paquetes de una forma, a veces a favor de los nuevos datos y otras prevaleciendo los datos antiguos y desechando los nuevos. Por ejemplo, un Windows NT resuelve la superposición de fragmentos manteniendo las datos antiguos (no sobrescribe en caso de superposición). Esto difiere de los BSD y UNIX que tal y como sugieren los estándares sobrescribe con los datos nuevos que recibe.

En este ejemplo de red heterogénea vemos que el NIDS (ejecutandose en un sistema SUN) después de reensamblar no ve el mismo flujo de datos que otras máquinas (Linux, FreeBSD y HP). El resultado final es que el reensamblado de fragmentos es diferente en el sistema final dependiendo del sistema operativo. A menos que el IDS conozca que sistema operativo está ejecutandose en cada máquina y además conozca cómo maneja ese SO la superposición, técnicamente es vulnerable a ataques de superposición de fragmentos.

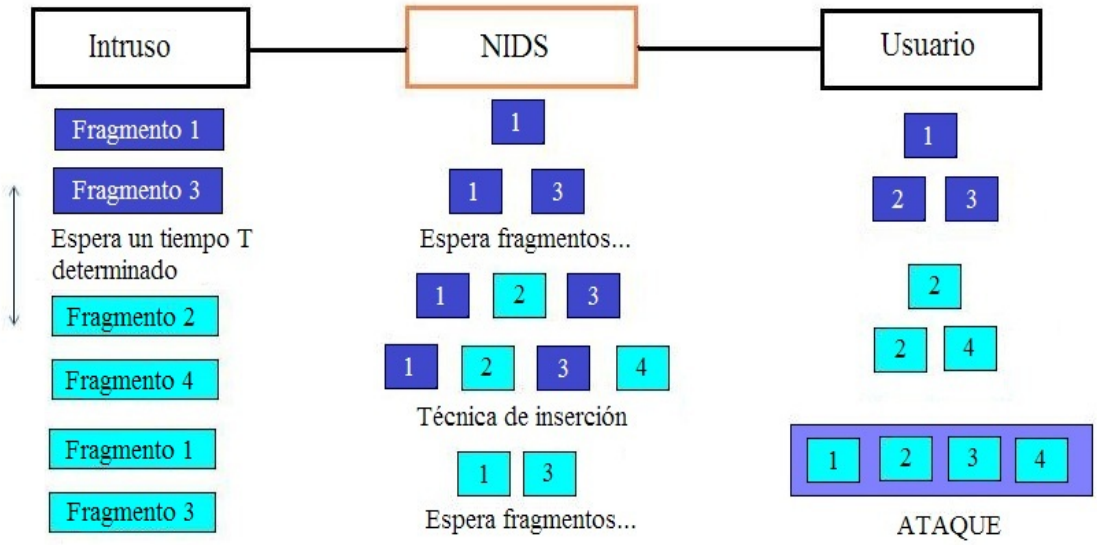

Figura 2.21: Ataque usando *timeout*

2.2.5. Ataques en la capa de transporte

En el nivel de transporte encontramos quizás uno de los protocolos más importantes de Internet: TCP.

Prácticamente todas las comunicaciones se realizan sobre conexiones TCP, por lo que es evidente que la gran mayoría de ataques se realizarán sobre este protocolo. Esto supone una serie de requerimientos para los sistemas de detección de intrusos a nivel de red. Los NIDS deben ser capaces de reconstruir toda la sesión TCP y analizar la información de envía a través de ella, tal y como la ven los sistemas que protege. Pero si no es capaz de ello, tal y como pasaba con el protocolo IP, el NIDS será vulnerable a cierto tipo de ataques.

A. Cabeceras malformadas

Normalmente en los sistemas de detección de intrusos, los segmentos TCP son reensamblados y extraídos sus datos sin verificar la mayoría de los campos de la cabecera. Esto hace peligrosamente sencillo realizar ataques de inserción contra el NIDS. El atacante podría insertar correctamente en un flujo de datos, segmentos con cabeceras malformadas, de forma que fuesen procesados por el NIDS pero descartados en la máquina destino. Por lo tanto es importante que un NIDS valide las cabeceras TCP antes de procesar sus datos.

Un buen lugar para empezar, es examinar el código fuente de la pila TCP/IP en algún sistema operativo, con el objetivo de encontrar todos los puntos donde un segmento TCP puede ser descartado o cuándo puede provocar la pérdida de conexión. Un campo fácilmente modificable para que un segmento TCP sea descartado es el de las opciones binarias o flags. Cierta combinación de opciones en ciertas situaciones no serán validas y el paquete simplemente no se procesará.

Otro ejemplo de ataque típico es que, muchas implementaciones TCP no aceptarán da-

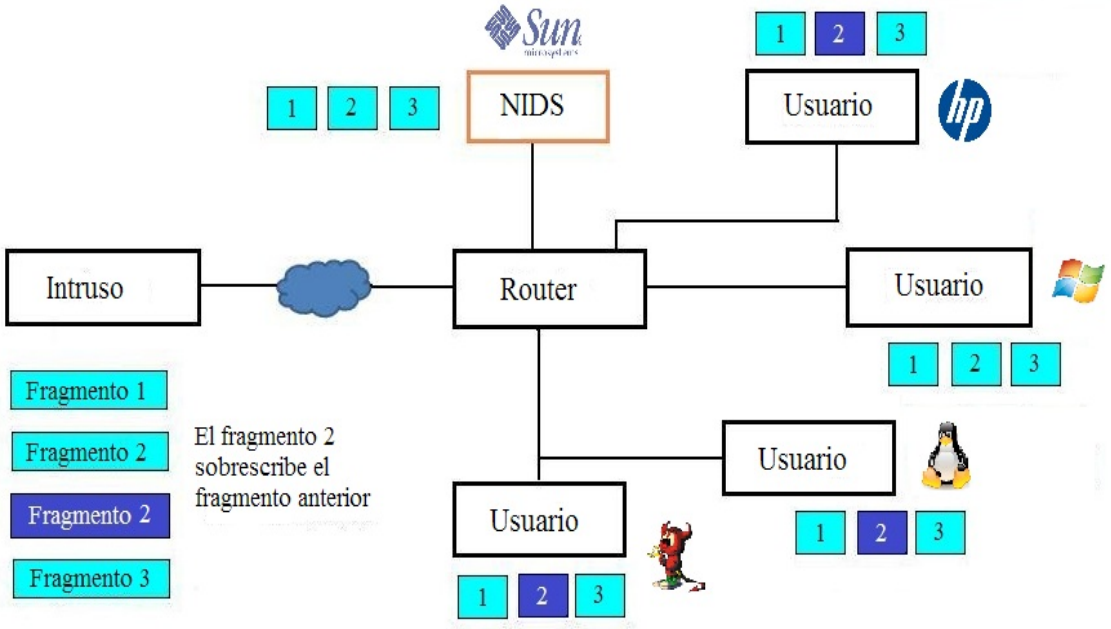

Figura 2.22: Paquetes procesados por cada sistema operativo según el ejemplo

tos en un paquete que no tiene bit ACK activo. De acuerdo con la especificación TCP, los sistemas deberían ser capaces de aceptar datos contenidos en un paquete SYN. La realidad es que al no ser una situación nada común, muchas implementaciones de la pila TCP/IP no tienen en cuenta este caso. Desde el punto de vista del NIDS, si descarta los datos de los paquetes SYN será vulnerable a un trivial ataque de evasión y si por el contrario, los tiene en cuenta pero las máquinas objetivo no lo implementan correctamente, el NIDS es vulnerable a ataque de inserción. Como vemos el gran problema suponen mantener la consistencia entre el analizador y los sistemas atacados.

Otro frecuente campo clave es el de *checksum*. Todas las implementaciones de TCP requieren comprobarla validez de los paquetes entrantes con el *checksum* de la cabecera. Pero sorprendentemente, muchos IDS no realizan esta comprobación, por lo que el sistema es vulnerable a la inserción de segmentos TCP con *checksum* erróneo, que por supuesto serán descartados en la máquina destino.

B. Opciones TCP

Tal y como pasaba con el protocolo IP, es importante que un IDS procese y analice las opciones de la cabecera TCP correctamente. Desafortunadamente, el procesamiento de las opciones TCP es significativamente más complicado que el procesamiento de las opciones IP, por lo que pueden haber más puntos por donde atacar a un NIDS. Una razón para esto son las muchas opciones TCP que han sido creadas como puede ser el TCP *timestamps* and Protection Against Wrapped Sequence Numbers (PAWS) que permiten mejorar el control de flujo descartando paquetes duplicados o segmentos antiguos con un *timestamp* (que llevará cada paquete) superior a un cierto umbral. Consultar RFC1323.

Otra razón de la complejidad de analizar las opciones TCP es que existen restricciones

concretas que hacen que algunas opciones sean inválidas en ciertos estados de conexión. Por ejemplo, algunas implementaciones TCP puede que rechacen paquetes no SYN con opciones que no se han especificado antes. Es importante que un IDS no acepte este tipo de paquetes de forma ciega, ya que seguramente serán rechazados en la máquina final y se producirá un ataque de inserción.

Por otro lado algunos sistemas puede que simplemente ignoren las opciones erróneas, pero continúen procesando el segmento. Si el IDS no conoce bien como se comportarán las máquinas que vigila será técnicamente posible realizar ataques de inserción y evasión.

Vemos como aprovecharlos de las nuevas opciones implementadas en el RFC 1323: Un atacante puede trivialmente crear un segmento TCP, con un *timestamp* muy bajo, que causa que el módulo PAWS de la pila deseche el paquete sin procesarlo. El NIDS cuando analice el tráfico deberá estar seguro que los segmentos que procesa no serán descartados en la máquina final por culpa del PAWS. El IDS no solo necesita saber que el sistema final suporta PAWS necesita conocer además el umbral para los *timestamp*. Sin esta información, un IDS puede erróneamente procesar segmentos TCP inválidos, siendo vulnerable a ataques de inserción.

Otra forma de aprovecharse de esta nueva funcionalidad es que el umbral de *timestamp* normalmente depende del *timestamp* del último paquete procesado correctamente que llega. Un intruso puede maliciosamente generar un segmento con un tiemstamp adecuado para modificar el umbral y que los próximos fragmentos que lleguen sean descartados a pesar de ser completamente correctos. Esto es posible ya que en muchos casos en la actualización de este umbral no se comprueba la validez del segmento (uso de números de secuencia no válidos).

C. Sincronización de NIDS

Cada conexión TCP la podemos identificar por cuatro variables que la distinguen de cualquier otra conexión de la red existente en el mismo instante. Estas variables son la dirección IP origen del cliente, el puerto origen, la dirección IP destino del servidor y el puerto destino. Son los llamados parámetros de conexión y no podrán existir a la vez en la misma red dos conexiones con estos mismos parámetros.

Una sesión TCP va a poder estar en diferentes estados, en el siguiente grafo (figura 2.23) vemos como se realiza esta transición:

D. Creación del BCT

Es un punto crítico. La forma que tiene un NIDS de detectar una nueva conexión condicionará en gran medida su precisión. En el momento de inicializar BCT se almacenan datos como los números de secuencia iniciales que el NIDS utilizará para mantenerse "sincronizado" y poder reensamblar la sesión. Si un intruso consigue forzar la creación de un falso BCT con falsos números de secuencia, conseguirá que las siguientes conexiones con los mismos parámetros (direcciones y puertos) estén desincronizadas con el NIDS, ya que detectará que el SEQ de los paquetes es erróneo.

Existen diversas aproximaciones para detectar una nueva conexión, pero prácticamen-

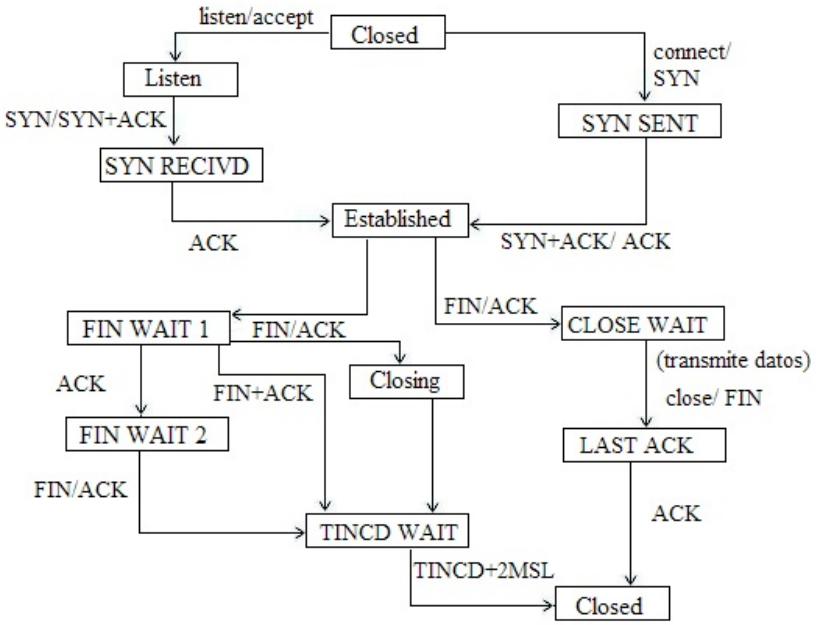

Figura 2.23: Autómata TCP

te todas presentan inconvenientes que un atacante puede vulnerar. La primera opción y más obvia consiste en crear el BCT cuando el NIDS detecte el saludo a tres vías TCP. El inconveniente claro que sin saludo, analizará cada paquete individualmente y no como un flujo TCP. Cosa conseguirá, si el atacante se asegura que el NIDS no vea el saludo utilizando alguna de las técnicas de evasión antes comentadas. Otro problema es la creación de falsos BCT forzados por el atacante, de desincronizará cualquier otra conexión con los mismos parámetros (ejemplo del párrafo anterior). A priori en una red donde se controle el *spoofing* de IP será una tarea realmente complicada falsear las respuestas del servidor para engañar al NIDS.

Existen otras posibilidades que consisten, por ejemplo, en crear el BCT cuando detecten paquetes SYN u obtener el estado de la conexión y los números de secuencia analizando el tráfico existente.

E. Reensamblado

Como ya se ha comentado en varias ocasiones, para conseguir que el eensamblado del flujo TCP sea correcto es necesario un seguimiento fiable de los números de secuencia. Veamos algunos puntos débiles donde romper esa sincronización y evadir el análisis de patrones.

Un problema inherente de los monitores pasivos, es que si pierden paquetes no pueden solicitar una retransmisión como cualquiera de las máquinas implicadas en la comunicación. Por tanto un intruso puede provocar de algún modo intencionado la pérdida de paquetes, por ejemplo gracias a ataques de DoS o paquetes que llegan fueran de orden y el NIDS no procesa adecuadamente. Las pérdidas de paquetes provocan una desincronización del NIDS y por consiguiente evita que el eensamblado del flujo TCP sea posible.

Irix 5.3	Procesa los datos nuevos
HP-UX 9.01	Procesa los datos nuevos
Linux	Procesa los datos nuevos
AIX 3.25	Procesa los datos nuevos
Solaris 2.6	Procesa los datos nuevos
FreeBSD 2.2	Procesa los datos nuevos
Windows NT	Procesa los datos antiguos

Tabla 2.2: Superposición de segmentos TCP

Otra clase de problemas hacen referencia al tamaño de la ventana de la comunicación TCP, un NIDS normalmente no analizará si los datos segmentos que procesa se encuentran dentro de la ventana indicada por la máquina destino. Un NIDS que no realiza esta comprobación será vulnerable a ataques de inserción ya que estos paquetes que superan el tamaño de la ventana serán descartados en la máquina destino.

Al igual que ocurría con el reensamblado de fragmentos IP, en TCP también es posible que ocurra una superposición de segmentos. Un intruso intencionadamente puede generar dos segmentos TCP exactamente iguales salvo en el campo de datos (y por supuesto en el *checksum*).A la máquina objetivo le llegarán los dos, pero solo procesará uno dependiendo de su implementación (coger el más antiguo o el más nuevo), el NIDS sin conocer el SO de la máquina atacada, solo con el tráfico de red no puede saber cuál de los dos segmentos ha procesado. Y si no los procesa exactamente igual, es susceptible de ataques de inserción o evasión.

La segunda posibilidad es el envío de segmentos TCP desordenados con un tamaño tal que al reensamblarlos se sobrescriban ciertos segmentos. En la siguiente tabla podemos ver como manejan los diferentes sistemas operativos la superposición de segmentos TCP:

Una posibilidad que tiene el IDS para saber si un paquete es procesado en la maquina destino, es esperar hasta que escuche el ACK de la máquina destino, que indicará que el paquete se ha procesado correctamente. Pero no será posible en todos los casos ya que el número ACK es acumulativo y no se realiza el asentamiento de todos y cada uno de los segmentos.

F. Destrucción del BCT

Evidentemente es imposible mantener eternamente los BTC en memoria, por lo que el NIDS tendrá algún mecanismo para desecharlos. Mecanismo que intentará vulnerar un atacante.

Un intruso puede hacer creer al NIDS que la conexión con la máquina objetivo ha terminado, entonces el NIDS al cabo de un tiempo o inmediatamente desechará el BTC de memoria, y el intruso podrá seguir atacando sin miedo a que el NIDS reensamble los paquetes. Por lo tanto el NIDS debe conocer exactamente cuando se cierra una conexión y evitar ser engañado.

En TCP una conexión se finaliza con dos formas diferentes: utilizando paquetes FIN o paquetes RST. Cuando un sistema envía un segmento FIN indica que ha terminado de enviar datos y está preparado para cerrar la conexión. Los mensajes FIN se responden con un ACK y cada extremo de la conexión envía deberá enviar un segmento FIN para terminar la comunicación. A priori, en un entorno sin posibilidad de *spoofing* de IP, es suficientemente fiable que el NIDS confíe en la desconexión FIN para determinar que realmente una conexión ha terminado.

No ocurre lo mismo con los paquetes RST. Éste flag indica que ha habido algún error y la conexiones debe cerrarse inmediatamente. Como el segmento RST no tiene acuse de recibo, el NIDS no puede saber si el servidor que vigila le ha llegado el paquete y ha cerrado la conexión realmente. Un atacante puede generar un segmento RST válido (SEQ ok) pero que no será procesado por la máquina objetivo. De esta forma el NIDS cree que ha finalizado la conexión.

El único método que tiene el IDS para verificar que se ha cerrado la conexión es escuchando durante un periodo de tiempo si se transmiten datos, si no es así se desecha el BTC. Pero esto implica potencialmente un nuevo tipo de ataque contra el NIDS. Imaginemos la siguiente situación: El atacante inicia una conexión normalmente, inmediatamente después genera un segmento RST que no llega a la máquina objetivo pero que si procesa el NIDS. El atacante inicia otra conexión con los mismos parámetros pero con números de secuencia diferentes.

¿Cómo va a actuar el NIDS? ¿Generando un nuevo BTC y eliminando el antiguo? o ¿desecha la nueva conexión porque los segmentos tienen un número de secuencia diferente al que espera? En un caso como este cada IDS actuará de un modo, cosa que el intruso puede intentar explotar.

2.2.6. Ataques en otras capas

A. Nivel de enlace

Existen las mismas implicaciones comentadas durante todo el documento para los ataques de inserción en la campa de enlace. Un atacante en la misma LAN que el monitor de red puede directamente enviar paquetes de la capa de enlace del IDS, sin permitir que el host especificado como IP destino vea el paquete (Ya que el direccionamiento se hace a nivel MAC). Si el atacante conoce la MAC del NIDS, el atacante utilizando la técnica del ARP-*spoofing*, simplemente debe falsificar el paquete con la dirección del NIDS en la MAC destino, y la dirección IP de la máquina atacada. Ningún otro sistema de la LAN procesará el paquete aunque. Y si el NIDS no comprueba la correspondencia MAC-IP es correcta, el monitor de red es vulnerable a ataques de inserción.

