UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIAS E INGENIERÍA DE LA COMPUTACIÓN

INSTITUTO DE INVESTIGACIONES EN MATEMÁTICAS APLICADAS Y EN SISTEMAS

INTELIGENCIA ARTIFICIAL

Detección y Clasificación de Malware: Un Enfoque Basado en Algoritmos de Aprendizaje Supervisado

T E S I S

QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN CIENCIAS E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:
A L E X I V Á N V A L E N C I A V A L E N C I A

Director de Tesis:
Dr. Christopher Rhodes Stephens
Centro de Ciencias de la Complejidad

Codirectora de Tesis:
Dra. María del Pilar Ángeles
Posgrado de Facultad de Ingeniería

Ciudad Universitaria, Cd. Mx. **Noviembre 2016**

ÍNDICE

Página

DEDICATORIAS

A aquellos hombres cuyos pasos han sido un camino en mi andar,
cuyas miradas me acompañan como aves revoloteando en mi historia,
por su comprensión, su apoyo, sus consejos y por su forma de amar
a mi papá Víctor y mi abuelo Alejandro, que Dios los tenga en su gloria.

A las estrellas que le dan nombre a la noche, mis cefeidas:
A mi mamá y mis hermanas, las mujeres que siempre me han apoyado,
con su sensibilidad, su tiempo, su paciencia, y sus atentos cuidados.

A la mujer que se ha convertido en una compañera imprescindible,
por la diversidad de emociones y de sueños compartidos,
porque: “el Sol es una promesa y mi vida es contigo”.

A mi familia, que como un gran nicho fraterno siempre ha estado presente
no sólo durante la tempestad, sino también con toda su disposición y cariño.

A mi mejor maestro en la vida, Mat. Luis Ramírez Flores Q.E.P.D,
por su pasión y pedagogía íntegra en matemáticas y humanismo.

AGRADECIMIENTOS

Gracias a Dios

A mis Amigo(a)s del IIMAS, por su amistad sincera, su apoyo profesional, y sobre todo por su compañía.

A la Dra. María del Pilar y a la Dra. Sofía Galicia Haro, por apoyar desde el comienzo este trabajo con su tiempo y dedicación, además de sus enseñanzas en el arte y disciplina de la investigación.

Al Dr. Christopher Stephens Stevens, por darle lugar y continuidad al proyecto de investigación además de brindarle apertura a su concreción en la industria.

A la Universidad Nacional Autónoma de México, por darme un segundo hogar y la oportunidad de desarrollarme integra y profesionalmente sin ninguna restricción salvo los límites propios.

Al UNAM CERT, por su capacitación, y por haber generado el apasionado gusto por la investigación en seguridad informática.

A mis profesores del posgrado y sinodales, por su gran pasión hacia la enseñanza y a la trascendencia del Posgrado en Ciencias e Ingeniería de la Computación.

A las secretarias y personal del IIMAS, por siempre mantener una actitud servicial y atenta a los alumnos del posgrado.

Al CONACYT, por el gran apoyo a realizar mis estudios de posgrado.

DETECCIÓN Y CLASIFICACIÓN DE MALWARE: UN ENFOQUE BASADO EN ALGORITMOS DE APRENDIZAJE SUPERVISADO

RESUMEN

Ante el crecimiento exponencial de las variantes de malware subyacente de la cantidad de tecnologías vulnerables a estas, el malware en sí mismo representa un problema complejo, el cual impacta considerablemente a aquellas organizaciones cuyo principal activo sea la información, en ese sentido las soluciones de seguridad deben considerar en primera instancia el tiempo de su detección, el cual depende esencialmente del tiempo de su análisis. En esta investigación se propone buscar patrones de comportamiento de malware en el sistema operativo Windows, empleando algoritmos de aprendizaje supervisado en dos conjuntos de variables. Para lo cual se llevaron a cabo los siguientes experimentos:

En el primer experimento se estableció como objetivo: Obtener los valores de los contadores más relacionados con la presencia de malware, al inducir actividad maliciosa y utilizando un clasificador Bayesiano. Para ello se utilizaron dos variantes de dos familias de Gusanos de red: Net-Worm.Win32.Kolab y Net-Worm.Win32.Kido además de Whiteware (software benigno que sirvió para diferenciar su comportamiento del malware) contenido en el directorio de System32. Las muestras fueron ejecutadas con la SandBox Cuckoo en un ambiente virtual con VirtualBox, los contadores de rendimiento fueron obtenidos por medio de un script en Powershell que obtenía las muestras cada segundo durante tres minutos por muestra. Después de obtener 494 reportes de tres minutos, resultado de: Net-Worm.Win32.Kolab.II 69, Net-Worm.Win32.Kolab.se 72, Net-Worm.Win32.Kido 58 Net-Worm.Win32.Kido.ih 66 y Whiteware 230. Posteriormente se obtuvo su promedio en los tres minutos de cada contador del reporte a través de un script en Bash, como parte del pre procesamiento de los datos, debido a que estos presentaban un comportamiento continuo se empleó la técnica de Coarse Grain para poder hacerlos discretos. Finalmente, con otro script en Bash se generó el modelo de Score utilizando un 70% de los datos como entrenamiento y su complemento fue utilizado para probar el modelo, obteniendo un rendimiento de 90%.

En el segundo experimento se utilizó la Sandbox Cuckoo para llevar a cabo el análisis de seis tipos de malware: Troyanos, Gusanos, Virus, Troyanos Espía, Puertas Traseras y Rootkits, además de un conjunto de Whiteware con el mismo número de muestras para cada tipo. Utilizando como fuente de información los n-gramas resultantes de las llamadas al sistema del tipo Win API, por los procesos ejecutados durante la ejecución de cada muestra y variando el tamaño de los n-gramas de uno a diez, se obtuvieron 10 diferentes bases de conocimiento correspondientes en cada modelo, a la relación de presencia de cada n-grama en cada muestra de malware, resultando diez tablas binarias con dicha información. Esencialmente se usaron las bases de conocimiento para dos experimentos:

a) Clasificación de malware utilizando Máquinas de Soporte Vectorial (SVM) con kernel polinomial como algoritmo de aprendizaje supervisado y la variación del tamaño de n-gramas para saber cual tiene mayor exactitud acc. Dónde la mejor relación de exactitud promedio y su desviación estándar, fue generada con la base de conocimiento de 3gramas, con un valor de 75.5 y 1.26 correspondientemente.
b) Minería de datos con el clasificador Bayesiano sobre los n-gramas de Win API con malware y whiteware para conocer que n-gramas están más relacionados con procesos maliciosos. Siguiendo la metodología, se realizaron diez iteraciones con diferentes algoritmos de clasificación, y puede apreciarse que nuestra implementación de Naive Bayes obtuvo los mejores resultados de exactitud considerando todos los n-gramas y concretamente con el modelo de 3gramas obtuvo su mejor resultado con 99.31% y con una desviación estándar de 1.

DETECTION AND CLASIFICATION MALWARE: A FOCUS BASED ON SUPERVISED LEARNING ALGORITHMS

ABSTRACT

The exponential growth of variants of malware, as well as the amount of technologies vulnerable them, malware itself represents a complex problem that has a significant impact on those organizations have main asset is information, in that sense security solutions must consider in the first instance the time of their detection, which depends essentially on the time of their analysis. In this research, it is proposed to search for malware behavior patterns in the Windows operating system, using supervised learning algorithms in two sets of variables, for which the following experiments were carried out:

The first experiment had the following objective: Obtain the values of the counters most related with the presence of malware, by inducing malicious activity and using a Bayesian classifier. For this, two variants of two families of network worms are used: Net-Worm.Win32.Kolab and Net-Worm.Win32.Kido as well as Whiteware (benign software which serves to differentiate between the behaviour of malwares) contained in the System32 directory. The tests were executed with Sandbox Cuckoo in a virtual environment via VitrualBox, the performance counters were obtained by a Powershell script that gathered the data every second over three minutes for each test. After obtaining 494 reports of three minutes in duration, the counts were: Net-Worm.Win32.Kolab.II 69, Net-Worm.Win32.Kolab.se 72, Net-Worm.Win32.Kido 58 Net-Worm.Win32.Kido.ih 66 and Whiteware 230. The average in three minutes of each counter of the report was then obtained through a script in Bash. As part of the pre-processing of the data, all continuous data was converted to discrete data via the coarse graining technique. Finally, using a different script in Bash, a Score model was generated using 70% of the data for training the model and the remaining data was used to test the model, achieving 90% performance.