De nuevo, a menos que el IDS compruebe la dirección de destino en la cabecera contrastándola con la cabecera de la capa de enlace (lo cual es normal que haga),podrá ser vulnerable a este tipo de ataques.

B. Nivel de aplicación

Todos los ataques que sea han comentado hasta ahora aprovechan inconsistencias del entorno y manipulan el tráfico de red, por supuesto no son las únicas técnicas existentes

a la hora de evadir la detección de patrones de un IDS.

Técnicas sobre el protocolo HTTP

Los NIDS pueden ser evitados con el simple uso de técnicas que convierten las URL's a hexadecimal [47]. También es posible explotar las vulnerabilidades CGI (common Gateway Interface) mediante el intercambio de los métodos utilizados (GET o HEAD). El atacante puede transformar el contenido de formato Unicode a hexadecimal, generando una gran cantidad de firmas diferentes. Como último ejemplo, el NIDS puede ser evitado con la colocación de los parámetros de instrucciones en diferente lugar al que corresponde en la petición HTML. Vamos a ver algunos ejemplos de las técnicas sobre el protocolo HTTP (RFC2616) para evitar que un IDS pueda detectar ataques *web*.

Todos esto ataques son bien conocidos y están basados en diferentes maneras de ofuscar contenido *web* malintencionado, explotando vulnerabilidades que no involucren un cifrado, por lo que prácticamente todos los sistemas de detección deberían manejarlos correctamente. En muchas ocasiones, el intruso intentará combinar varias técnicas o modificar las ya existentes.

Inyección SQL

Una técnica que está siendo ampliamente usada es "Structured Query Language (SQL) Injection". Es un método que permite explotar las aplicaciones *web* mediante la inserciones de comandos y meta-caracteres SQL entre los campos de entrada de una *web* con el fin de manipular la ejecución de las consultas SQL [48]. Es tan vulnerable que puede pasar por alto muchas de las capas de seguridad tradicionales como por ejemplo los *Firewall*, los cifrados, y también saltarse la autenticación de usuarios, lo cual es un gran fallo en las aplicaciones *web* [49].

La forma principal de inyección de código SQL consiste en la inserción directa de código en variables especificadas por el usuario que se concatenan y se ejecutan con comandos SQL. Existe un ataque menos directo que inyecta código dañino en cadenas que están destinadas a almacenarse en una tabla. Cuando las cadenas almacenadas se concatenan posteriormente en un comando SQL dinámico, se ejecuta el código dañino.

Se dice que existe o se produjo una inyección SQL cuando, de alguna manera, se inserta o "inyecta" código SQL invasor dentro del código SQL programado, a fin de alterar el funcionamiento normal del programa y lograr así que se ejecute la porción de código "invasor" incrustado, en la base de datos.

El proceso de inyección consiste en finalizar prematuramente una cadena de texto y anexar un nuevo comando. Como el comando insertado puede contener cadenas adicionales que se hayan anexado al mismo antes de su ejecución, el atacante pone fin a la cadena inyectada con una marca de comentario "–".

Al ejecutar una consulta en la base de datos, el código SQL inyectado también se ejecutará y podría hacer un sinnúmero de cosas, como insertar registros, modificar o eliminar datos, autorizar accesos e, incluso, ejecutar otro tipo de código malicioso en el computador.

Cambio de método	GET /cgi-bin/some.cgi - HEAD /cgi-bin/some.cgi	En muchos casos para explotar un CGI puede ser posible cambiar el método utilizado y evadir la detección del ataque.
Codificación de la URL	cgbin - %63 %67 %69 %2d %62 %69 %6e	Técnica clásica que todos los IDS tienen en cuenta. El http permite la codificación de los caracteres de la URL en formato hexadecimal.
Transformación UNICODE	G0x5C 0xC19C 0xE0819C - U+005C En el ejemplo todos esos valores en hexadecimal al ser traducidos a Unicode representan el mismo valor, la barra . ¿Cuántas firmas tendría que tener el IDS?	-
Directorio 1	GET ////cgi-bin///vuln.cgi	-
Directorio 2	GET /cgi-bin/blahblah/../some.cgi HTTP/1.0	-
Directorio 3	GET /cgi-bin/./././vuln.cgi	-
Múltiples peticiones	GET / HTTP/1.0 Header: /../../cgi-bin/some.cgi HTTP/1.0	Un IDS es posible que solo analice la primera línea de la petición, hasta el HTTP/1.x
Evitar la detección de parámetros	GET /index.htm %3fparam=/../cgi-bin/some.cgi HTTP/1.0 GET /index.htm?param=/../cgi-bin/some.cgi HTTP/1.0	Muchos IDS buscan patrones en los parámetros de las aplicaciones de ataque. Se intenta evitar que el IDS situe los parámetros dentro de la petición http.
Directorios largos	GET /rfprfp[caracteres]rfprfp/../cgi-bin/some.cgi HTTP/1.0	-
Caracteres NULL	GET %00 /cgi-bin/some.cgi HTTP/1.0	-
Sensible a mayúsculas	/CGI-BIN/SOME.CGI	Al atacar máquinas Windows es posible utilizar mayúsculas para evadir detecciones de patrones básicas.

Tabla 2.3: Técnicas para evitar que un IDS detecte ataques *web*

2.3. Arquitectura x86 y acceso a memoria privilegiada

2.3.1. Introducción

En los inicios de la informática eran muy pocas las personas que tenían los conocimientos necesarios para usar los ordenadores. Sobre los años 50 y 60, los ordenadores eran máquinas enormes que ocupaban habitaciones enteras y que necesitaban de varias personas operando sobre ellas para su correcto funcionamiento. Sólo se podían permitir el lujo de tener ordenadores o universidades muy importantes o el ejército, en los hogares era impensable. Es evidente que a día de hoy, todo esto ha cambiado y hoy no concebimos un hogar en el que no haya un ordenador.

Para nuestro proyecto se ha elegido la arquitectura x86 como base para nuestra implementación y pruebas, por ello haremos una breve introducción para saber un poco más sobre esta arquitectura.

2.3.2. Arquitectura x86

X86 es la denominación genérica que se ha dado a una familia de microprocesadores de la marca Intel debido a la terminación de sus respectivos nombres 8086, 80286, 80386, 80486, etc. Desde los inicios de esta familia han sido prácticamente un estándar para los ordenadores. Esta arquitectura comienza sobre 1978, año en el que Intel presenta el 8086/88. Anteriormente habían publicado otros como 4004, 8080 y 8085, pero pertenecían a otra familia (IA-16, Intel Architecture 16 *bits*).

Durante toda la era de la x86 se ha mantenido la compatibilidad entre arquitecturas con procesadores precedentes. Cosa que no ocurre a partir de los últimos modelos de Intel como son el Itanium e Itanium 2 con arquitectura IA-64, que son totalmente incompatibles con procesadores anteriores. El 80386 (386 a partir de ahora) fue el primer procesador con un juego de instrucciones de 32 *bits*. Tanto operandos como direccionamiento en memoria los utilizan, lo que permite un direccionamiento de 4 GB. Este modelo también fue el primero en incorporar un MMU (Unidad de Gestión de Memoria) para realizar las traducciones de memoria.

Conocer esta arquitectura x86 es conocer la base de los procesadores. Es incontable la cantidad de procesadores de esta arquitectura que se han vendido a lo largo de la corta vida de la informática. Además ha sido una arquitectura que dio pie del concepto de ordenador personal.

Por otro lado, el gran rival de x86 es ARM (ver apéndice), una arquitectura cuyo crecimiento en los últimos años está siendo importantísimo al conseguir introducirse en dispositivos pequeños, eficientes y con un precio generalmente más atractivo, que seducen a muchos usuarios.

A. Características básicas del x86

- La arquitectura x86 es de longitud de instrucción variable, de tipo registro memoria y diseño CISC.

- El espacio de direcciones lineal es de 4GB, aunque la memoria física puede llegar hasta los 64GB en algunos modelos, con acceso desalineado y almacenamiento little-endian.
- Un programa normal dispone de 8 registros de propósito general de 32 *bits*, 6 registros de segmento de 16 *bits*, un registro de estado EFLAGS y un puntero de instrucción EIP, ambos de 32 *bits*. Dichos registros se pueden acceder desde las operaciones de propósito general, compuestas por las instrucciones de aritmética entera, las instrucciones de control de flujo, las de operaciones con *bits* y con cadenas de *bytes*, y las instrucciones de acceso a memoria.
- Un conjunto de 8 registros de coma flotante de 80 *bits*, un conjunto de 8 registros MMX y XMM, de 64 y 128 *bits* respectivamente, para realizar operaciones SIMD.
- Un conjunto de recursos para el manejo de la pila y la invocación de subrutinas.
- El SO dispone además de puertos E/S, registros de control, de manejo de memoria, de depuración, de monitorización, etc.

I. Registros de propósito general

Los registros de propósito general se ilustran en la figura 2.24. Para la confección de la herramienta TYNIDS se ha apoyado en el uso del registro EAX para implementar la técnica de polimorfismo que más adelante se explicará.

Figura 2.24: Registros de propósito general

Además estos registros tienen un uso especial en algunas instrucciones.

- EAX Registro acumulador. Fuente y destino en algunas operaciones.
- EBX Puntero a los datos en el segmento DS.
- ECX Contador en las operaciones de cadena y bucles.
- EDX Puntero a los puertos de E/S.
- ESI Puntero fuente en las operaciones de cadena y puntero a datos en el segmento DS.

- EDI Puntero destino en las operaciones de cadena y puntero a datos en el segmento ES.
- ESP Puntero de pila.
- EBP Puntero a los datos alojados en la pila.

II. Niveles de privilegio

En la arquitectura x86 existen 4 niveles de privilegios, siendo el nivel 0 el más privilegiado, teniendo acceso a todos los recursos del sistema y el nivel 3, el menos privilegiado. En un sistema operativo tradicional, el nivel 0 es usado por el kernell y el de menor nivel es usado por el usuario.

El hecho de usar distintos niveles de privilegio, genera latencias, ya que el SO debe de estar todo el tiempo comprobando que cada acceso de lectura, escritura o ejecución es válido. En un SO de propósito general, esta jerarquía de privilegios es totalmente necesaria por ejemplo para que se puedan ejecutar varios procesos y, también para que en caso de que algún proceso esté mal programado, no sobrescriba código que pertenece al propio kernell. Existen instrucciones que otorgan el acceso a algunos puertos de E/S. Desde luego que estas instrucciones no se pueden ejecutar en el nivel de privilegio 3 sino que las ejecuta el S.O. en el nivel de privilegio 0 y su efecto se mantiene al volver al proceso. Los niveles de privilegio 2 y 3 son intermedios y se usan para servicios del sistema operativo, drivers o dispositivos. Una representación gráfica de los niveles de privilegio podría ser como la siguiente. (Figura 2.25)

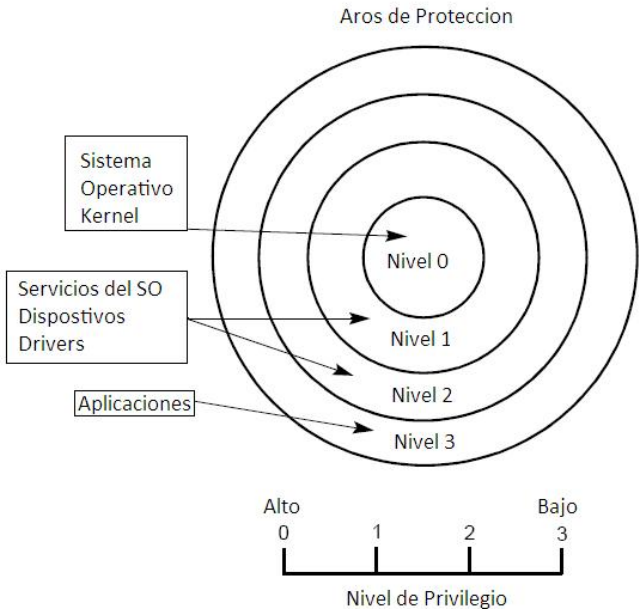

Figura 2.25: Niveles de privilegios

2.3.3. Técnicas de acceso a zonas de memoria privilegiadas

A. Buffer overflow y stack overflow

I. *Buffer overflow*

El *buffer overflow* [50] es un error de *software* que se produce cuando un programa no controla bien el tamaño de los datos que tiene que procesar, de tal forma que el tamaño de esos datos que tiene que procesar, es superior al tamaño máximo que le ha sido asignado al proceso, por lo que al escribir esos datos, en realidad no se están escribiendo en zonas de la memoria que pertenecen a ese proceso, si no que pertenecen a otro, pudiendo sobre escribir incluso zonas de la memoria que pertenecen al SO.

En una arquitectura común, como la zona de memoria entre datos e instrucciones no es diferenciable, si se sobre escribe una zona de memoria que no se debe, se pueden reemplazar zonas en las que antes había instrucciones, y esto provocaría que el flujo del programa se vería afectado.

Por lo tanto, esta vulnerabilidad la podría utilizar un usuario malintencionado para cambiar el flujo del programa y redirigirlo al código malintencionado.

En los procesadores modernos, existen directivas del SO para poder proteger zonas de memoria, y que cuando un proceso intente escribir sobre estas, se lance una excepción. Esta es una técnica para evitar esta clase de ataques, aunque estas zonas de memoria debe de declararlas restringidas el programador.

Un ejemplo sencillo de *buffer* overflow sería el siguiente. Disponemos de 2 *buffers* (figura 2.26 y figura 2.27) con los siguientes valores:

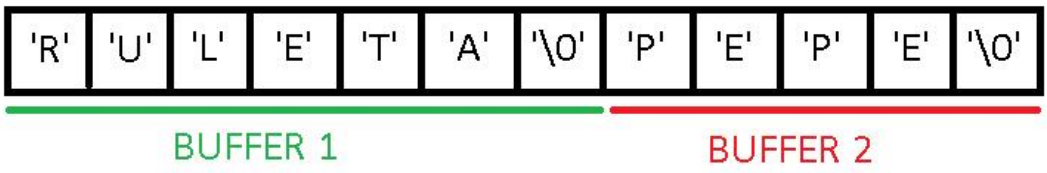

Figura 2.26: Estado inicial del *buffer*

El valor de buffer1 es "RULETA" y el valor del *buffer* es "PEPE". Nuestro programa inicialmente tiene una variable con el valor "HOLA MUNDO". A continuación copiamos el valor de esta variable en el buffer1 y como existen funciones por ejemplo en 2 que no hacen comprobación de los tamaños, si utilizasemos una de estas funciones el resultado sería:

Como "hola mundo" ocupa más que el buffer1, lo que ocurre es que se sobre escriben las direcciones de memoria siguientes al del buffer1, en este caso las de buffer2, y en un principio, si no es código mal intencionado, esto no es lo que se busca. Para evitar lo anterior existen funciones que copian valores en buffers con comprobación de tamaños.

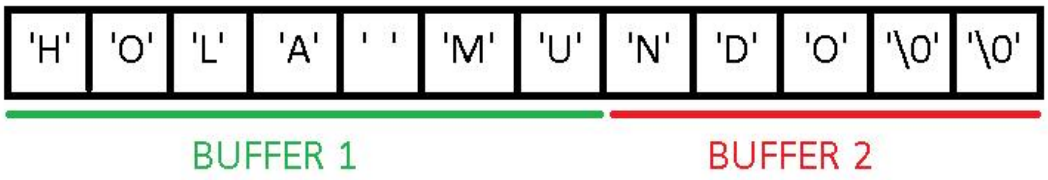

Figura 2.27: Buffer sobre escrito

II. Stack overflow

La técnica de stack overflow se basa en el mismo principio que el *buffer* overflow, pero en lugar de buscar sobrescribir zonas de memoria en las que haya instrucciones para poder cambiar el flujo de programa, lo que busca es sobrescribir la zonas de memoria que pertenezcan a la pila.

Ya que en la pila se guarda el contexto del estado del procesador cuando hay una llamada a una función, una interrupción, etc. Lo que busca esta técnica es poder sobrescribir la zona de la pila en la que se encuentra el puntero de retorno, por lo que cuando el procedimiento o función termina, en lugar de volver a la dirección de la instrucción donde se encontraba, como se ha sobrescrito el valor, presumiblemente ira a una zona de memoria donde el usuario mal intencionado ha colocado su código.

B. NOP *Sleed*

Esta técnica también es conocida como NOP *slide* y NOP *ramp*. Consiste en una secuencia de NOPs (instrucción que no hace nada). Esta técnica se utiliza cuando la ejecución de un programa se ramifica, por lo que no se sabe en qué posiciones de memoria so colocará el código malicioso, por lo que el usuario mal intencionado sobrescribe zonas de memorias anteriores y posteriores a su código malicioso con secuencias de NOPs, y al final de ellos coloca un salto que será relativo, y este salto conduce a su código. Entonces cuando una de estas ramificaciones, ejecuta un salto y cae en una zona de NOPs, se ejecutara toda la secuencia y la final se ejecutará nuestra instrucción de salto efectiva para ejecutar el código malintencionado.

La descripción de dicha técnica hace muy fácilmente detectar que se está usando. Basta con buscar zonas de un programa que haya secuencias de NOPs. Para ocultar esta técnica lo que se hace es en lugar de rellenar con NOPs, se rellena con instrucciones que no afecten a la ejecución del programa, como por ejemplo sumar cero a un registro, o mover un registro a sí mismo).

C. Heap Spray

Esta técnica es una alternativa a las técnicas de acceso a zonas de memoria con acceso privilegiado que atacan a la pila. Para comprender su funcionamiento, hay que tener claro las diferencias entre pila de programa y montículo (heap). Ambas partes de los procesos se crean al principio de la ejecución.

Figura 2.28: Ejemplo de NOP *Sleed*

PILA	MONTÍCULO
Tamaño constante	Tamaño variable
Gestión de la CPU	Gestión del compilador
Almacena parámetros de funciones, variables globales y constantes (todas ellas de tamaño fijo)	Almacena variables locales (tamaño variable)

Tabla 2.4: Diferencias entre PILA y MONTÍCULO en la técnica de *Heap Spray*

Las principales diferencias son:

D. *Heap Sleed* dirigido

En este tipo de ataque, el atacante conoce la dirección a la que va a saltar el programa atacado en alguna función, o conoce alguna variable que puede ser explotada.

Entonces crea un *Sleed* entre esa dirección vulnerable y el código que quiere ejecutar.

Cuando el programa vulnerable accede al *Sleed*, éste le conduce al código malicioso, y se apodera del flujo de ejecución.

E. *Heap Sleed* no dirigido

El atacante coloca arbitrariamente un *Sleed* muy grande cerca de las regiones de memoria en las que opera el programa vulnerable. Se espera a que el programa caiga dentro del *Sleed*. Una vez dentro, se hace con el flujo de ejecución del programa.

La ventajas de la técnica de Heap spray con respecto a otras técnicas de overflow de la pila es que al estar gestionada la pila por el SO, su ubicación en memoria sea prácticamente impredecible para el atacante, por lo que el éxito de un ataque con la técnica de stack overflow tiene mucho porcentaje de azar.

Por el contrario, la dirección de inicio del montículo es predecible por el atacante, por lo que el atacante puede actuar con mayor precisión.

Capítulo 3

Proyecto TYNIDS: Test Your NIDS

3.1. Introducción

El proyecto de Sistema de Ofuscación de *Malware* para la Evasión de NIDS, de ahora en adelante SOMEN, tiene como objetivo ofuscar *malware* para que pase inadvertido ante un NIDS. SOMEN combina algoritmos de cifrado de clave simétrica y técnicas de polimorfismo para conseguir la ofuscación. El resultado es un fichero ejecutable que contiene el *malware* ofuscado y lo hace invisible ante un NIDS, incluso *malware* previamente conocido.

TYNIDS es el nombre con el que se ha bautizado a la herramienta generada a través del proyecto SOMEN. En este apartado se presentará información sobre los métodos de ofuscación empleados, además se comentará algunos aspectos sobre los algoritmos utilizados en la ofuscación y las técnicas de polimorfismo empleadas. A continuación se procederá a mostrar los resultados obtenidos en el envío de malware ofuscado ante APAP, un preprocesador de Snort desarrollado en la Universidad Complutense de Madrid.

Adicionalmente se introducen los trabajos realizados para mejorar tanto el proceso de ofuscación, como el de des-ofuscación que realiza el fichero resultado durante su ejecución, mediante el uso de la librería *OpenMP* y arquitecturas de memoria compartida (en nuestro caso las pruebas se hicieron sobre un procesador Intel Core i3). Así mismo se recopila los resultados obtenidos para las distintas opciones de paralelización que fueron probadas durante el desarrollo del proyecto.

Por último se hará un análisis de posicionamiento de la herramienta en el mercado actual de Sistemas Detectores de Intrusos teniendo en cuenta los resultados obtenidos en el anterior apartado.

3.2. Técnicas de Ofuscación usadas en la implementación

Son los métodos que se utilizan en la creación de virus para asegurar que este no sea detectable fácilmente y que cumpla con su objetivo [36]. A continuación se presentan las dos técnicas de ofuscación que se han utilizado.

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 3.1: Valores de la operación lógica XOR

3.2.1. Cifrado con algoritmos simétricos

A. Cifrado XOR

Este algoritmo sencillo pero que cumple su función, utiliza la operación lógica XOR para cifrar bloques bit a bit utilizando una clave. A continuación se explica con un ejemplo el mecanismo de este algoritmo. El fundamento de la OR exclusiva se muestra en la siguiente tabla.

Dada la cadena "0101 1011 0100" y la clave "1101 1111 0010" el resultado de aplicar la OR exclusiva es el siguiente:

```
    0101 1011 0100
    1101 1111 0010
-------------------------
    1000 0100 0110
```

Si se vuelve a aplicar la XOR a la secuencia de bit resultante con la misma clave se obtiene el mensaje original.

```
    1000 0100 0110
    1101 1111 0010
-------------------------
    0101 1011 0100
```

B. AES

AES (*Advanced Encryption Standard* por sus siglas en Inglés), también conocido como Rijndael, es un algoritmo de cifrado por bloques creado por dos criptólogos belgas, Joan Daemen y Vicent Rijmen, que ha sido adoptado como un estándar de cifrado por el gobierno de los Estados Unidos.

El cifrado por bloques consiste en coger texto de una longitud fija de bits y transformarlo mediante una serie de operaciones de cifrado en otro texto de la misma longitud.

Descripción del algoritmo

Se caracteriza por ser un algoritmo de cifrado por bloques. Los datos a cifrar se dividen en bloques de tamaño fijo (128 bits), donde cada bloque se representa como una matriz de 4x4 bytes llamada estado, como se muestra en la siguiente figura.

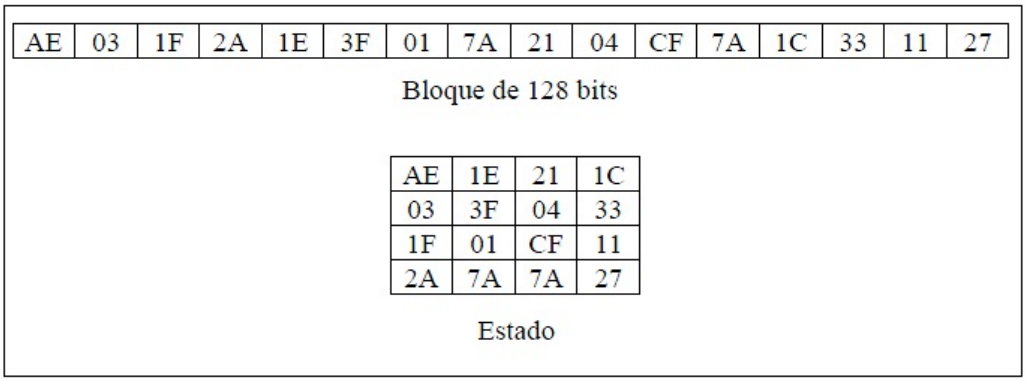

Figura 3.1: Estado de AES

A cada estado se le aplican N rondas, cada una está compuesta por un conjunto de operaciones. Las N rondas se pueden clasificar en tres tipos: una ronda inicial, Nr rondas estándar y una ronda final. Por ser AES un algoritmo simétrico, utiliza la misma clave para cifrar y descifrar los datos. A esta clave se la denomina clave inicial (K), la longitud de la clave K varia de 128, 192 y 256 bits, en cada caso AES tiene 10, 12, y 14 rondas respectivamente. Las claves resultantes junto con la clave inicial son denominadas sub-claves y cada una es utilizada en una de las rondas.

La ronda inicial realiza una sola operación:

- AddRoundKey: se hace un XOR byte a byte entre el estado y la clave inicial.

En cada una de las siguientes Nr rondas, denominadas estándar, se aplican 4 operaciones en el siguiente orden:

- SubBytes: se reemplaza cada byte del estado por otro de acuerdo a una tabla de sustitución de bytes con valores predeterminados. Este valor resultante se obtiene accediendo a la tabla tomando como índice de fila los primeros 4 bits del byte a reemplazar y como índice de columna los últimos 4 bits. El tamaño de la tabla es de 16x16 bytes.
- ShiftRows: a excepción de la primera fila del estado, que no se modifica, los bytes de las filas restantes se rotan cíclicamente a izquierda: una vez en la segunda fila, dos veces en la tercera y tres veces en la cuarta.
- MixColumns: a cada columna del estado se le aplica una transformación lineal y es reemplazada por el resultado de esta operación. AddRoundKey: es igual a la ronda inicial pero utilizando la siguiente sub-clave.

- AddRoundKey: es igual a la ronda inicial pero utilizando la siguiente sub-clave.

La ronda final consiste de 3 operaciones:

- SubBytes: de la misma forma que se aplica a las rondas estándar.
- ShiftRows: de la misma forma que se aplica a las rondas estándar.
- AddRoundKey: al igual que las rondas anteriores pero utilizando la última sub-clave.

La siguiente figura 3.2 muestra un esquema del funcionamiento del algoritmo explicado anteriormente.

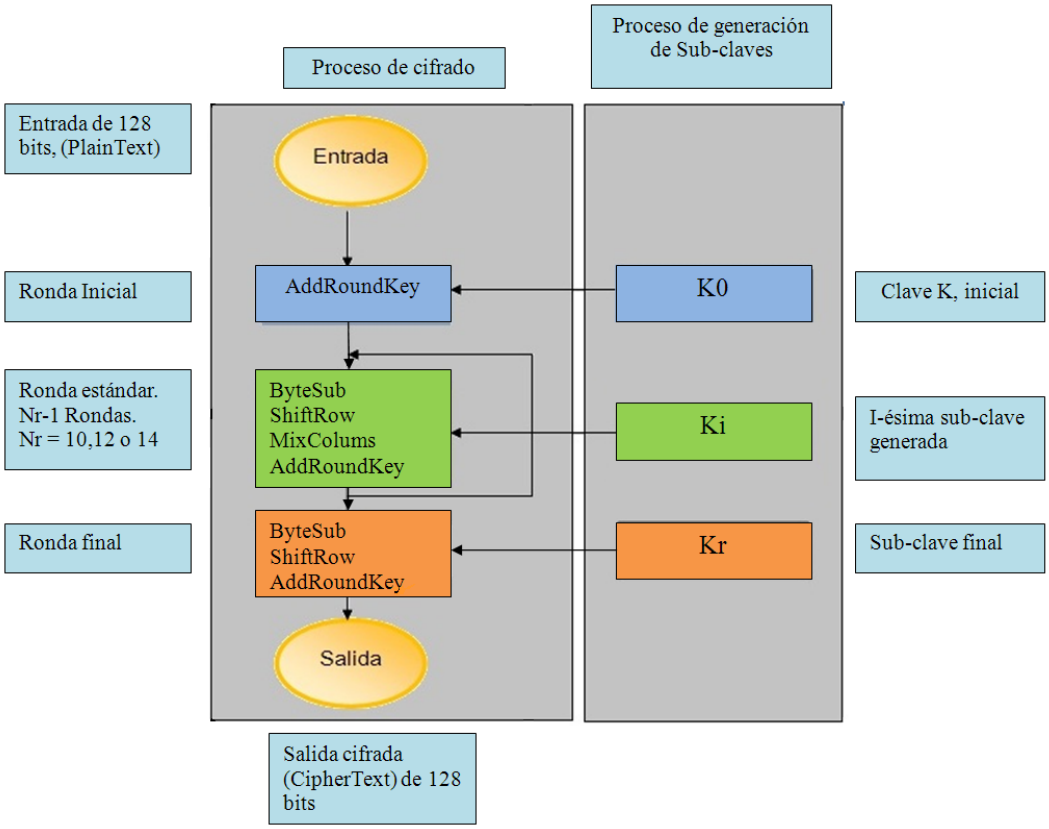

Figura 3.2: Esquema del algoritmo AES

C. Triple DES

Triple DES es un algoritmo que hace un triple cifrado DES, desarrollado por IMB en 1988. DES es un algoritmo de cifrado por bloques, que utiliza un tamaño de bloque de 64 bits. También utiliza una clave para realizar el cifrado y el descifrado. La clave mide 64 bits, aunque en realidad solo 56 de ellos son empleados por los algoritmos. Los 8 bits restantes se utilizan para comprobar la paridad, y después son descartados. El Instituto Nacional de Estándares Americano (ANSI por sus siglas en ingles) adoptó DES como

estándar para el cifrado comercial y datos sensibles. Esto está definido en Estándares Federales de Procesamiento de la Información (FIPS 46, 1977) publicado por el Instituto Nacional de Estándares y Tecnología (NIST) [51] [52]

Descripción

La estructura básica del algoritmo presenta 16 fases idénticas de proceso denominadas rondas. También hay una permutación inicial y final denominada PI y PF, que son funciones inversas entre sí (PI "deshace" la acción de PF, y viceversa). PI y PF no son criptográficamente significativas, pero se incluyeron presuntamente para facilitar la carga y descarga de bloques sobre el hardware de mediados de los 70. Antes de las rondas, el bloque es dividido en dos mitades de 32 bits y procesadas alternativamente. Este entrecruzamiento se conoce como esquema Feistel.

La estructura de Feistel asegura que el cifrado y el descifrado sean procesos muy similares, la única diferencia es que las sub-claves se aplican en orden inverso cuando desciframos. El resto del algoritmo es idéntico.

La función-F mezcla la mitad del bloque con parte de la clave. La salida de la función-F se combina entonces con la otra mitad del bloque, y los bloques son intercambiados antes de la siguiente ronda. Tras la última ronda, las mitades no se intercambian; ésta es una característica de la estructura de Feistel que hace que el cifrado y el descifrado sean procesos parecidos.

La siguiente figura 2.31 muestra un esquema de la estructura general de Feistel.

La función F de Feistel

La función-F, representada en la figura 2.32, opera sobre medio bloque (32) cada vez y consta de cuatro pasos:

1. Expansión: La mitad del bloque de 32 bits se expande a 48 bits mediante la permutación de expansión, denominada E en el diagrama, duplicando algunos de los bits.

2. Mezcla: El resultado se combina con una sub-clave utilizando una operación XOR. Dieciséis sub-claves, una para cada ronda, se derivan de la clave inicial mediante la generación de subclaves descrita más abajo.

3. Sustitución: Tras mezclarlo con la sub-clave, el bloque es dividido en ocho trozos de 6 bits antes de ser procesados por las S-cajas, o cajas de sustitución. Cada una de las ocho S-cajas reemplaza sus seis bits de entrada con cuatro bits de salida, de acuerdo con una trasformación no lineal, especificada por una tabla de búsqueda. Las S-cajas constituyen el núcleo de la seguridad de DES, sin ellas el cifrado sería lineal y fácil de romper.

4. Permutación: Finalmente, las 32 salidas de las S-cajas se reordenan de acuerdo a una permutación fija; la P-caja.

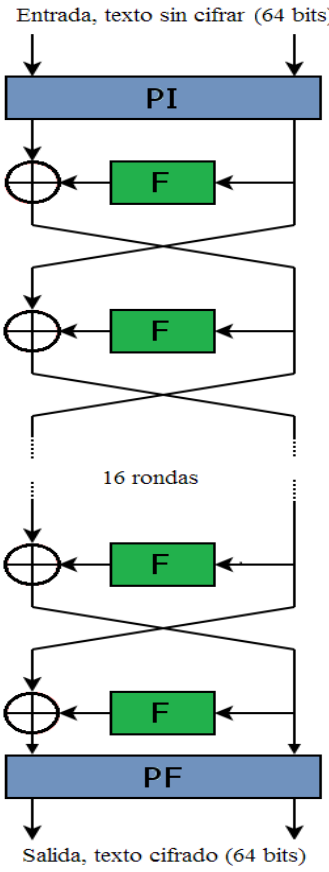

Figura 3.3: Estructura general de Feistel

Alternando la sustitución de las S-cajas, y la permutación de bits de la P-caja y la expansión-E proporcionan las llamadas "confusión y difusión" respectivamente, un concepto identificado por Claude Shannon en los 40 como una condición necesaria para un cifrado seguro y práctico.

Generación de claves

Primero, se seleccionan 56 bits de la clave de los 64 iniciales mediante la Elección Permutada 1 (PC-1), los ocho bits restantes pueden descartarse o utilizarse como bits de comprobación de paridad. Los 56 bits se dividen entonces en dos mitades de 28 bits; a continuación cada mitad se trata independientemente. En rondas sucesivas, ambas mitades se desplazan hacia la izquierda uno o dos bits (dependiendo de cada ronda), y entonces se seleccionan 48 bits de sub-clave mediante la Elección Permutada 2 (PC-2), 24 bits de la mitad izquierda y 24 de la derecha. Los desplazamientos (indicados por "" en el diagrama) implican que se utiliza un conjunto diferente de bits en cada sub-clave; cada bit se usa aproximadamente en 14 de las 16 sub-claves. (Figura 2.33)

La generación de claves para descifrado es similar, se debe generar las claves en orden inverso.

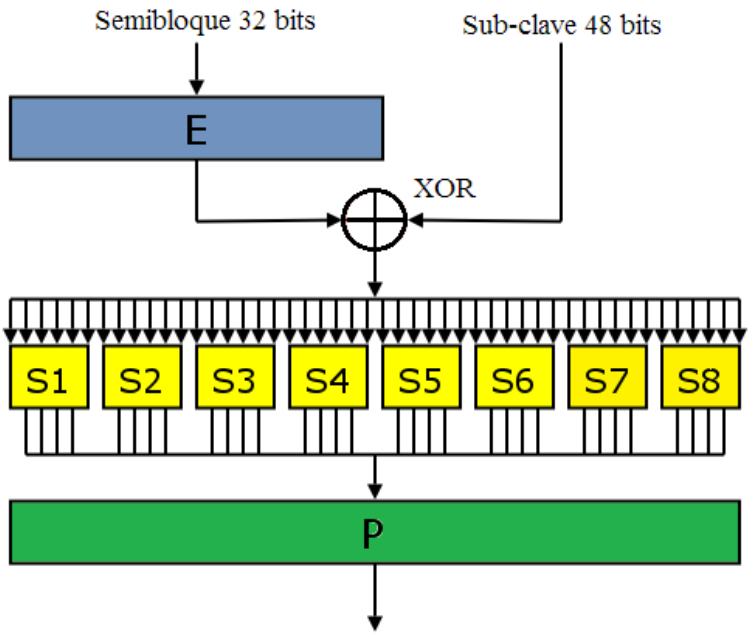

Figura 3.4: Función F

3.2.2. Polimórfismo

A continuación se explica las técnicas de polimorfismo, sin entrar en mucho detalle, que se han incluido en el proyecto. La herramienta cuenta con tres técnicas de polimorfismo posible que podrá ser seleccionado en cada ejecución.

TYNIDS es capaz de generar para una misma entrada distintas salidas, es decir, para un mismo malware a cifrar, la herramienta es capaz de aplicar distintas técnicas de polimorfismo con variables aleatorias que hará que la salida siempre tenga una forma distinta, en cuanto a contenido de instrucciones se refiere. Esto provocará que el NIDS tenga mucha más carga de trabajo a la hora de intentar detectar el malware debido a que la detección por firmas no le será útil.

A. Inserción de código basura

Este tipo de procedimiento genera un número aleatorio de instrucciones que serán introducidas en distintas partes del código. Como no es posible introducir las instrucciones en cualquier parte, se realizan unas series de búsquedas de lugar previamente ya estudiadas y almacenadas.

Los tipos de instrucciones que se introducen son de un tipo especial inofensivo, es decir, son instrucciones que sirven de relleno, no afectan al contexto del código y sin embargo hace que el código crezca en la medida que sea deseada.

Con esto tratamos de conseguir que distintas ejecuciones del código genere distinto número de instrucciones resultantes. Por ejemplo, un código inicial de 1000 instrucciones, tras aplicar esta técnica a veces podrá generar códigos de 1050 instrucciones, 2000 instrucciones o 7000 instrucciones por poner algún ejemplo.

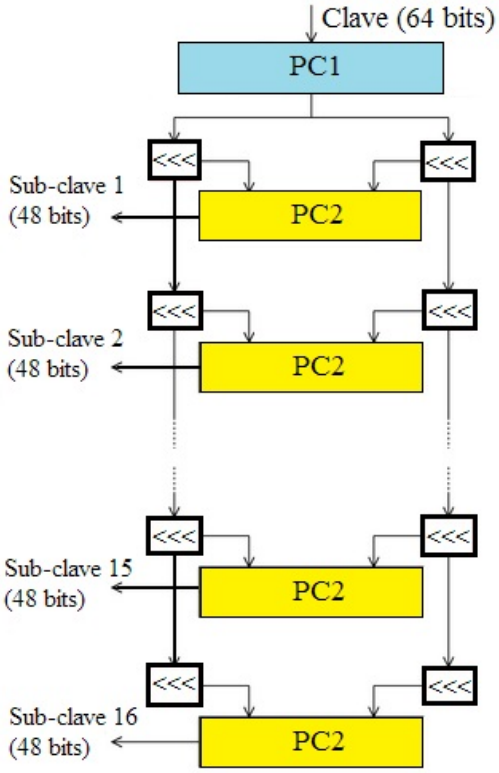

Figura 3.5: Generación de claves

Al ser un número diferente de instrucciones de relleno en cada ejecución generará que un análisis por firma no sea capaz de detectar el código malicioso que penetrará el NIDS.

B. Intercambio de instrucciones independientes

La segunda técnica que se ha utilizado es la de la búsqueda de instrucciones independientes. Aquí usamos un concepto de independencia propia. Se ha implementado un mecanismo de búsqueda de pares de instrucciones consecutivas en las que no se compartan los registros ni origen ni destino entre las instrucciones respectivamente.

Esto hace que un intercambio entre instrucciones consecutivas, es decir, poner la instrucción A en lugar de la instrucción B y la instrucción B justo donde estaba antes la instrucción A, haga que el código varíe de forma y sin embargo el contexto final, el resultado de la ejecución del código sea exactamente la misma que antes del intercambio.