In the second experiment the Sandbox Cuckoo was used to carry out the analysis of six types of malware: Trojans, Worms, Viruses, Spy Trojans, Rear Doors and Rootkits, as well as a set of Whiteware with the same number of samples for each type. Using the n-grams resulting from the Win API system calls as an information source, for the processes executed during the running of each sample, while varying the size of the n-grams from one to ten, we obtained 10 different knowledge bases. They correspond to each model, where the percentage of presence of each n-gram in each sample of malware are detailed in ten binary tables. Essentially, the knowledge bases were used for two experiments:

a) Classification of malware using Support Vector Machines (SVM) with a polynomial kernel as a supervised learning algorithm and the variation of n-grams size to know which has the highest acc. Where the best mean accuracy ratio and its standard deviation was generated with the knowledge base of trigrams, with a value of 75.5 and 1.26 respectively.

b) Data mining with the Bayesian classifier on Win API n-grams with malware and whiteware, to determine which n-grams are most related to malicious processes. Following this, ten iterations were performed with different classification algorithms which show that our use of the naive Bayes classifier obtained the best accuracy results considering all the n-grams and specifically with the 3 n-grams model obtained its best result with 99.31% accuracy and a standard deviation of 1.

CAPÍTULO I

«La verdad no se aprende, se piensa
y se experimenta por uno mismo.»
Octavio Paz

Introducción

Al día de hoy las computadoras se han convertido en grandes herramientas de procesamiento, y en este sentido los códigos maliciosos han evolucionado tanto en el daño causado cómo en las características que permiten ocultarlos del análisis. Respecto a las estadísticas, la firma Symantec en su reporte de Amenazas de Seguridad en Internet de 2015 manifiesta que en 2014 fueron introducidas 317 millones de variantes de malware. Mantenerse al día con tal cantidad de variantes es desalentador para las organizaciones. [1] El método de análisis de malware es uno de los problemas clave en la técnica de detección de intrusos. En la literatura, los principales métodos de análisis de malware están basados en análisis de contenido estático y en el comportamiento dinámico. [2] En el análisis estático, las características se extraen del código binario de los programas y son usados para crear modelos que los describan. Los modelos se usan para distinguir entre malware y software legítimo (también conocido como whiteware). [3] Sin embargo el análisis estático falla en diferentes técnicas de ofuscación de código usado por los desarrolladores de códigos maliciosos y también en malcodes metamórficos y polimórficos. [4]

En las técnicas de ofuscación el código fuente y el código binario son transformados de tal manera que tanto el proceso de de-compilación sea más difícil, así como la lectura y análisis del mismo, cómo se verá en el Capítulo 2. Aunado a lo anterior el polimorfismo de malware consiste en cambiar la apariencia de éste a través de métodos de cifrado, de agregación de datos o eliminación de datos. [5] Debido a que el análisis dinámico de malware consiste en el estudio del mismo a través de su ejecución en un entorno controlado, su principal ventaja es que dicho análisis de comportamiento no puede ser ofuscado. [6]-[7] Sin embargo, hay algunas limitaciones en el análisis dinámico, ya que cada muestra de malware debe ejecutarse dentro de un entorno seguro por un tiempo específico para monitorear el comportamiento. El proceso de monitoreo consume tiempo y debe asegurarse que la ejecución del malware no infecta la plataforma. [8] Los entornos seguros discrepan sólo un poco de un entorno de ejecución real y el malware puede comportarse de diferente manera en los dos entornos, causando una bitácora inexacta del comportamiento del malware. [9] Además de que algunas acciones del software malicioso se activan o ejecutan bajo ciertas condiciones (la fecha y hora del sistema o alguna entrada particular proporcionada por el usuario) puede no detectarse por el entorno virtual seguro. [10]

El análisis dinámico es un complemento necesario al enfoque estático como una medida preventiva de ofuscación de código, consiste de: una colección de muestras de malware, la ejecución de dichas muestras, entornos de monitoreo y la determinación de un modelo de características de comportamiento y análisis de comportamiento (clustering, clasificación, reconocimiento, etc.). La investigación de recolección de malware, ejecución de malware y

métodos de monitoreo ha alcanzado un resultado maduro. Algunos sistemas de recolección y métodos se proponen basados en honeypot, como: Dionaea [11] y Kippo. [12] Los sistemas típicos tales como Anubis, [13]-[14] CWSandbox [15], Cuckoo SandBox [16] ejecutan el malware en un ambiente controlado y monitorean el comportamiento del mismo (nombrado método de Sandboxing), finalmente genera un reporte de comportamiento para cada muestra. El análisis profundo es necesario para revelar características del comportamiento y la detección de malware. Dos principales conceptos para el análisis automático de comportamiento se han propuesto: clustering y clasificación, los cuales se verán con más detalle en el Capítulo 3, e. [17]-[18]