De mismo modo que la técnica anterior, se aplica un número aleatorio para decidir cuantas veces se ejecutará este tipo de ejercicio, lo que hará que dos ejecuciones seguidas del mismo código generen código resultante diferente.

C. Inserción de código basura e intercambio de instrucciones independientes

Por último se usa un tercer tipo de técnica que es una mezcla de las dos anteriores. Primero se ejecutará la técnica de introducción de operaciones NOP o código basura y a continuación se empezará a hacer intercambios entre instrucciones independientes.

Es lógico que primero se introduzcan las instrucciones basura antes de hacer intercambios ya que cuantas más instrucciones tenga el código más posibilidades habrán de encontrar instrucciones candidatas para hacer intercambios.

Este tipo de técnica es la más prometedora para evadir al NIDS ya que incluye las otras dos técnicas explicadas anteriormente, pero por sí solas cualquiera de ellas podría servir para violar el sistema de detección del NIDS.

3.2.3. Funcionalidad de la herramienta TYNIDS

Para explicar cómo trabaja TYNIDS, lo más sencillo es enumerar y describir las distintas fases o estados por los que pasa desde que inicia su ejecución hasta que finaliza.

Primera fase: Cifrado

Es quizás la fase más importante, ya que aparte de ser la primera, es la que hace que el *malware* quede completamente ofuscado para intentar impedir que el NIDS sea capaz de detectarlo.

Lo primero que se hace para cifrar *malware* es obtener el *Opcode* del fichero ejecutable que queremos cifrar, a continuación, se le aplica uno de los algoritmos de clave simétrica comentados anteriormente (Apartado 3.2.1. Cifrado XOR, AES, 3DES) y una clave, ambos elegidos por el usuario de la aplicación.

El código del *malware* resultante de aplicar uno de estos tres algoritmos de cifrado, aún sufrirá más cambios, los cuales están explicados en los siguientes apartados.

A. Elaboración de ficheros intermedios

Dado que el código obtenido después del cifrado es totalmente incomprensible, se realiza un paso intermedio (figura 3.6) para dotar ese código ofuscado de una apariencia más amigable, y fácilmente manipulable.

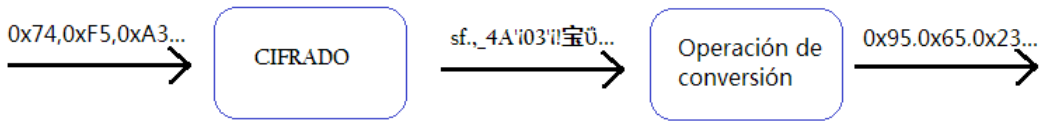

Figura 3.6: Esquema de elaboración de ficheros intermedios

Este paso de convertir el código incomprensible a algo más apetecible visualmente podría considerarse como la agregación de un nivel más al proceso de ofuscación.

Una vez que finaliza esta fase, se guarda en un fichero todo lo necesario para llevar a cabo el proceso inverso al de cifrado.

3.2.4. Segunda fase: Aplicación de técnicas de polimorfismo

Una vez obtenido el código ofuscado y con un formato fácilmente manipulable se genera un fichero que contiene las instrucciones en ensamblador sobre las que se aplican las técnicas polimórficas descritas anteriormente.

Este es el siguiente punto importante donde se aplica ofuscación al código del *malware*. Mediante la técnica del polimorfismo logramos que el resultado final nunca sea igual. Se aplican distintas transformaciones de forma aleatoria en cada ejecución, como por ejemplo inserción de código basura o intercambio de instrucciones independientes. Con esto conseguimos que el *malware* tenga más posibilidades de pasar inadvertido ante el NIDS.

3.2.5. Generación de ficheros ejecutables

Cuando se ha terminado de aplicar todas las técnicas polimórficas, TYNIDS genera un fichero ejecutable que contiene el *malware* cifrado, el mecanismo de descifrado y la autonomía necesaria para ejecutarse en la máquina destino, una vez burlada la seguridad del NIDS.

3.2.6. Ejecución de ficheros ejecutables

Dado que la finalidad de este proyecto es evadir un NIDS que vigila una red y atacar a una máquina que se esconde tras él, el fichero tiene que ser ejecutado en algún momento para llevar a cabo el ataque completo y así producirse la infección con el *malware*.

La ejecución tiene dos partes claramente diferenciadas, el descifrado y la ejecución del *malware* descifrado.

3.2.7. Descifrado

Para que el descifrado sea posible, primero se tienen que revertir todas las operaciones intermedias que se han realizado hasta obtener el código cifrado original (figura 2.35), excepto las aplicadas en la fase de aplicación de técnicas polimórficas.

Recordemos que el código del *malware* tiene una apariencia en hexadecimal y que podría interpretarse como código ejecutable, no obstante, este código no sigue el patrón necesario para que ese código pueda ser ejecutado. Dada esta falsa apariencia, se realiza el proceso inverso en dos etapas antes de proceder al descifrado.

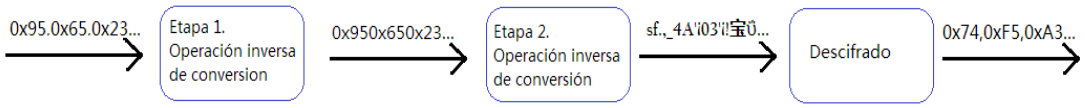

Figura 3.7: Proceso de descifrado del código

El código cifrado original se descifra aplicando la función de descifrado del algoritmo seleccionado al principio de la aplicación.

3.2.8. Generación de ficheros finales y ejecución

Una vez obtenido el código del *malware* descifrado, se procede a su ejecución. Para ello se crean dos ficheros finales, uno de ellos contendrá el código del malware descifrado y el otro las funciones encargadas de crear un fichero binario con el código del primer fichero y la llamada al sistema que ejecuta el fichero binario anteriormente mencionado. (Figura 3.8)

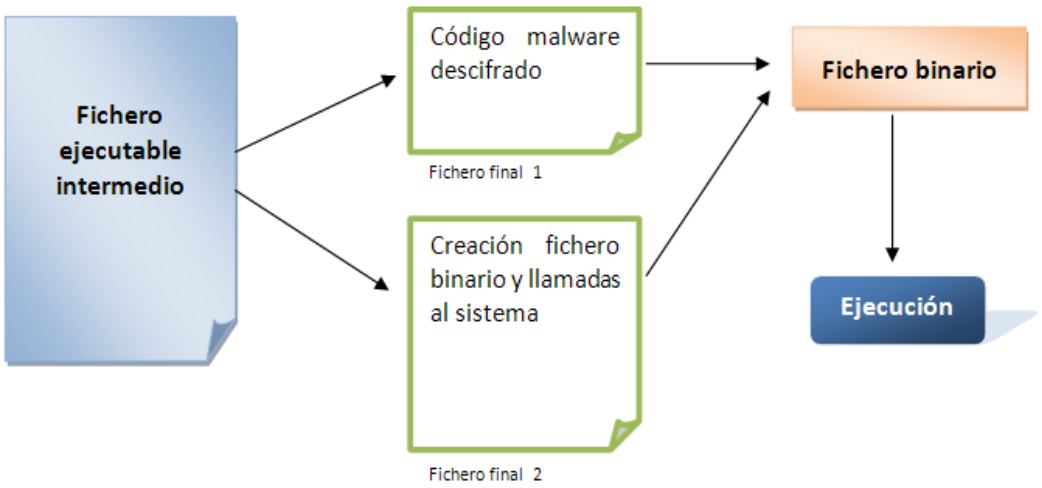

Figura 3.8: Esquema de generación de ficheros

3.2.9. Esquema de la estructura de los ficheros generados

A continuación se detalla la estructura (figura 2.37) de los ficheros intermedios generados por la aplicación, tanto en la fase de cifrado, como en la de descifrado.

Se comienza mostrando la estructura del fichero ejecutable generado en la fase de cifrado. Este fichero contiene el *malware* cifrado, en una falsa apariencia hexadecimal, a continuación el modulo de descifrado del algoritmo elegido para el cifrado, así como las operaciones inversas intermedias que se deben realizar antes del propio descifrado, y el cuerpo del programa principal.

La estructura de los fichero finales es muy simple, pero no por ello menos importante, ya que sin ellos la ejecución del *malware* no sería posible. (Figuras 2.38 y 2.39)

El fichero binario generado por los ficheros finales, es un fichero temporal que tiene una estructura binaria que no vamos a representar, dado que es una estructura específica de cada sistema operativo.

3.2.10. Guía de ejecución

Para hacer uso de la herramienta no es necesario tener unos profundos conocimientos en redes, NIDS o *malware*, sólo se necesitan los conocimientos básicos que permitan mo-

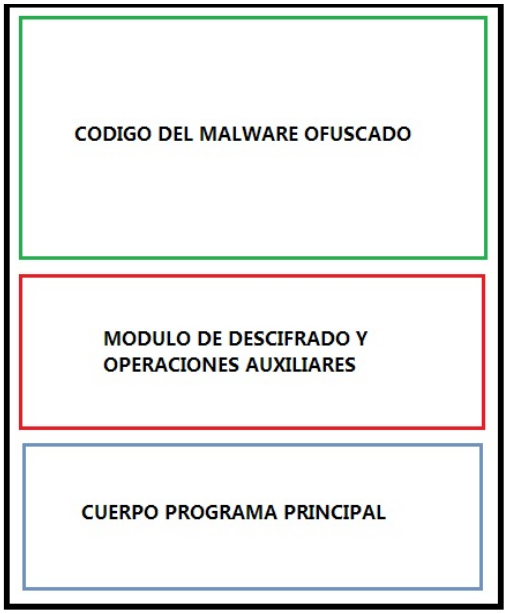

Figura 3.9: Estructura ficheros intermedios

verse por diferentes directorios mediante una *shell* de Linux.

El objetivo de la herramienta es ofuscar *malware* en ficheros ejecutables, por ello se hace requisito indispensable saber previamente donde está dicho *malware*. La herramienta generara todos los directorios que contendrán los ficheros intermedios que se creen durante el proceso de ofuscación, por lo que sabiendo el directorio fuente de los ficheros ejecutables, no hay nada más de lo que preocuparse. Para poder ejecutar la herramienta es necesario ingresar el comando de Linux en consola necesario para tal fin, seguido de directorio fuente donde están los ficheros a ofuscar. A continuación se muestra un ejemplo de cómo se debería ejecutar la herramienta. Nótese que el directorio tiene que acabar con la barra transversal "/".

```
./tynids.o directorio_fuente/
```

Una vez se ha ejecutado el comando, aparecerá un menú donde podremos elegir el algoritmo que deseamos para el proceso de ofuscación. A continuación se pide que se introduzca una clave, esta clave la necesitan los algoritmos para realizar el cifrado, después de esto mostrará un submenú donde se muestran los modos de polimorfismo disponibles y se pedirá que se ingrese el que se quiere aplicar. Si todo ha ido bien, por cada fichero que trate la herramienta, se mostrará un mensaje que indica el fichero de código fuente que se ha generado, si la compilación de dicho código ha sido correcto o no y los ficheros generados al aplicar polimorfismo. Si la compilación no ha sido correcta, se generará un fichero "log" que contendrá el error que devuelve el proceso de compilación.

Cuando la herramienta haya finalizado se generán varios directorios. El directorio "crypt_files" contiene el código fuente de los ficheros sin compilar, cuyo contenido es el malware cifrado con el algoritmo que se haya elegido anteriormente, y todo lo necesario

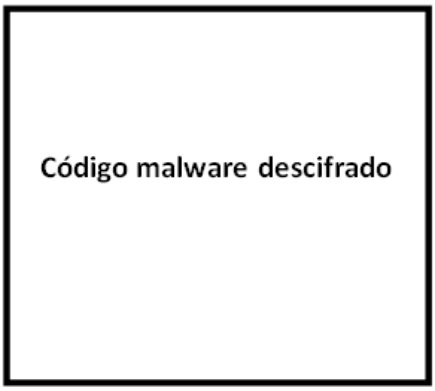

Figura 3.10: Estructura fichero finales I

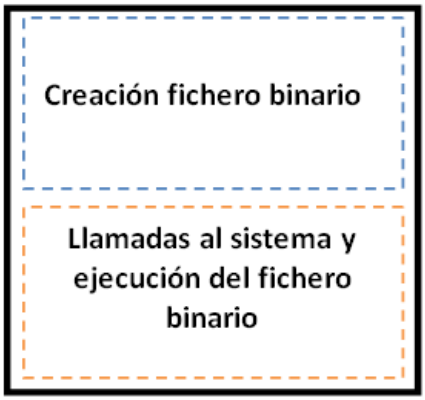

Figura 3.11: Estructura fichero finales II

para su posterior ejecución. El directorio "asm_files" contiene los ficheros compilados con el flag -S, es decir, los ficheros que contienen el código en ensamblador de los archivos que hay en el directorio "crypt_files". Estos ficheros con extensión ".s" son sobre los que se aplica la técnica de polimórfismo que se ha escogido previamente. Y por último el directorio "compilados" que contiene todos los ficheros finales, listos para ser usados.

Capítulo 4

Pruebas realizadas y resultados

En esta sección se describe qué tipo de pruebas se han generado para probar la eficacia de la herramienta TYNIDS objeto de este proyecto, así como también cómo se ha comportado y los resultados obtenidos con ella.

4.1. Introducción a las pruebas

La batería de pruebas está basada en una colección de 25 *malwares* y 47 antivirus escogidos de forma aleatoria y listados a continuación.

Listado de *Malwares*

- 13th
- 5120
- addict9
- cybernet
- darksatan
- diamon.gr
- elvis
- erasmus
- glitch
- guerilla
- halloween1376
- harakiri
- hero394
- ifreetv

Listado de Antivirus

- Agnitum
- AhnLab-V3
- AntiVir
- Antiy-AVL
- Avast
- AVG
- BitDefender
- ByteHero
- CAT-QuickHeal
- ClamAV
- Commtouch
- Comodo
- DrWeb
- Emsisoft
- eSafe
- ESET-NOD32
- F-Prot
- F-Secure
- Fortinet
- GData
- Ikarus
- Jiangmin
- K7AntiVirus
- K7GW
- Kaspersky
- Kingsoft
- Malwarebytes
- McAfee
- McAfee-GW-Edition
- Microsoft

- MicroWorld-eScan
- NANO-Antivirus

Las pruebas están divididas en 2 fases.

1. **Primera fase:** Se ha realizado un análisis de los 25 *malware* sin cifrar con cada antivirus de los mencionados anteriormente. Supone un total de 1175 análisis.
2. **Segunda fase:** Una vez cifrado el *malware* con cifrado AES, 3DES ó XOR se le ha aplicado técnicas de polimorfismo descritas en capítulos anteriores. Lográndose esta vez un total de 2350 análisis.

4.2. Resultados

Resultados primera fase:

Del total de 1175 análisis realizados en esta fase, 941 han sido positivos. Esto significa que en el 80 % de los análisis el *malware* ha sido descubierto, resultado que lógicamente es el esperado puesto que la herramienta TYNIDS aún no ha entrado en acción en esta primera fase. (Tablas 4.1, 4.2, 4.3, 4.4 y 4.5)

Resultados segunda fase:

Del total de 2350 análisis realizados en esta fase, 0 han dado positivo. Lo que significa que ningún antivirus ha sido capaz de descubrir el *malware* cifrado y ofuscado por TYNIDS. Es decir, se ha resuelto el objetivo con un 100 % de eficacia. (Tablas 4.6, 4.7, 4.8, 4.9, 4.10, 4.11, 4.12, 4.13, 4.14 y 4.15)

antivirus/malware	13th	5120	addict9	cybernet	darksatan
Avast	SI	SI	SI	SI	SI
AVG	SI	SI	SI	SI	SI
BitDefender	SI	SI	SI	SI	SI
ByteHero	NO	NO	NO	NO	NO
CAT-QuickHeal	SI	NO	SI	SI	SI
ClamAV	SI	SI	SI	SI	SI
Commtouch	SI	SI	SI	SI	SI
Comodo	SI	SI	SI	SI	SI
DrWeb	SI	SI	SI	SI	SI
Emsisoft	SI	SI	SI	SI	SI
eSafe	NO	NO	SI	SI	SI
ESET-NOD32	NO	NO	NO	SI	SI
F-Prot	SI	SI	S	SI	SI
F-Secure	SI	SI	SI	SI	SI
Fortinet	SI	SI	NO	SI	SI
GData	SI	SI	SI	SI	SI
Ikarus	SI	SI	SI	SI	SI
Jiangmin	SI	SI	SI	SI	SI
K7AntiVirus	SI	SI	SI	SI	SI
K7GW	SI	NO	SI	SI	NO
Kaspersky	SI	SI	SI	SI	SI
Kingsoft	NO	NO	NO	NO	NO
Malwarebytes	NO	NO	NO	NO	NO
McAfee	SI	SI	SI	SI	SI
McAfee-GW-Edition	SI	SI	SI	SI	SI
Microsoft	SI	SI	SI	SI	SI
MicroWorld-eScan	SI	SI	NO	SI	NO
NANO-Antivirus	SI	SI	SI	SI	SI
Norman	NO	NO	NO	SI	SI
nProtect	SI	SI	SI	SI	SI
Panda	SI	SI	SI	SI	SI
PCTools	SI	SI	SI	SI	SI
Rising	SI	SI	SI	SI	NO
Sophos	SI	SI	SI	SI	SI
SUPERAntiSpyware	NO	NO	NO	NO	NO
Symantec	SI	SI	SI	SI	SI
TheHacker	NO	SI	NO	SI	NO
TotalDefense	SI	SI	SI	SI	SI
TrendMicro	SI	SI	SI	SI	SI

Tabla 4.1: Resultado Fase 1: *Malware* no cifrado. Ejemplo I

antivirus/malware	diamon.gr	elvis	erasmus	glitch	guerilla
Avast	SI	SI	SI	SI	SI
AVG	NO	SI	SI	SI	SI
BitDefender	SI	SI	SI	SI	SI
ByteHero	NO	NO	NO	NO	NO
CAT-QuickHeal	SI	SI	SI	NO	SI
ClamAV	SI	SI	SI	SI	SI
Commtouch	SI	SI	SI	SI	SI
Comodo	SI	SI	SI	SI	SI
DrWeb	SI	SI	SI	SI	SI
Emsisoft	SI	SI	SI	SI	SI
eSafe	SI	SI	SI	SI	SI
ESET-NOD32	NO	NO	NO	SI	SI
F-Prot	SI	SI	SI	SI	SI
F-Secure	SI	SI	SI	SI	SI
Fortinet	SI	SI	SI	SI	SI
GData	SI	SI	SI	SI	SI
Ikarus	SI	SI	SI	SI	SI
Jiangmin	SI	SI	SI	SI	SI
K7AntiVirus	SI	SI	SI	SI	SI
K7GW	SI	NO	SI	SI	SI
Kaspersky	SI	SI	NO	SI	SI
Kingsoft	NO	NO	NO	NO	NO
Malwarebytes	NO	NO	NO	NO	NO
McAfee	SI	SI	SI	SI	SI
McAfee-GW-Edition	SI	SI	SI	SI	SI
Microsoft	SI	SI	SI	SI	SI
MicroWorld-eScan	NO	SI	SI	SI	SI
NANO-Antivirus	SI	SI	SI	SI	SI
Norman	NO	SI	NO	SI	SI
nProtect	SI	SI	SI	SI	SI
Panda	SI	SI	SI	SI	SI
PCTools	SI	SI	SI	SI	SI
Rising	SI	SI	SI	SI	SI
Sophos	SI	SI	SI	SI	SI
SUPERAntiSpyware	NO	NO	NO	NO	NO
Symantec	SI	SI	SI	SI	SI
TheHacker	SI	SI	SI	SI	SI
TotalDefense	SI	SI	SI	SI	SI
TrendMicro	SI	SI	SI	SI	SI

Tabla 4.2: Resultado Fase 1: *Malware* no cifrado. Ejemplo II

antivirus/malware	halloween1376	harakiri	hero394	ifreetv	int1988
Avast	SI	SI	SI	SI	SI
AVG	SI	SI	SI	SI	SI
BitDefender	SI	SI	SI	SI	SI
ByteHero	NO	NO	NO	NO	NO
CAT-QuickHeal	SI	NO	SI	NO	SI
ClamAV	SI	SI	SI	SI	SI
Commtouch	SI	SI	SI	SI	SI
Comodo	SI	SI	SI	SI	SI
DrWeb	SI	SI	SI	SI	SI
Emsisoft	SI	SI	SI	SI	SI
eSafe	SI	SI	SI	SI	SI
ESET-NOD32	SI	SI	NO	NO	SI
F-Prot	SI	SI	SI	SI	SI
F-Secure	SI	SI	SI	SI	SI
Fortinet	SI	SI	SI	SI	SI
GData	SI	SI	SI	SI	SI
Ikarus	SI	SI	SI	SI	SI
Jiangmin	SI	SI	SI	SI	SI
K7AntiVirus	SI	SI	SI	SI	SI
K7GW	SI	NO	SI	SI	SI
Kaspersky	SI	SI	NO	SI	SI
Kingsoft	NO	NO	NO	NO	NO
Malwarebytes	NO	NO	NO	NO	NO
McAfee	SI	SI	SI	SI	SI
McAfee-GW-Edition	SI	SI	SI	SI	SI
Microsoft	SI	SI	SI	SI	SI
MicroWorld-eScan	SI	SI	NO	NO	SI
NANO-Antivirus	SI	SI	SI	SI	SI
Norman	NO	SI	NO	SI	SI
nProtect	SI	SI	SI	SI	SI
Panda	SI	SI	SI	SI	SI
PCTools	SI	SI	SI	SI	SI
Rising	SI	SI	SI	SI	SI
Sophos	SI	SI	SI	SI	SI
SUPERAntiSpyware	NO	NO	NO	NO	NO
Symantec	SI	SI	SI	SI	SI
TheHacker	SI	SI	NO	NO	SI
TotalDefense	SI	SI	SI	SI	SI
TrendMicro	SI	SI	SI	SI	SI

Tabla 4.3: Resultado Fase 1: *Malware* no cifrado. Ejemplo III

antivirus/malware	intrud1	jerusalem	jsr3a	manequin	nextgen
Avast	SI	SI	SI	SI	SI
AVG	SI	SI	SI	SI	SI
BitDefender	SI	SI	SI	SI	SI
ByteHero	NO	NO	NO	NO	NO
CAT-QuickHeal	SI	SI	NO	SI	NO
ClamAV	SI	SI	SI	SI	SI
Commtouch	SI	SI	SI	SI	SI
Comodo	SI	SI	SI	SI	SI
DrWeb	SI	SI	SI	SI	SI
Emsisoft	SI	SI	SI	SI	SI
eSafe	SI	SI	SI	SI	SI
ESET-NOD32	SI	SI	NO	NO	SI
F-Prot	SI	SI	SI	SI	SI
F-Secure	SI	SI	SI	SI	SI
Fortinet	NO	NO	SI	SI	SI
GData	SI	SI	SI	SI	SI
Ikarus	SI	SI	SI	SI	SI
Jiangmin	SI	SI	SI	SI	SI
K7AntiVirus	SI	SI	SI	SI	SI
K7GW	SI	SI	SI	SI	NO
Kaspersky	SI	SI	SI	SI	SI
Kingsoft	SI	NO	NO	NO	SI
Malwarebytes	NO	NO	NO	NO	NO
McAfee	SI	SI	SI	SI	SI
McAfee-GW-Edition	SI	SI	SI	SI	SI
Microsoft	SI	SI	SI	SI	SI
MicroWorld-eScan	SI	NO	SI	SI	SI
NANO-Antivirus	SI	SI	SI	SI	SI
Norman	SI	SI	SI	SI	SI
nProtect	SI	SI	SI	SI	SI
Panda	NO	SI	NO	SI	SI
PCTools	SI	SI	SI	SI	SI
Rising	SI	NO	SI	SI	SI
Sophos	SI	SI	SI	SI	SI
SUPERAntiSpyware	SI	NO	SI	NO	NO
Symantec	SI	SI	SI	SI	SI
TheHacker	SI	NO	SI	SI	NO
TotalDefense	SI	SI	SI	SI	NO
TrendMicro	SI	SI	SI	SI	SI

Tabla 4.4: Resultado Fase 1: *Malware* no cifrado. Ejemplo IV

antivirus/malware	pas3072b	pdfcreator	quake	server	xvidsetup8
Avast	SI	SI	SI	SI	SI
AVG	SI	SI	SI	SI	SI
BitDefender	SI	SI	SI	SI	SI
ByteHero	NO	NO	NO	NO	NO
CAT-QuickHeal	NO	SI	NO	SI	SI
ClamAV	SI	SI	NO	SI	SI
Commtouch	SI	SI	SI	SI	NO
Comodo	SI	SI	SI	SI	SI
DrWeb	SI	SI	SI	SI	SI
Emsisoft	SI	SI	SI	SI	SI
eSafe	SI	SI	SI	SI	NO
ESET-NOD32	SI	SI	SI	NO	SI
F-Prot	SI	SI	SI	SI	SI
F-Secure	SI	SI	SI	SI	SI
Fortinet	SI	SI	SI	SI	SI
GData	SI	SI	SI	SI	SI
Ikarus	SI	SI	SI	SI	SI
Jiangmin	SI	SI	SI	SI	SI
K7AntiVirus	SI	SI	SI	SI	SI
K7GW	SI	NO	SI	SI	SI
Kaspersky	SI	SI	SI	SI	SI
Kingsoft	SI	SI	SI	SI	NO
Malwarebytes	SI	SI	NO	NO	SI
McAfee	SI	SI	SI	NO	SI
McAfee-GW-Edition	SI	SI	SI	NO	SI
Microsoft	NO	SI	SI	NO	SI
MicroWorld-eScan	NO	SI	SI	SI	SI
NANO-Antivirus	NO	SI	NO	NO	NO
Norman	SI	SI	SI	SI	SI
nProtect	SI	SI	SI	NO	SI
Panda	SI	SI	SI	SI	NO
PCTools	SI	SI	SI	SI	SI
Rising	SI	NO	SI	NO	SI
Sophos	SI	SI	SI	SI	SI
SUPERAntiSpyware	NO	NO	NO	NO	SI
Symantec	SI	SI	SI	SI	SI
TheHacker	SI	SI	SI	SI	NO
TotalDefense	SI	SI	NO	NO	SI
TrendMicro	SI	SI	SI	NO	SI

Tabla 4.5: Resultado Fase 1: *Malware* no cifrado. Ejemplo V

antivirus/malware	13th	5120	addict9	cybernet	darksatan
Avast	NO	NO	NO	NO	NO
AVG	NO	NO	NO	NO	NO
BitDefender	NO	NO	NO	NO	NO
ByteHero	NO	NO	NO	NO	NO
CAT-QuickHeal	NO	NO	NO	NO	NO
ClamAV	NO	NO	NO	NO	NO
Commtouch	NO	NO	NO	NO	NO
Comodo	NO	NO	NO	NO	NO
DrWeb	NO	NO	NO	NO	NO
Emsisoft	NO	NO	NO	NO	NO
eSafe	NO	NO	NO	NO	NO
ESET-NOD32	NO	NO	NO	NO	NO
F-Prot	NO	NO	S	NO	NO
F-Secure	NO	NO	NO	NO	NO
Fortinet	NO	NO	NO	NO	NO
GData	NO	NO	NO	NO	NO
Ikarus	NO	NO	NO	NO	NO
Jiangmin	NO	NO	NO	NO	NO
K7AntiVirus	NO	NO	NO	NO	NO
K7GW	NO	NO	NO	NO	NO
Kaspersky	NO	NO	NO	NO	NO
KingsoftNO	NO	NO	NO	NO	
Malwarebytes	NO	NO	NO	NO	NO
McAfee	NO	NO	NO	NO	NO
McAfee-GW-Edition	NO	NO	NO	NO	NO
Microsoft	NO	NO	NO	NO	NO
MicroWorld-eScan	NO	NO	NO	NO	NO
NANO-Antivirus	NO	NO	NO	NO	NO
Norman	NO	NO	NO	NO	NO
nProtect	NO	NO	NO	NO	NO
Panda	NO	NO	NO	NO	NO
PCTools	NO	NO	NO	NO	NO
Rising	NO	NO	NO	NO	NO
Sophos	NO	NO	NO	NO	NO
SUPERAntiSpyware	NO	NO	NO	NO	NO
Symantec	NO	NO	NO	NO	NO
TheHacker	NO	NO	NO	NO	NO
TotalDefense	NO	NO	NO	NO	NO
TrendMicro	NO	NO	NO	NO	NO

Tabla 4.6: Resultado Fase 2: Cifrado XOR con polimorfismo. Ejemplo I

antivirus/malware	diamon.gr'	elvis	erasmus	glitch	guerilla
Avast	NO	NO	NO	NO	NO
AVG	NO	NO	NO	NO	NO
BitDefender	NO	NO	NO	NO	NO
ByteHero	NO	NO	NO	NO	NO
CAT-QuickHeal	NO	NO	NO	NO	NO
ClamAV	NO	NO	NO	NO	NO
Commtouch	NO	NO	NO	NO	NO
Comodo	NO	NO	NO	NO	NO
DrWeb	NO	NO	NO	NO	NO
EmNOsoft	NO	NO	NO	NO	NO
eSafe	NO	NO	NO	NO	NO
ESET-NOD32	NO	NO	NO	NO	NO
F-Prot	NO	NO	NO	NO	NO
F-Secure	NO	NO	NO	NO	NO
Fortinet	NO	NO	NO	NO	NO
GData	NO	NO	NO	NO	NO
Ikarus	NO	NO	NO	NO	NO
Jiangmin	NO	NO	NO	NO	NO
K7AntiVirus	NO	NO	NO	NO	NO
K7GW	NO	NO	NO	NO	NO
Kaspersky	NO	NO	NO	NO	NO
Kingsoft	NO	NO	NO	NO	NO
Malwarebytes	NO	NO	NO	NO	NO
McAfee	NO	NO	NO	NO	NO
McAfee-GW-Edition	NO	NO	NO	NO	NO
Microsoft	NO	NO	NO	NO	NO
MicroWorld-eScan	NO	NO	NO	NO	NO
NANO-Antivirus	NO	NO	NO	NO	NO
Norman	NO	NO	NO	NO	NO
nProtect	NO	NO	NO	NO	NO
Panda	NO	NO	NO	NO	NO
PCTools	NO	NO	NO	NO	NO
RiNOng	NO	NO	NO	NO	NO
Sophos	NO	NO	NO	NO	NO
SUPERAntiSpyware	NO	NO	NO	NO	NO
Symantec	NO	NO	NO	NO	NO
TheHacker	NO	NO	NO	NO	NO
TotalDefense	NO	NO	NO	NO	NO
TrendMicro	NO	NO	NO	NO	NO

Tabla 4.7: Resultado Fase 2: Cifrado XOR con polimorfismo. Ejemplo II

antivirus/malware	halloween1376	harakiri	hero394	ifreetv	int1988
Avast	NO	NO	NO	NO	NO
AVG	NO	NO	NO	NO	NO
BitDefender	NO	NO	NO	NO	NO
ByteHero	NO	NO	NO	NO	NO
CAT-QuickHeal	NO	NO	NO	NO	NO
ClamAV	NO	NO	NO	NO	NO
Commtouch	NO	NO	NO	NO	NO
Comodo	NO	NO	NO	NO	NO
DrWeb	NO	NO	NO	NO	NO
EmNOsoft	NO	NO	NO	NO	NO
eSafe	NO	NO	NO	NO	NO
ESET-NOD32	NO	NO	NO	NO	NO
F-Prot	NO	NO	NO	NO	NO
F-Secure	NO	NO	NO	NO	NO
Fortinet	NO	NO	NO	NO	NO
GData	NO	NO	NO	NO	NO
Ikarus	NO	NO	NO	NO	NO
Jiangmin	NO	NO	NO	NO	NO
K7AntiVirus	NO	NO	NO	NO	NO
K7GW	NO	NO	NO	NO	NO
Kaspersky	NO	NO	NO	NO	NO
Kingsoft	NO	NO	NO	NO	NO
Malwarebytes	NO	NO	NO	NO	NO
McAfee	NO	NO	NO	NO	NO
McAfee-GW-Edition	NO	NO	NO	NO	NO
Microsoft	NO	NO	NO	NO	NO
MicroWorld-eScan	NO	NO	NO	NO	NO
NANO-Antivirus	NO	NO	NO	NO	NO
Norman	NO	NO	NO	NO	NO
nProtect	NO	NO	NO	NO	NO
Panda	NO	NO	NO	NO	NO
PCTools	NO	NO	NO	NO	NO
RiNOng	NO	NO	NO	NO	NO
Sophos	NO	NO	NO	NO	NO
SUPERAntiSpyware	NO	NO	NO	NO	NO
Symantec	NO	NO	NO	NO	NO
TheHacker	NO	NO	NO	NO	NO
TotalDefense	NO	NO	NO	NO	NO
TrendMicro	NO	NO	NO	NO	NO

Tabla 4.8: Resultado Fase 2: Cifrado XOR con polimorfismo. Ejemplo III

antivirus/malware	intrud1	jerusalem	jsr3a	manequin	nextgen
Avast	NO	NO	NO	NO	NO
AVG	NO	NO	NO	NO	NO
BitDefender	NO	NO	NO	NO	NO
ByteHero	NO	NO	NO	NO	NO
CAT-QuickHeal	NO	NO	NO	NO	NO
ClamAV	NO	NO	NO	NO	NO
Commtouch	NO	NO	NO	NO	NO
Comodo	NO	NO	NO	NO	NO
DrWeb	NO	NO	NO	NO	NO
EmNOsoft	NO	NO	NO	NO	NO
eSafe	NO	NO	NO	NO	NO
ESET-NOD32	NO	NO	NO	NO	NO
F-Prot	NO	NO	NO	NO	NO
F-Secure	NO	NO	NO	NO	NO
Fortinet	NO	NO	NO	NO	NO
GData	NO	NO	NO	NO	NO
Ikarus	NO	NO	NO	NO	NO
Jiangmin	NO	NO	NO	NO	NO
K7AntiVirus	NO	NO	NO	NO	NO
K7GW	NO	NO	NO	NO	NO
Kaspersky	NO	NO	NO	NO	NO
Kingsoft	NO	NO	NO	NO	NO
Malwarebytes	NO	NO	NO	NO	NO
McAfee	NO	NO	NO	NO	NO
McAfee-GW-Edition	NO	NO	NO	NO	NO
Microsoft	NO	NO	NO	NO	NO
MicroWorld-eScan	NO	NO	NO	NO	NO
NANO-Antivirus	NO	NO	NO	NO	NO
Norman	NO	NO	NO	NO	NO
nProtect	NO	NO	NO	NO	NO
Panda	NO	NO	NO	NO	NO
PCTools	NO	NO	NO	NO	NO
RiNOng	NO	NO	NO	NO	NO
Sophos	NO	NO	NO	NO	NO
SUPERAntiSpyware	NO	NO	NO	NO	NO
Symantec	NO	NO	NO	NO	NO
TheHacker	NO	NO	NO	NO	NO
TotalDefense	NO	NO	NO	NO	NO
TrendMicro	NO	NO	NO	NO	NO

Tabla 4.9: Resultado Fase 2: Cifrado XOR con polimorfismo. Ejemplo IV

antivirus/malware	pas3072b	pdfcreator	quake	server	xvidsetup8
Avast	NO	NO	NO	NO	NO
AVG	NO	NO	NO	NO	NO
BitDefender	NO	NO	NO	NO	NO
ByteHero	NO	NO	NO	NO	NO
CAT-QuickHeal	NO	NO	NO	NO	NO
ClamAV	NO	NO	NO	NO	NO
Commtouch	NO	NO	NO	NO	NO
Comodo	NO	NO	NO	NO	NO
DrWeb	NO	NO	NO	NO	NO
EmNOsoft	NO	NO	NO	NO	NO
eSafe	NO	NO	NO	NO	NO
ESET-NOD32	NO	NO	NO	NO	NO
F-Prot	NO	NO	NO	NO	NO
F-Secure	NO	NO	NO	NO	NO
Fortinet	NO	NO	NO	NO	NO
GData	NO	NO	NO	NO	NO
Ikarus	NO	NO	NO	NO	NO
Jiangmin	NO	NO	NO	NO	NO
K7AntiVirus	NO	NO	NO	NO	NO
K7GW	NO	NO	NO	NO	NO
Kaspersky	NO	NO	NO	NO	NO
Kingsoft	NO	NO	NO	NO	NO
Malwarebytes	NO	NO	NO	NO	NO
McAfee	NO	NO	NO	NO	NO
McAfee-GW-Edition	NO	NO	NO	NO	NO
Microsoft	NO	NO	NO	NO	NO
MicroWorld-eScan	NO	NO	NO	NO	NO
NANO-Antivirus	NO	NO	NO	NO	NO
Norman	NO	NO	NO	NO	NO
nProtect	NO	NO	NO	NO	NO
Panda	NO	NO	NO	NO	NO
PCTools	NO	NO	NO	NO	NO
RiNOng	NO	NO	NO	NO	NO
Sophos	NO	NO	NO	NO	NO
SUPERAntiSpyware	NO	NO	NO	NO	NO
Symantec	NO	NO	NO	NO	NO
TheHacker	NO	NO	NO	NO	NO
TotalDefense	NO	NO	NO	NO	NO
TrendMicro	NO	NO	NO	NO	NO

Tabla 4.10: Resultado Fase 2: Cifrado XOR con polimorfismo. Ejemplo V

antivirus/malware	13th	5120	addict9	cybernet	darksatan
Avast	NO	NO	NO	NO	NO
AVG	NO	NO	NO	NO	NO
BitDefender	NO	NO	NO	NO	NO
ByteHero	NO	NO	NO	NO	NO
CAT-QuickHeal	NO	NO	NO	NO	NO
ClamAV	NO	NO	NO	NO	NO
Commtouch	NO	NO	NO	NO	NO
Comodo	NO	NO	NO	NO	NO
DrWeb	NO	NO	NO	NO	NO
Emsisoft	NO	NO	NO	NO	NO
eSafe	NO	NO	NO	NO	NO
ESET-NOD32	NO	NO	NO	NO	NO
F-Prot	NO	NO	S	NO	NO
F-Secure	NO	NO	NO	NO	NO
Fortinet	NO	NO	NO	NO	NO
GData	NO	NO	NO	NO	NO
Ikarus	NO	NO	NO	NO	NO
Jiangmin	NO	NO	NO	NO	NO
K7AntiVirus	NO	NO	NO	NO	NO
K7GW	NO	NO	NO	NO	NO
Kaspersky	NO	NO	NO	NO	NO
Kingsoft	NO	NO	NO	NO	NO
Malwarebytes	NO	NO	NO	NO	NO
McAfee	NO	NO	NO	NO	NO
McAfee-GW-Edition	NO	NO	NO	NO	NO
Microsoft	NO	NO	NO	NO	NO
MicroWorld-eScan	NO	NO	NO	NO	NO
NANO-Antivirus	NO	NO	NO	NO	NO
Norman	NO	NO	NO	NO	NO
nProtect	NO	NO	NO	NO	NO
Panda	NO	NO	NO	NO	NO
PCTools	NO	NO	NO	NO	NO
Rising	NO	NO	NO	NO	NO
Sophos	NO	NO	NO	NO	NO
SUPERAntiSpyware	NO	NO	NO	NO	NO
Symantec	NO	NO	NO	NO	NO
TheHacker	NO	NO	NO	NO	NO
TotalDefense	NO	NO	NO	NO	NO
TrendMicro	NO	NO	NO	NO	NO

Tabla 4.11: Resultado Fase 2: Cifrado AES con polimorfismo. Ejemplo I

antivirus/malware	diamon.gr	elvis	erasmus	glitch	guerilla
Avast	NO	NO	NO	NO	NO
AVG	NO	NO	NO	NO	NO
BitDefender	NO	NO	NO	NO	NO
ByteHero	NO	NO	NO	NO	NO
CAT-QuickHeal	NO	NO	NO	NO	NO
ClamAV	NO	NO	NO	NO	NO
Commtouch	NO	NO	NO	NO	NO
Comodo	NO	NO	NO	NO	NO
DrWeb	NO	NO	NO	NO	NO
EmNOsoft	NO	NO	NO	NO	NO
eSafe	NO	NO	NO	NO	NO
ESET-NOD32	NO	NO	NO	NO	NO
F-Prot	NO	NO	NO	NO	NO
F-Secure	NO	NO	NO	NO	NO
Fortinet	NO	NO	NO	NO	NO
GData	NO	NO	NO	NO	NO
Ikarus	NO	NO	NO	NO	NO
Jiangmin	NO	NO	NO	NO	NO
K7AntiVirus	NO	NO	NO	NO	NO
K7GW	NO	NO	NO	NO	NO
Kaspersky	NO	NO	NO	NO	NO
Kingsoft	NO	NO	NO	NO	NO
Malwarebytes	NO	NO	NO	NO	NO
McAfee	NO	NO	NO	NO	NO
McAfee-GW-Edition	NO	NO	NO	NO	NO
Microsoft	NO	NO	NO	NO	NO
MicroWorld-eScan	NO	NO	NO	NO	NO
NANO-Antivirus	NO	NO	NO	NO	NO
Norman	NO	NO	NO	NO	NO
nProtect	NO	NO	NO	NO	NO
Panda	NO	NO	NO	NO	NO
PCTools	NO	NO	NO	NO	NO
RiNOng	NO	NO	NO	NO	NO
Sophos	NO	NO	NO	NO	NO
SUPERAntiSpyware	NO	NO	NO	NO	NO
Symantec	NO	NO	NO	NO	NO
TheHacker	NO	NO	NO	NO	NO
TotalDefense	NO	NO	NO	NO	NO
TrendMicro	NO	NO	NO	NO	NO

Tabla 4.12: Resultado Fase 2: Cifrado AES con polimorfismo. Ejemplo II

antivirus/malware	halloween1376	harakiri	hero394	ifreetv	int1988
Avast	NO	NO	NO	NO	NO
AVG	NO	NO	NO	NO	NO
BitDefender	NO	NO	NO	NO	NO
ByteHero	NO	NO	NO	NO	NO
CAT-QuickHeal	NO	NO	NO	NO	NO
ClamAV	NO	NO	NO	NO	NO
Commtouch	NO	NO	NO	NO	NO
Comodo	NO	NO	NO	NO	NO
DrWeb	NO	NO	NO	NO	NO
EmNOsoft	NO	NO	NO	NO	NO
eSafe	NO	NO	NO	NO	NO
ESET-NOD32	NO	NO	NO	NO	NO
F-Prot	NO	NO	NO	NO	NO
F-Secure	NO	NO	NO	NO	NO
Fortinet	NO	NO	NO	NO	NO
GData	NO	NO	NO	NO	NO
Ikarus	NO	NO	NO	NO	NO
Jiangmin	NO	NO	NO	NO	NO
K7AntiVirus	NO	NO	NO	NO	NO
K7GW	NO	NO	NO	NO	NO
Kaspersky	NO	NO	NO	NO	NO
Kingsoft	NO	NO	NO	NO	NO
Malwarebytes	NO	NO	NO	NO	NO
McAfee	NO	NO	NO	NO	NO
McAfee-GW-Edition	NO	NO	NO	NO	NO
Microsoft	NO	NO	NO	NO	NO
MicroWorld-eScan	NO	NO	NO	NO	NO
NANO-Antivirus	NO	NO	NO	NO	NO
Norman	NO	NO	NO	NO	NO
nProtect	NO	NO	NO	NO	NO
Panda	NO	NO	NO	NO	NO
PCTools	NO	NO	NO	NO	NO
RiNOng	NO	NO	NO	NO	NO
Sophos	NO	NO	NO	NO	NO
SUPERAntiSpyware	NO	NO	NO	NO	NO
Symantec	NO	NO	NO	NO	NO
TheHacker	NO	NO	NO	NO	NO
TotalDefense	NO	NO	NO	NO	NO
TrendMicro	NO	NO	NO	NO	NO

Tabla 4.13: Resultado Fase 2: Cifrado AES con polimorfismo. Ejemplo III

antivirus/malware	intrud1	jerusalem	jsr3a	manequin	nextgen
Avast	NO	NO	NO	NO	NO
AVG	NO	NO	NO	NO	NO
BitDefender	NO	NO	NO	NO	NO
ByteHero	NO	NO	NO	NO	NO
CAT-QuickHeal	NO	NO	NO	NO	NO
ClamAV	NO	NO	NO	NO	NO
Commtouch	NO	NO	NO	NO	NO
Comodo	NO	NO	NO	NO	NO
DrWeb	NO	NO	NO	NO	NO
EmNOsoft	NO	NO	NO	NO	NO
eSafe	NO	NO	NO	NO	NO
ESET-NOD32	NO	NO	NO	NO	NO
F-Prot	NO	NO	NO	NO	NO
F-Secure	NO	NO	NO	NO	NO
Fortinet	NO	NO	NO	NO	NO
GData	NO	NO	NO	NO	NO
Ikarus	NO	NO	NO	NO	NO
Jiangmin	NO	NO	NO	NO	NO
K7AntiVirus	NO	NO	NO	NO	NO
K7GW	NO	NO	NO	NO	NO
Kaspersky	NO	NO	NO	NO	NO
Kingsoft	NO	NO	NO	NO	NO
Malwarebytes	NO	NO	NO	NO	NO
McAfee	NO	NO	NO	NO	NO
McAfee-GW-Edition	NO	NO	NO	NO	NO
Microsoft	NO	NO	NO	NO	NO
MicroWorld-eScan	NO	NO	NO	NO	NO
NANO-Antivirus	NO	NO	NO	NO	NO
Norman	NO	NO	NO	NO	NO
nProtect	NO	NO	NO	NO	NO
Panda	NO	NO	NO	NO	NO
PCTools	NO	NO	NO	NO	NO
RiNOng	NO	NO	NO	NO	NO
Sophos	NO	NO	NO	NO	NO
SUPERAntiSpyware	NO	NO	NO	NO	NO
Symantec	NO	NO	NO	NO	NO
TheHacker	NO	NO	NO	NO	NO
TotalDefense	NO	NO	NO	NO	NO
TrendMicro	NO	NO	NO	NO	NO

Tabla 4.14: Resultado Fase 2: Cifrado AES con polimorfismo. Ejemplo IV

antivirus/malware	pas3072b	pdfcreator	quake	server	xvidsetup8
Avast	NO	NO	NO	NO	NO
AVG	NO	NO	NO	NO	NO
BitDefender	NO	NO	NO	NO	NO
ByteHero	NO	NO	NO	NO	NO
CAT-QuickHeal	NO	NO	NO	NO	NO
ClamAV	NO	NO	NO	NO	NO
Commtouch	NO	NO	NO	NO	NO
Comodo	NO	NO	NO	NO	NO
DrWeb	NO	NO	NO	NO	NO
EmNOsoft	NO	NO	NO	NO	NO
eSafe	NO	NO	NO	NO	NO
ESET-NOD32	NO	NO	NO	NO	NO
F-Prot	NO	NO	NO	NO	NO
F-Secure	NO	NO	NO	NO	NO
Fortinet	NO	NO	NO	NO	NO
GData	NO	NO	NO	NO	NO
Ikarus	NO	NO	NO	NO	NO
Jiangmin	NO	NO	NO	NO	NO
K7AntiVirus	NO	NO	NO	NO	NO
K7GW	NO	NO	NO	NO	NO
Kaspersky	NO	NO	NO	NO	NO
Kingsoft	NO	NO	NO	NO	NO
Malwarebytes	NO	NO	NO	NO	NO
McAfee	NO	NO	NO	NO	NO
McAfee-GW-Edition	NO	NO	NO	NO	NO
Microsoft	NO	NO	NO	NO	NO
MicroWorld-eScan	NO	NO	NO	NO	NO
NANO-Antivirus	NO	NO	NO	NO	NO
Norman	NO	NO	NO	NO	NO
nProtect	NO	NO	NO	NO	NO
Panda	NO	NO	NO	NO	NO
PCTools	NO	NO	NO	NO	NO
RiNOng	NO	NO	NO	NO	NO
Sophos	NO	NO	NO	NO	NO
SUPERAntiSpyware	NO	NO	NO	NO	NO
Symantec	NO	NO	NO	NO	NO
TheHacker	NO	NO	NO	NO	NO
TotalDefense	NO	NO	NO	NO	NO
TrendMicro	NO	NO	NO	NO	NO

Tabla 4.15: Resultado Fase 2: Cifrado AES con polimorfismo. Ejemplo V

Capítulo 5

Conclusiones finales y propuestas de mejora

5.1. Conclusiones sobre los resultados obtenidos

Una vez realizadas las pruebas de cifrado y polimorfismo y teniendo presente los resultados obtenidos se puede concluir que la aplicación TYNIDS que es el objeto de este trabajo, aporta un resultado excelente ya que al ofuscar el *malware* por cualquiera de los métodos que permite la aplicación, ya sea AES, XOR o 3DES, tanto si aplicamos polimorfismo como si no, ningún *malware* ha sido detectado por los antivirus con los que se han realizado las pruebas.

Se ha obtenido una tasa de eficacia del 100 %.

5.2. Conclusiones sobre el proyecto

La asignatura de sistemas informáticos no sólo ha servido de vía para superar el proyecto fin de carrera, si no que ha supuesto una oportunidad para abordar líneas de investigación que el equipo de trabajo desconocía como por ejemplo las diversas técnicas de cifrado de código que se han incluido en este proyecto. Tampoco se tenía un adecuado conocimiento del uso que aportaba un sistema NIDS y para lograr evadirlo se ha tenido que investigar su funcionamiento y atacar sus puntos débiles usando potentes técnicas de cifrado de código y estrategias de polimorfismo.

Como se ha podido observar, la evasión de un NIDS no es una tarea sencilla. Hacer de ello una tarea manual resulta tedioso a la par que propenso a errores. Para ahorrar tiempo y molestias a la hora de probar un NIDS, se ha implementado una herramienta que automatiza la ofuscación del *malware* en grandes cantidades, solo hay que especificar el directorio donde se encuentra dicho *malware* y la herramienta trata cada fichero de forma independiente aplicándole todo el proceso de ofuscación mencionado en el capítulo 3.

5.3. Posibles usos de la herramienta. Educativo / Comercial

En esta sección se describe los posibles usos que se han pensado que se podría hacer con la herramienta final. Existen dos vertientes a abordar, una sería un uso educativo y

otro el comercial.

Este proyecto surge como consecuencia de la realización de la asignatura de Sistemas Informáticos, por lo que TYNIDS se podría utilizar para hacer pruebas y mejoras sobre sistemas NIDS. Si a futuro, surge un proyecto que consista en mejorar un sistema NIDS, nuestra herramienta puede ser clave en la consecución del mismo, pues continuamente puede usarse para generar código con el que probar las distintas reglas por las que se rige el sistema de intrusos. Otro punto de vista es el comercial y es que cómo cada día que pasa, la necesidad de seguridad en sistemas comerciales es mayor, empresas podrían comprar licencias de nuestra herramienta para probar sus NIDS y estar así continuamente mejorando sus sistemas de seguridad de cara a posibles amenazas procedentes de la red.

5.4. Propuestas para futuros trabajos

A lo largo de proyecto han ido surgiendo necesidades que se han ido abordando de la mejor manera posible, además de problemas que se han tenido que solucionar. Por lo tanto han ido quedando mejoras y optimizaciones que no ha dado tiempo a incluir en el proyecto debido al tiempo necesario en investigación e implementación de las mismas. Así proponemos para futuras versiones de esta herramienta los siguientes puntos:

- Generación de más instrucciones de código basura a la hora de realizar el polimorfismo.
- Inserción de más métodos de polimorfismo como por ejemplo sustitución de variables/registros y transposición de código.
- Inclusión de la técnica del metamorfismo descrita en este proyecto.
- Desarrollo de nuevas estrategias y métodos de cifrado e inclusión de nuevos algoritmos.

Bibliografía

[1] D. Caruso: "¿Qué es Internet? Localización de información específica en la web", Universidad Politécnica de Valencia, España, 2005.

[2] E. Maiwald, E.A Miguel: "Fundamentos de seguridad de redes". McGraw-Hill, 2000.

[3] D. Gonzalez Gómez: "Sistemas de Detección de Intrusiones" diponible en: http://www.dggomez.arrakis.es/secinf/ids/html/cap01.htm, Julio 2003.

[4] E. Gallego, J. E. Lopez de Vergara: "Honeynets, Aprendiendo del atacante", Universidad Politécnica de Madrid, Madrid, España, 2004.

[5] L. F. Fuentes: "Malware, una amenaza de Internet", Revista Digital Universitario, Vol. 9 No. 4, 1-9, Universidad Autónoma de México, México, 2008.

[6] R. Heady, G. Luger, A. Maccabe, M. Servilla:'The Architecture of a Network Level Intrusion Detection System'. Technical report, Department of Computer Science, University of New Mexico, Agosto 1990.

[7] J. Maestre Vidal, H. Villanúa Vega, J. A. García Guijarro: "Sistema de detección de anomalías de red basado en el procesamiento de la Carga Útil [Payload]". Facultad de Informática, Universidad Complutense de Madrid, Madrid, España, 2012.

[8] X.Zhang, C. Li, W. Zheng: "Intrusion prevention system design", Computer and Information Technology, 2004. CIT'04. The Fourth International Conference on (pp. 386-390) IEEE, Septiembre 2004.

[9] A. Gramajo: "Introducción a conceptos de IDS y técnicas avanzadas con Snort", Jornadas de Software libre 2005. [Online] http://www.baicom.com/eventos/010905.pdf.

[10] D .Dolz, D, G. Parra: "Ofuscadores de código intermedio", VIII Workshop de Investigadores en Ciencias de la Computación, Buenos Aires, Argentina, p 703-707, Junio 2006.

[11] I. Santos: "Nuevo Enfoque para la Detección de Malware basado en Métodos de Recuperación de Información" Universidad de Deusto, Bilbao, España, Junio 2011.

[12] T. Sommestad, A. Hunstad: "Intrusion detection and the role of the system administrator", Information Management & Computer Security, 21(1), 3-3, 2003.

[13] Y. Yong Sun, G. Qiu Huang: "Technique of Code Obfuscation Based on Class Structure", Applied Mechanics and Materials, 271, 674-678, Diciembre 2012.

[14] A. P. Sample, J. R. Smith: "The wireless identification and sensing platform" Wirelessly Powered Sensor Networks and Computational RFID, 33-56, Springer New York 2013.

[15] J. Zhang and G. Gu: "Neighborwatcher: A content-agnostic comment spam inference system", In Proceeding of the Network and Distributed System Security Symposium (NDSS'13), 2013.

[16] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang: "On the impossibility of obfuscating programs". In Advances in Cryptology - CRYPTO 2001, volume 2139 of Lecture Notes in Computer Science. Pag 1-18. Springer-Verlag, 2001. Versión completa disponible en: http://eprint.iacr.org/2001/069/.

[17] M. Berón, P. Rangel Henriques, M. J. Varanda Pereira, R. Uzal "Herramientas para la comprensión de programas" In VIII Workshop de Investigadores en Ciencias de la Computación. Junio 2006.

[18] C. Collberg, C. Thomborson, D.Low: "A Taxonomy of Obfuscating Transformations", Dept of Computer Science, Universidad de Auckland, Julio 1997. Disponible en: http://www.cs.auckland.ac.nz/ collberg/Research/Publications/CollbergThomborsonLow97a.

[19] I. Jacobson, G. Booch, J. Rumbaugh: "El proceso unificado de desarrollo de software" (Vol. 7). Addison Wesley 2000.

[20] A. Miralles Arévalo, A. J. Sierra Collado: "Estudio y Prueba de una Métrica para los Ofuscadores Java" Proyecto de Fin de Carrera, Universidad de Sevilla, Sevilla, España, Mayo 2005.

[21] B. Mukherjee:"Threats to Digitization: Computer Virus", International CALIBER, Febrero 2008.

[22] R. Gómez Cárdenas, Plaza, R. C. Lira Plaza, A. Grego: "Esquema de Almacenamiento Seguro de Llaves Criptográficas" Departamento de Ciencias computacionales, Instituto Tecnológico y de Estudios Superiores de Monterrey-Campus Edo, México, 2001.

[23] S. Woolgar, G. Russell: "Las bases sociales de los virus informáticos" Política y sociedad 14, 171-196, 1993.

[24] A. Mosquera Quinto, R. Zulúaga, A. Guillermo: "Guía de referencia: los antivirus y sus tendencias futuras", Universidad Tecnológica de Pereira, Colombia 2011.

[25] P. Ször, P. Ferrie: "Hunting for metamorphic", Virus Bulletin Conference, Symantec Corporation, Santa Monica, USA, Septiembre 2001.

[26] J. C. Sarmiento Tovilla: "Un modelo del sistema inmune para prevenir y eliminar los virus informáticos". Tesis Doctoral, Instituto Politécnico Nacional, Centro de Investigación en Computación. Mexico 2003.

[27] P. Barder: "Conficker, la última amenaza en Internet: los gusanos atacan de nuevo". PC world profesional, (264), 76-81. 2009

[28] O. Colomina Pardo, P. Arques Corrales, J. Montoyo Bojo: "Introducción a Javascript. Manejo de eventos", Tema 3, parte 1, Tecnologías Web, 2011.

[29] A. SEO.: "Auditoria, Pagina Web Con Certificado Digital" Disponible en: http://www.asociacionseo.com/auditoria-pagina-web-con-certificado-digital/, 2012.

[30] M. T. J. García, C. M. Pérez, A. M. M. García, J. P. Agudín: "Hacker", Anaya, ISBN: 978-84-415-2323-4 84-415-2323-1, España 2008.

[31] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, N. Weaver: "Inside the slammer worm" Security & Privacy, IEEE, 1(4), 33-39. 2003

[32] J. Bradenbaugh: "Aplicaciones JavaScript" Anaya, España, 2000.

[33] E. Conde: "Introducción a los lenguajes del web" Disponible en: http://www.ipereda.com/descargas/manuales/php/1.-Manual%20de%20Introducci%C3%B3n%20Lenguajes%20Web%20-%2022%20pag.pdf, 2006.

[34] L. Sánchez-Guerrero, L., J. R. Lira-Cortés, M. Henaine-Abed: "Algoritmos para Encriptamiento y Desencritar en Diferentes Lenguajes (Lenguaje Java, Perl y C#)", Universidad Autónoma Metropolitana, Departamento de Sistemas, Mexico, 2011.

[35] G. Antichi, D. Ficara, S. Giordano, G. Procissi, F. Vitucc: "Counting Bloom Filters for Pattern Matching and Anti-Evasion at the Wire Speed" IEEE Network Magazine of Global Internetworking. Vol. 23, no. 1, 30-35. 2009.

[36] D. Son. Fragroute. [Online]. http://www.monkey.org/dugsong/fragroute/ 2002.

[37] M. Handley, C. Kreibich, V. Paxson: "Network intrusion detection: Evasion, traffic normalization and end-to-end protocol semantics", Proceedings of the 10th Conference on USENIX Security Symposium, Volume 10, 9. 2001.

[38] Nikto. [Online]. HYPERLINK http://www.cirt.net/code/nikto.shtml

[39] Nmap. [Online]. HYPERLINK http://nmap.org/

[40] D. Watson, M. Smart, R. G. Malan, and F. Jahanian: "Protocol scrubbing: network security through transparent flow modification" IEEE/ACM Transactions on Networking. Vol. 12. 261-273. 2004.

[41] M. Vutukuru, H. Balakrishnan, V. Paxson: "Efficient and Robust TCP Stream Normalization", SP '08: Proceedings of the 2008 IEEE Symposium on Security and Privacy, Washington DC. 96-110, 2008.

[42] G. Varghese, J. A. Fingerhut, F. Bonomi: "Detecting evasion attacks at high speeds without reassembly", SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications 327-338, Pisa, Italia 2006.

[43] U. Shankar, V. Paxson, Active Mapping: "Resisting NIDS Evasion without Altering Traffic" SP '03: Proceedings of the 2003 IEEE Symposium on Security and Privacy, Washington, DC, USA, 44. 2003.

[44] M. Roesch: "Snort - Lightweight Intrusion Detection for Networks", LISA '99: Proceedings of the 13th USENIX conference on System administration 229-238, Seattle, Washington, 1999.

[45] D. Mutz, C. Kruegel, W. Robertson, G. Vigna, R. A. Kemmerer: "Reverse Engineering of Network Signatures" AusCERT Asia Pacific Information Technology Security Conference, Gold 2005.

[46] T. H. Ptacek, T. N. Newsham: "Insertion, evasion and denial of service: Eluding network intrusion detection", Technical report, 1998.

[47] J. Maestre Vidal, J. D. Mejía Castro, A. L. Sandoval Orozco, L. J. García Villalba: "Evolutions of evasion techniques against network instrusion detection systems" ICIT 2013 The 6tm International conference on Information Technology, 2013.

[48] W. G. J. Halfond, A. Orso: "AMNESIA: analysis and monitoring for Neutralizing SQL-injection attacks" 174-183, California, USA 2005.

[49] S. Ali, A. Rauf, H. Javed: "SQLIPA: An authentication mechanism Against SQL Injection" European Journal of Scientific Research, Vol.38 No.4 pp 604-611, 2009.

[50] W.Lin: "Study on the principle and defense of buffer overflow attacks." 2012 International Conference on Graphic and Image Processing, International Society for Optics and Photonics, 2013.

[51] ANSI: "Triple Data Encryption Algorithm Modes of Operation" American National Standards Institute X9.52-1998, American Bankers Association, Washington DC, July 29, 1998.

[52] FIPS: "Data Encryption Standard", Federal Information Processing Standards Publication 46-3, available at: http://csrc.nist.gov/encryption/tkencryption.html th. IEEE/ACM International Conference on Automated Software Engineering, Long 111, October 1999.

[53] U. Zurutuza, R. Uribeetxeberria: "Revisión del estado actual de la investigación en el uso de data mining para la detección de intrusiones"Escuela Politécnica Superior de Mondragón, Departamento Informática, España 2005.

[54] J. Gómez López: "Optimización de sistemas de detección de intrusos en red utilizando técnicas computacionales avanzadas", Editorial Universidad de Almería, España, 2009

[55] U. Zurutuza, "Estado del arte Sistemas de Detección de intrusos", Universidad de Mondragón, Guipúzcoa, España, 2004.

[56] D. López, O. Pastor, L. J. Garcia Villalba: "Concepto y Enfoques sobre el Análisis y la Gestión Dinámica del Riesgo en Sistemas de Información", Actas de la XII Reunión Española de Criptologia y Seguridad de la Información (RECSI 2012), Donostia-San Sebastián, España. 2012.

[57] L. Juan, C. Kreibich, C. H. Lin, V. Paxson: "A Tool for Offline and Live Testing of Evasion Resilience in Network Intrusion Detection Systems" DIMVA '08: Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment pp. 267-278, Paris, France, 2008.

[58] P. García-Teodoro, J. E. Díaz-Verdejo, G. Maciá-Fernández, F. J. de Toro-Negro, C. Antas Vilanova: "Detección Híbrida de Intrusiones en Red y Esquemas de Respuesta Activa", 2007.

[59] R. C. Santos: "Detección de intrusos mediante técnicas de minería de datos" Revista Clepsidra, Vol. 2, pp. 31, 2006.

[60] R. Monroy, L. A. Trejo, C. Mex, J. A. Nolazco: "On the Timely Detection of Mimicry Attacks", Departamento de ciencias computacionales, Campus Estado de México del Tecnológico de Monterrey, Programa FRIDA, Junio 2006.

[61] J. Argente Ferrero, R. García González, J. Martínez Puentes: "Sistema híbrido para la detección de código malicioso". Facultad de Informática, Universidad Complutense de Madrid, Madrid 2009.

[62] D. Campoy Miñarro: "Propuesta teórica de Webmining para Detección de Intrusos mediante Lógica Borrosa", Trabajo Final de Máster, Universitat Oberta de Catalunya, España 2011

Apéndices

Apéndices A

Apéndice

A.1. Un repaso a la historia del *Malware*

El surgimiento de estas amenazas puede ubicarse a finales de los años 80, más precisamente en 1988, cuando apareció el gusano Morris, considerado el primero de su especie. Este, logró colapsar cientos de sistemas pertenecientes a ArpaNet (la predecesora de Internet). En el año 2000 aparece el generador de gusanos VBSWG (*Visual Basic Script Worm Generator*), que dio lugar a algunos *worms* importantes, como por ejemplo el gusano "Anna Kournikova" que infectó a la NASA. Este mismo año nació otro gusano, "ILOVEYOU", que logró infectar millones de máquinas a través de correos electrónicos. En 2001 aparece el primer *malware* escrito en PHP, "Pirus". También se desarrollaron gusanos importantes para Linux, como "Ramen" y "Lion".

En 2003, el gusano "Slammer" que explotaba una vulnerabilidad del servidor Microsoft SQL, logró infectar en solo 10 minutos el 90 % de las máquinas vulnerables. Otro *malware* destacable de este año fue "Blaster". En el año 2004 aparece "Mydoom", este gusano se caracterizaba porque podía propagarse tanto por *email* como por la red *Kazaa*, permitiendo el control de los computadores infectados. También aparece "Brador" un troyano para Windows CE y "MP3Concept", el primer troyano para MAC OS X. En 2006 destaca "Stardust" que infectaba macros de los paquetes OpenOffice y Startoffice y "Leap" otro *malware* que afecta a Mac OS X.

En 2008 aparece el gusano "Conficker" que explotaba una vulnerabilidad de los sistemas Windows. En el año 2010, se ha creado por el momento un 34 % de todo el *malware* que se ha desarrollado en la historia, un dato importante. A mediados de año se descubre "Stuxnet", gusano que afecta a equipos con Windows y es capaz de reprogramar sistemas industriales SCADA pudiendo afectar a infraestructuras críticas (centrales nucleares, depuradoras, etc.)

En Diciembre de 2010, nace el troyano "Flu" como una herramienta libre y educativa para formar en materia de seguridad y servir en auditoría de seguridad. Y un mes más tarde, en Enero de 2011, nace su variante "Flu-AD", diseñada especialmente para colaborar con las fuerzas de seguridad para combatir los casos de *cibergrooming* en *Internet*.

Ref.: http://www.flu-project.com/un-repaso-a-la-historia-del-malware-vi-de-vi.html

A.2. El crecimiento del *malware* es exponencial

El incremento de los ciberataques es una realidad que nos afecta a todos. Kaspersky Lab detectaba hace un año 70.000 muestras de *malware* al día. La cifra, que ya ascendía a 125.000 en el mes de mayo, ahora llega a 200.000. Que los programas maliciosos crezcan en este periodo de tiempo un 285 % es sorprendente a la vez que preocupante. Ya no está a salvo ningún sistema operativo en ningún dispositivo con conexión a *Internet* sin la correcta protección.

Algunos datos de Kaspersky Lab a tener en cuenta:

En 2012 se bloquearon más de 1,5 millones de ataques basados en Web, cifra 1,7 veces superior a la del año 2011. Se detectaron más de 3 millones de infecciones locales en equipos de usuarios en 2012. El *malware* Mac OS X sigue aumentando. Kaspersky Lab creó un 30 % más de parches que en 2011.

Una de las noticias más importantes de 2012 fue el descubrimiento de "Flashback", una potente *botnet* formada por 700.000 ordenadores Apple infectados (sistemas Mac OS X). El incidente de seguridad puso punto final a la percepción de que la plataforma Mac OS X era invulnerable. Además, los equipos Mac OS X también se han convertido en víctimas de ataques dirigidos. La razón principal la encontramos en que los productos de Apple son muy populares entre políticos y empresarios influyentes, y la información almacenada en los dispositivos resulta de gran interés para los delincuentes cibernéticos. En total, los expertos de Kaspersky Lab crearon un 30 % más de firmas para detectar troyanos para Mac en 2012 que en 2011.

En 2012, los productos de Kaspersky Lab bloquearon una media de más de 4 millones de ataques diarios basados en vulnerabilidades del navegador. La técnica más utilizada para atacar a los usuarios online son las vulnerabilidades en los programas o aplicaciones. En más del 50 % de los ataques se usaron brechas en *Java*. Diferentes versiones de esta máquina virtual, según los datos de Oracle, están instaladas en más de 1.000.000.000 de ordenadores.

En segundo lugar están los ataques mediante Adobe Reader, que constituyeron la cuarta parte de los ataques neutralizados. Poco a poco va bajando la popularidad de los *exploits* para Adobe Reader debido a que el mecanismo usado para detectarlos es bastante sencillo y a las actualizaciones automáticas introducidas en las últimas versiones de Reader.

No cabe duda de que el *malware* crece a un ritmo vertiginoso. Si en diciembre de 2012 se han detectado 200.000 muestras al día, en 2013 ningún sistema operativo en ningún dispositivo con conexión a *Internet* sin la correcta protección estará a salvo.

Ref.: http://www.elcandelero.es/tecnologico/blog/comment.php?type=trackback&entry_id=10079

A.3. Tipos de *malware* y otras amenazas

Gusanos

En términos informáticos, los gusanos son en realidad un sub-conjunto de malware. Su

principal diferencia con los virus radica en que no necesitan de un archivo anfitrión para seguir vivos. Los gusanos pueden reproducirse utilizando diferentes medios de comunicación como las redes locales o el correo electrónico. El archivo malicioso puede, por ejemplo, copiarse de una carpeta a otra o enviarse a toda la lista de contactos del correo electrónico.

En general, los gusanos son mucho más fáciles de eliminar de un sistema que los virus, dado que no infectan archivos. Los gusanos muy a menudo se añaden al inicio del sistema o modifican las claves de registro, para asegurarse que serán cargados cada vez que el mismo se inicie.

Gusanos de correo electrónico

Los gusanos que se envían por correo electrónico son los más antiguos y los más populares también hasta el momento. Las técnicas utilizadas son similares a las explicadas en el ítem anterior: mensajes atractivos utilizando ingeniería social, envío automático a la libreta de contactos si el usuario está infectado y archivos adjuntos por lo general ejecutables. Una de las características relevantes es la de suplantar la identidad de un usuario con fines maliciosos, conocida como spoofing. De esta forma, el correo llega a través de un remitente conocido por el usuario, solo que este no ha enviado el correo de forma voluntaria. Este detalle puede hacer que el usuario confíe en el correo solo por los datos de quién lo envía.

Gusanos de *P2P*

Los gusanos también utilizan las redes *P2P* como medio de transmisión para infectar ordenadores. En este tipo de redes (como el Emule o el Kazaa) utilizan nombres atractivos, teniendo en cuenta cuáles son las palabras más buscadas en estos servicios. Entre los disfraces más comunes, aparentan ser generalmente *cracks* de programas, fotos o videos de mujeres famosas o películas taquilleras.

Gusanos de Web

También es posible descargar un gusano a través de páginas web. Las técnicas utilizadas son similares al resto de los tipos de gusanos, utilizando nombres atractivos y creando páginas web falsas. Estos gusanos se combinan muchas veces con técnicas de *phishing* para lograr que la víctima ingrese a la página web y descargue el archivo malicioso.

Otra metodología de propagación web, utilizada por los gusanos, es a través de vulnerabilidades en las aplicaciones utilizadas. A través de la ejecución de scripts (porciones de código), es posible generar una descarga y ejecución del código malicioso, sin necesidad de intervención del usuario. Esta técnica se denomina *Drive-By-Download*, siendo el único paso que realiza el usuario para infectarse, el ingreso a una URL con el código malicioso. Estos scripts, se encuentran por lo general ofuscados, de forma tal de evitar su detección.

Gusanos de mensajería instantánea

Estos gusanos aprovechan el envío de archivos en el cliente de mensajería instantánea. Por lo general, solo afectan a los más populares: el MSN y el Windows Messenger. De esta forma, el usuario recibe la opción para transferir un archivo que en realidad es *malware*. Por lo general, estos archivos utilizan nombres atractivos para asegurarse que más usuarios lo descargarán, combinando el *malware* con técnicas de ingeniería social. Por lo general,

un PC que esté infectado con este tipo de *malware*, al iniciar sesión en Messenger, automáticamente el código malicioso envía el archivo (la invitación para descargar el archivo específicamente) a todos los contactos que tenga el usuario conectados. Otra característica es que comúnmente estos archivos son ejecutables (extensión .exe o .bat) a pesar de que en el texto de envío figuran como fotos o videos.

Esta nueva manera de pensar, de reconocimiento a negocio, ha traído consigo una gran variedad de códigos maliciosos que podemos clasificar de la siguiente manera:

Virus: "Programas informáticos o secuencias de comandos que intentan propagarse sin el consentimiento y conocimiento del usuario" y que realizan alguna acción maliciosa. Entre sus principales características podemos identificar las siguientes:

- Se presentan como archivos ejecutables, o han adherido su código malicioso a imágenes, hojas de cálculo o documentos.
- No pueden reproducirse por sí mismos, es decir para infectar otras computadoras es necesario que el usuario intervenga.
- Llevan a cabo una actividad maliciosa.

Caballo de troya (troyano): "Programa de computadora que aparenta tener una función útil, pero que contiene código posiblemente malicioso para evadir mecanismos de seguridad, a veces explotando accesos legítimos en un sistema."

***Bot*:** "Programa o script que realiza funciones que de otra manera habría que hacer manualmente. También se refiere a una computadora que ha sido comprometida y que ejecuta las instrucciones que el intruso ordena."

***Spyware*:** "También conocido como programa espía y comúnmente se refiere a aplicaciones que recopilan información sobre una persona u organización, las cuales se instalan y se ejecutan sin el conocimiento del usuario."

***Adware*:** Son programas que se instalan en el equipo con o sin intervención del usuario, su objetivo principal es descargar publicidad a la computadora infectada.

Dialers: Programas que utilizan el modem para realizar llamadas a servicios telefónicos con alto costo.

Puertas traseras: Son programas que tienen por objetivo hacer alguna modificación que permita a un tercero tener acceso total al equipo, a la red y/o a la información.

Hasta el momento se ha mencionado que estos programas realizan alguna actividad maligna en el equipo infectado, pero ¿qué tipo de actividad?

Entre las acciones más representativas del *malware* encontramos las siguientes:

- **Robo de información:** Entre la información que puede buscar un intruso a través de un código maliciosos encontramos: información relacionada con juegos, datos personales (teléfonos, direcciones, nombres, etcétera), información de inicio de sesión (usuarios y contraseñas) y también información relacionada con la actividad que realizamos en nuestra computadora, incluyendo hábitos de navegación y programas comúnmente utilizados.
- **Envío de correo no deseado (*spam*):** Algunos programas maliciosos se encargan de utilizar nuestra computadora y conexión a Internet para enviar correos publicitarios o con contenido malicioso a múltiples usuarios en Internet.
- **Control remoto:** Esta acción permite a un usuario malicioso tomar control de nuestro equipo, esto le permitiría utilizar nuestros recursos para almacenar más *malware* o para instalar programas o eliminar datos; aunque también podría utilizarse el equipo para llevar a cabo ataques a otros equipos de *Internet*.
- **Ataques de ingeniería social:** Existe una nueva tendencia de fabricar *malware* que tiene por objetivo intimidar, espantar o molestar a los usuarios para que compren ciertos productos (Roberts, 2008). Por ejemplo, existe código malicioso que se hace pasar por un antivirus y alerta a los usuarios de que el equipo está supuestamente infectado y que la única manera de eliminar la infección es adquiriendo un software promocionado por el *malware* (Garnham, 2009).

Estos problemas ocasionados por los códigos maliciosos pueden ser mitigados si como usuarios de los sistemas establecemos mecanismos para prevenir o erradicar una infección. Para ello es indispensable que identifiquemos si nuestro equipo ha sido infectado; anteriormente se ha mencionado que los desarrolladores de *malware* han mejorado sus técnicas para prevenir que los usuarios localicen su presencia, a pesar de ello aún existen características que nos permiten identificar si nuestra computadora está infectada:

- **Disminución del rendimiento del equipo:** Cuando un código malicioso se ejecuta utiliza recursos de memoria y procesamiento por lo que podemos identificar la presencia de *malware* si nuestro equipo se vuelve más "lento" sin razón aparente.
- **Problemas en la red:** Debido a que algunos códigos maliciosos hacen uso de la conexión a red, podemos detectar su presencia si se presentan fallas en la red (no es posible conectarse a sitios, no se pueden compartir archivos, etcétera) o simplemente si nuestras transferencias tardan más de lo esperado.
- **Aparición inesperada de publicidad.**
- **Pérdida inesperada de información.**
- **Aparición de nuevos archivos no creados por el usuario:** Algunos códigos maliciosos crean archivos en el sistema por lo que la disminución repentina de espacio en disco, así como la aparición de archivos en el sistema que no hayan sido creados por el usuario puede ser un síntoma de infección.

Algunas variantes de códigos maliciosos son capaces de desactivar la protección antivirus y de cualquier otro *software* de seguridad de nuestro equipo como *firewall*, *antispyware*,

etcétera.

Una vez identificada la infección por *malware* en nuestro equipo es necesario establecer mecanismos que nos permitan erradicarla:

- **Realizar un escaneo con un antivirus y antispyware actualizados:** Un antivirus y un antispyware actualizados podrían localizar la infección y erradicarla por completo del equipo. El escaneo se puede realizar mediante el antivirus instalado en el equipo o utilizando el servicio de un antivirus en línea.
- **Utilizar programas de eliminación automática:** En ocasiones podemos encontrar en *Internet* herramientas automatizadas que permiten eliminar códigos maliciosos, la desventaja es que solamente eliminan una variedad o un código malicioso muy específico.
- **Eliminar manualmente:** Si investigamos un poco acerca del código malicioso que afecta al sistema podemos encontrar cuales son las acciones que realiza y cómo se podría eliminar manualmente, sin embargo estos procedimientos se recomiendan solamente a usuarios experimentados.
- **Reinstalar el sistema operativo:** Este método sólo debe ser utilizado cuando no es posible eliminar al *malware* por ninguna de las recomendaciones anteriores, sin olvidar que se debe respaldar la información antes de llevar a cabo ésta acción.

Sin embargo no es suficiente tomar acciones reactivas ante una infección por *malware*, pues actuar solamente cuando el *malware* ha tenido algún efecto podría traer consigo un daño a las computadoras o a la información que pudiera ser irreversible, por lo que es necesario que también establezcamos acciones preventivas que nos ayuden a disminuir las probabilidades de que códigos maliciosos nos afecten:

- **Instalar y actualizar un software antivirus:** Un *software* antivirus no evita la infección por parte de un virus de Internet, pero si ayuda a la detección de éste tipo de código malicioso. Es necesario instalar, administrar y actualizar un software antivirus de forma correcta, pero no debe ser el único software de seguridad en el equipo" (Fuentes, 2006).
- **No abrir archivos adjuntos contenidos en correos electrónicos de procedencia extraña:** Una gran diversidad de virus en el Internet se propagan a través del correo electrónico, adjuntando a estos un archivo infectado con lo cual se puedan seguir propagando (Fuentes, 2006), por lo que sólo deben de ser abiertos aquellos documentos adjuntos que provengan de una fuente confiable y siempre que haya sido analizado por un antivirus antes de abrirlo.
- **Analizar los archivos con un antivirus antes de abrirlos:** Es recomendable solicitar al antivirus que lleve a cabo un análisis antes de que abramos un archivo, en especial debemos realizar esta acción cuando son archivos que abrimos por primera vez y/o que provienen de otro equipo.

- **Analizar medios extraíbles como:** disquetes, memorias usb, cd's, etcétera. Cuando insertamos medios extraíbles en otros equipos pueden contagiarse de algún tipo de código malicioso, por lo que antes de abrir el dispositivo y los archivos que contiene debemos analizarlo con un antivirus en búsqueda de *malware*.
- **Actualizar el sistema constantemente:** Algunos virus pueden tomar ventaja de algunas vulnerabilidades no actualizadas en el sistema, por lo que es indispensable instalar las actualizaciones de seguridad más recientes. Esto permitirá estar protegido contra posibles ataques de distintos virus que traten de tomar ventaja de una vulnerabilidad no actualizada." (Fuentes, 2006)

Ref.: http://revista.seguridad.unam.mx/numero-01/c%C3%B3digos-maliciosos

A.4. Historia de los NIDS

Las primeras investigaciones sobre detección de intrusos comienzan en 1980 en un trabajo de consultoría realizado para el gobierno norteamericano por James P. Anderson [3], quien trató de mejorar la complejidad de la auditoría y la habilidad para la vigilancia de sistemas informáticos. Es el primero que introduce el término "amenaza" en la seguridad informática, y lo define como la potencial posibilidad de un intento deliberado de acceso a información, manipulación de la misma, o hacer que un sistema sea inutilizable. Anderson presentó la idea de que el comportamiento normal de un usuario podrá caracterizarse mediante el análisis de su actividad en los registros de auditoría. De ese modo, los intentos de abusos podrán descubrirse detectando actividades anómalas que se desviaran significativamente de ese comportamiento normal.

En 1988, en los laboratorios Lawrence Livermore de University of California en Davis, se realiza el proyecto Haystack para las fuerzas aéreas de EE.UU. Haystack era el primer IDS que analizaba los datos de auditoría y los comparaba con patrones de ataque predefinidos. De este modo nacía el primer sistema de detección de usos indebidos basado en firmas, el tipo de IDS más extendido en el mercado actual. En 1990, surgen los primeros proyectos de IDS basados en red.

Todd Heberlein introduce tal idea y desarrolla NSM [54] (*Network Security Monitor*) en University of California at Davis. En esa misma fecha, en Los Alamos National Laboratory de EEUU realizan un prototipo de un sistema experto que monitoriza la actividad de red. Su nombre es NADIR (*Network Anomaly Detector and Intrusion Reporter*) [55]. A partir de este momento, comienzan una gran variedad de proyectos de investigación que hacen uso de diferentes técnicas y algoritmos para el análisis del comportamiento de un sistema informático.

En el año 2007 podemos encontrar ya más de 300 IDS comerciales cada uno con sus puntos fuertes y flaquezas, pudiendo afirmar que nos encontramos ante un conjunto de tecnologías ya maduro y plenamente aceptado por la comunidad como un elemento necesario para la defensa en profundidad de un sistema. Sin embargo ya en 2001 habían aparecido nuevos conceptos y herramientas de seguridad tales como los IPS (Sistema de Prevención de Intrusiones) o *HoneyPots* (Panales de miel) los cuales en la actualidad han desplazado parcialmente a los IDS como herramienta de detección.

Taxonomía

Debido a la variedad de alternativas, las cuales presentan como es lógico sus particulares bondades y flaquezas, se hace necesario escoger una serie de ejes (características definitorias) en torno a los que agrupar la taxonomía con el fin de poder establecer una clasificación correcta para cualquier IDS existente.

A.4.1. Clasificación por estrategia de respuesta

Respuesta pasiva:

Típicamente los IDS presentan este tipo de respuesta siendo la respuesta activa una característica más propia de un IPS [56] (*Intrusion Prevention Systems*). Este tipo de respuesta consiste en enviar alertas al administrador del sistema cuando un evento supera el umbral de confianza especificado para el detector. El principal problema de este tipo de IDS es que puede sobrepasar a los operadores humanos con alertas procedentes de un mismo ataque, o simplemente con la generación de falsos positivos (En un entorno típico de trabajo de un IDS pueden llegar hasta el millón diarios).

Respuesta activa:

Es de fácil comprensión que la respuesta activa presenta ventajas fundamentales en la protección de cualquier sistema, puesto que una pronta respuesta ante un ataque significa que la integridad del sistema estaría mejor protegida que se ha de esperar a que el operador humano analice las alertas generadas por el IDS y efectúe las acciones correspondientes.

La naturaleza de las acciones que pueden ser llevadas a cabo por parte de un IDS en respuesta a un positivo en el análisis depende esencialmente de si se trata de un NIDS (*Host-based Intrusion Prevention System*) o de un HIDS [58].

Conclusión

En un IDS con respuesta pasiva los falsos positivos pueden llegar a ser muy molestos puesto que pueden llegar a sobrepasar la capacidad de los operadores humanos para discriminar entre una falsa alarma y una real, sin embargo cuando se emplea la respuesta activa podemos llegar a incurrir en denegaciones de servicio arbitrarias a usuarios legítimos con lo que el rango de circunstancias en las que se emplea una respuesta activa ha de ser limitado a un pequeño conjunto de actividades maliciosas bien detenidas como por ejemplo la exploración de puertos.

A.4.2. Clasificación por entorno monitorizado

Host IDS (HIDS)

Este tipo de IDS fue el primero en ser desarrollado y desplegado. Típicamente están instalados sobre el servidor principal, elementos clave o expuestos del sistema. Los HIDS operan monitorizando cambios en los registros de auditoría del host tales como: procesos del sistema, uso de CPU, acceso a ficheros, cuentas de usuario, políticas de auditoría,

registro de eventos. Si los cambios en alguno de estos indicadores exceden el umbral de confianza del HIDS o hace peligrar la integridad del sistema protegido el HIDS tomaría una acción en respuesta. La principal ventaja de estos detectores es su relativo bajo coste computacional puesto que monitorizan información de alto nivel, sin embargo esto también constituye su principal inconveniente puesto que solo pueden lanzar una alerta una vez que el daño al sistema ya está hecho. Ejemplos de HIDS comerciales son: Symantec Host IDS, ISS BlackICE PC16, TCPWrappers, Enterasys Dragon Host Sensor.

Network IDS (NIDS)

NIDS son las siglas de *Network Intrusion detection System*, estos IDS analizan eventos del tráfico provenientes del tráfico de red para clasificarlos adecuadamente en pertenecientes o no a un ataque. Existen diversas formas de clasificar los NIDS: por objeto de análisis, por situación de los componentes del IDS, por tipo de análisis efectuado y por último por la técnica de análisis empleada. Para el propósito de este documento tan solo entraremos en detalle sobre el objeto de análisis.

Un NIDS monitoriza los eventos de red, esto significa que el objeto de análisis son los paquetes y tramas de los diversos protocolos de la misma. Estos eventos de red se dividen en dos partes: cabecera y contenedor, en consecuencia se han desarrollado NIDS's que abordan el problema de la detección de intrusiones mediante el análisis de una de estas componentes dado que cada una de estas aproximaciones trae ventajas en la detección de distintos tipos de ataques.

A.4.3. Clasificación por estrategia de detección

La estrategia de análisis se refiere a la aproximación elegida para distinguir el uso legítimo y el uso anómalo del sistema. La mayor parte de autores coinciden al identificar dos categorías principales y una derivada de ellas: IDS basados en detección de firmas, IDS basados en detección de anomalías y por último IDS híbridos. En pocas palabras la detección de firmas construye modelos sobre ataques conocidos llamados firmas, cualquier coincidencia de una secuencia de eventos con un patrón almacenado disparará una alarma, por otro lado en la detección de anomalías intenta modelar el comportamiento usual del entorno monitorizado y se asume que cualquier evento anormal es en sí mismo sospechoso.

IDS basados en detección de firmas

Los IDS que basan su estrategia de detección de firmas monitorizan en el uso indebido de los recursos y realizan su trabajo mediante la clasificación de los eventos de acuerdo a una base de conocimiento que guarda firmas de ataques conocidos. Si una secuencia de eventos de sistema coincide con una firma almacenada en la base de datos se dispararía una alarma.

Originalmente las firmas eran construidas a mano usando en conocimiento de expertos, sin embargo hoy en día este proceso se ha agilizado gracias a la utilización de técnicas de minería de datos sobre registros de intrusiones conocidas [59].

Estos sistemas funcionan de manera muy precisa siempre que se trate de patrones de ataques conocidos, sin embargo resulta bastante fácil engañarlos con la introducción de

variaciones equivalentes a NOPS [60] en los eventos de los ataques, y son básicamente inútiles frente a ataques de nueva aparición.

IDS basados en detección de anomalías

Por el contrario los orientados a la detección de anomalías [54], se basan en el modelado de la actividad normal del sistema y en la detección de desviaciones sobre la misma. Han recibido una gran atención en los últimos años debido a que tienen la interesante propiedad de poder hacer frente a amenazas de nueva aparición. Sin embargo presentan varios problemas relacionados tanto con su proceso de construcción como con el elevado *ratio* de falsos positivos que suelen arrojar.

El primer obstáculo está en establecer cuál es el uso normal del sistema. Esto no es trivial, y todo IDS basado en anomalías ha de, o bien construir un conjunto de datos limpio, u obtenerlo de algún modo. Decimos limpio puesto que aunque recientemente estos IDS pueden tolerar cierto ruido o presencia de ataques en el conjunto de entrenamiento. Si se entrenan con tráfico real este puede contener ataques no detectados y el detector puede aprender esos ataques como parte del comportamiento normal del sistema, y por lo tanto serán incapaces de detectarlo una vez desplegados.

Por otro lado los conjuntos construidos artificialmente pueden ser demasiado restrictivos y no reflejar en absoluto el comportamiento del sistema, haciendo de esta manera que el número de falsos positivos se dispare. El elevado *ratio* de falsos positivos, es el precio a pagar por tener un detector de anomalías, puesto que cualquier nueva aplicación o incluso evolución en el uso del mismo por parte de los usuarios puede ser identificado como una amenaza y disparar una alerta.

Cabe reseñar que ante amenazas conocidas, los IDS basados en uso indebido presentan un mejor rendimiento y un coste menor. Sin embargo hay que ponderar la detección precisa de amenazas conocidas frente a la detección de amenazas de nueva aparición.

IDS híbridos

Existe también la posibilidad de combinar ambas estrategias dando lugar a detectores híbridos [61], los cuales intentan combinar los puntos fuertes de las estrategias antes descritas puesto que los detectores de anomalías no constituyen una respuesta completa al problema de la detección de intrusiones y usualmente han de ser combinados con estrategias de detección de uso indebido.

Las razones para esto son: alto índice de aciertos y bajo número de falsos positivos ante vulnerabilidades conocidas de los IDS basados en uso indebido, y la protección frente a nuevos ataques de los IDS basados en la de detección de anomalías. Un ejemplo de este tipo de IDS es Emerald [62], que realiza ambos tipos de análisis. Como otra característica interesante de Emerald es que hace uso de la plataforma CIDF para permitir la comunicación entre sus elementos de análisis y la adición de módulos procedentes de terceras partes permitiendo así configurar el sistema de detección para responder a las necesidades de la organización que lo despliegue.

A.4.4. Clasificación por técnicas de análisis

También se puede realizar la clasificación orientándonos por la técnica elegida para implementar el clasificador de eventos en los módulos de análisis del IDS. Clasificamos los métodos en 5 grupos: basados en conocimiento, basados en análisis de firmas, análisis de transición de estados, basados en métodos estadísticos y por último aquellos basados en aprendizaje automático.

Sistemas basados en el conocimiento: Basados en reglas de tipo *if-then-else*, si algún evento captado del entorno satisface todas las precondiciones de una regla, esta se disparara. Ofrecen la ventaja de separar la lógica de control del dominio del problema, sin embargo no son una buena alternativa para analizar secuencias de eventos. Ejemplo de este tipo de IDS es *P-Best*, aunque este sea más bien un intérprete utilizado por otros IDS.

Análisis de firmas: Observa la ocurrencia de cadenas especiales en los eventos monitorizados por el IDS y los compara con los almacenados en una base de datos con firmas de ataques. El principal problema con este enfoque es la necesidad de incorporar una nueva firma a la base de datos por cada nuevo ataque conocido. Quizás el ejemplo más conocido de estos IDS sea Snort [9] debido a su licencia libre y capacidad de desarrollo de preprocesadores y reglas por parte de la comunidad.

Análisis de transición de estados: Se trata de autómatas finitos que a través de transiciones basadas en los estados de seguridad del sistema intentan modelar el proceso de una intrusión. Cuando el autómata alcanza un estado considerado de peligro lanza una alerta. Representantes de estos sistemas que alcanzaron gran difusión son NetSTAT y USTAT (basados en red y local respectivamente).

Sistemas basados en métodos estadísticos: Se basan en modelizar el comportamiento de los usuarios en base a métricas extraídas (estrategia de detección de anomalías) de los registros auditorías del sistema. Una característica clásica de estos sistemas es el establecimiento de perfiles para los distintos usuarios del sistema. El proceso de comparación consiste en cotejar un vector de variables que han sido recogidas del sistema, con otro almacenado que guarda el comportamiento normal de ese usuario. Si la desviación supera un valor predefinido, se lanzaría una alarma. Es conveniente actualizar los perfiles cada cierto tiempo, usando para ello los registros de actividad, del usuario, más recientes, para de esa manera moderar el número de falsos positivos de la herramienta. Sin embargo esta es una fuente de vulnerabilidad de estos sistemas, puesto que un atacante puede desentrenar la herramienta para que permita una intrusión. Un ejemplo de estos sistemas es el ISA-IDS. [53]

Sistemas basados en aprendizaje automático: Se trata de sistemas que realizan uso de técnicas que no necesitan intervención humana para extraer los modelos que representan bien ataques, o bien el uso legítimo del sistema. La extracción de los modelos se realiza durante la fase de entrenamiento. Ejemplos de conjuntos representativos disponibles en la red son los elaborados por la DARPA en 1998 y 1999 así como el proveniente de la KDD Cup de 1999 organizada por la ACM. También pueden ser entrenados con datos recogidos de redes reales. Con respecto al mecanismo de entrenamiento se puede hacer la división entre entrenamiento supervisado y no supervisado.

El aprendizaje supervisado hace uso de conjuntos etiquetados, teniendo como desventaja la escasez de dichos conjuntos de datos sobre sistemas reales, a proteger por la

herramienta, ya que pocas organizaciones guardan conjuntos de datos exhaustivamente etiquetados. Por el contrario el aprendizaje no supervisado no precisa de conjuntos etiquetados. Muchas técnicas de aprendizaje automático pueden hacer uso de ambos métodos en distintas etapas del entrenamiento. Por ejemplo [35] hace uso de ambas. Estos sistemas han recibido gran atención tanto en el desarrollo de IDS orientados a detección de anomalías como en el de uso indebido. Ya que permiten la extracción de conocimiento sobre el dominio del problema, minimizando la intervención humana y el tiempo que se tarda en preparar las defensas frente a nuevas amenazas.

A.5. Introducción a la arquitectura ARM

Como se ha dicho anteriormente, la microarquitectura RISC tiene a ARM como su principal representante en el mercado actual. Es cierto que existen otras arquitecturas con el mismo enfoque, como lo son los IBM PowerPC o las MIPS32 y MIPS64 de MIPS Technologies, pero su uso es mucho menor que en el caso de nuestra principal protagonista. Sus siglas provienen de Advanced RISC Machine.

La simplicidad de los procesadores ARM es lo que le ha convertido a esta arquitectura en el líder para uso en ordenadores y dispositivos de bolsillo tales como tablets y móviles. Esta microarquitectura no sólo triunfa en el campo de los dispositivos móviles, también se ha hecho con el mercado como por ejemplo un disco duro tradicional o un SSD.

Todos estos usos de procesadores ARM existen porque es una arquitectura de muy bajo consumo energético, al menos en comparación con las alternativas existentes en el mercado. Además, la fabricación de estos chips es bastante barata. Por el contrario tiene que las implementaciones software específicas sí que suelen ser más costosas que en otras arquitecturas como la x86 más habitual de nuestros ordenadores. Desde la primera versión comercial allá por mediados de los 80, ARM ha ido evolucionando con el paso del tiempo y por supuesto se han introducido nuevas instrucciones con cada versión de la arquitectura, no sólo para mejorar el rendimiento si no también la seguridad.

Como se ha dicho muchas veces, la característica principal de ARM es que consume poco y que es barata. Esto es debido a que ARM utiliza relativamente pocos transistores, o al menos muchos menos que los procesadores de otras arquitecturas.

Desde el nacimiento de Advanced RISC Machines Ltd hace poco más de veinte años, ARM ha crecido de forma muy notable tanto económica como tecnológicamente. Ya se ha dicho que esta arquitectura se ha impuesto en el mercado de los dispositivos móviles, y no solo eso, sino que ahora con las mejores que han ido introduciendo a lo largo de los años, ARM se está imponiendo incluso en los elementos a priori más tradicionales para ofrecer nuevas funciones a los usuarios: por ejemplo los nuevos aspiradores robot, televisiones, o incluso los juguetes más tecnológicamente avanzados.

ARM es una microarquitectura tan fiable y de tan buenas prestaciones que se está convirtiendo en una opción real para la sustitución de las arquitecturas actuales en centro de datos. Ya que el bajo consumo de esta microarquitectura y el bajo coste, abarataría mucho dinero a estos centros, debido al consumo energético de estos. Ya que si los procesadores consumen menos, se calientan menos, se consume menos en ventilación, etc.

A día de hoy podemos asegurar que el futuro de ARM de lo más prometedor. Ganan rendimiento en cada nueva generación de procesadores, incluso reduciendo el consumo energético, y por si fuera poco la innovación tiene muy en cuenta a esta arquitectura. Como por ejemplo la domótica (conjunto de sistemas que permiten automatizar una vivienda), los coches están empezando a utilizarlos, además de nuevos elementos electrónicos que aparecerán en los próximos años de la misma forma que hace no mucho aterrizaron los teléfonos, tablets o libros electrónicos.

Y no solo eso: los usuarios están modificando su concepto de ordenador personal. Estamos ante eso que muchos denominan la era post-PC en la que ARM puede ser vital para conseguir nuevos tipos de dispositivos portátiles y con autonomías hoy impensables, pero que serán razonables y tecnológicamente viables dentro de unos pocos años.