En la presente investigación se proponen dos experimentos (detallados en los Capítulos 4 y 5), dónde se lleva a cabo la detección y clasificación de malware con algoritmos de aprendizaje supervisado como una alternativa al proceso que realizan los analistas de malcode en el sistema operativo Windows utilizando dos fuentes de datos distintas. Los algoritmos de aprendizaje supervisado corresponden al clasificador Bayesiano para el primer experimento dónde se utiliza como fuente los contadores de rendimiento de Windows y para el segundo experimento se utiliza el algoritmo de Máquinas de Soporte Vectorial con n-gramas de Win API Calls como datos de entrenamiento, y en este último experimento se utilizó nuevamente el clasificador Bayesiano para minar los n-gramas de Win API Calls más relacionados con la presencia de malware.

1.1 Estado del arte

En la industria existe software anti-virus, el cual está diseñado para proteger del malware al equipo de cómputo. Los antivirus detectan y bloquean los intentos de los atacantes al infectar el dispositivo, el problema estriba en que no pueden mantenerse al día con los atacantes debido al incremento de variantes diarias, y de esta manera no puedan detectar y proteger contra todas las amenazas. Por esta razón es posible que el equipo pueda ser infectado incluso con la última versión de anti-virus instalada.

Básicamente los programas de anti-virus funcionan con dos mecanismos de detección:

a) Detección por firma: Muchos anti-virus trabajan como el sistema inmune de los humanos al escanear el equipo en busca de las firmas (patrones) de patógenos digitales e infecciones. Esto se refiere a un diccionario de malware conocido, y si algo en un archivo hace match con un patrón en el diccionario, el antivirus intenta neutralizarlo. Como el sistema inmune, el contenido del diccionario necesita actualizarse, así como las vacunas de la influenza (siguiendo la analogía) para proveer protección contra nuevos ejemplares de malware. Uno de los principales problemas con este enfoque es que el equipo es vulnerable durante el retardo entre el tiempo en que el malware es identificado y el tiempo en que el diccionario es actualizado por las casas anti-virus.

b) Detección por comportamiento: En este enfoque, en lugar de intentar identificar malware conocido, el anti-virus monitorea el comportamiento de software instalado. Cuando un programa actúa de forma sospechosa, como intentar acceder a un archivo protegido o modificar otro programa, el anti-virus registra la actividad y es alertada al usuario. Esto provee de protección contra nuevos tipos de malware que no existían en ningún diccionario. El problema con este enfoque es que puede

generar un gran número de advertencias con falsos negativos, además de que sólo monitorea las rutas frecuentemente utilizadas por los atacantes. [19]

Por otro lado, en el contexto académico se han realizado investigaciones dónde se persigue la detección de malware teniendo o no conocimiento a priori del comportamiento del mismo al considerar diferentes fuentes de información.

En la investigación de Santos y Penya [20], extraen los n-gramas de cada muestra (149882 muestras de diferentes familias de malware y 4934 de whiteware) y son utilizados como datos de entrenamiento para el algoritmo de aprendizaje no supervisado: k nearest neighbor con lo cual mencionan que es posible clasificar malware que no es conocido, obteniendo un rendimiento máximo de 91.25% al emplear 4-gramas en su modelo.

Tang y Sethumadhavan [21] proponen un enfoque que no requiere de firmas de archivo para la detección de malware, en vez de ello analizan el comportamiento del microprocesador al ejecutar software legítimo (Adobe PDF Reader e Internet Explorer con una plataforma Windows x86) y de esta forma poder detectar alguna anomalía. Para ello muestran los contadores de rendimiento del microprocesador como datos de entrenamiento para el algoritmo de Máquinas de Soporte Vectorial con una función no lineal de base radial como kernel.

Rossow y Dietrich [22] se concretan en la actividad maliciosa de red en grandes periodos de tiempo y proveen de un compilado de comportamiento de red típico del malware con 100,000 muestras como datos de entrada para un entorno propio de análisis llamado Sandnet en el cual se analizan a fondo los protocolos DNS y HTTP los cuales son los más utilizados por los desarrolladores de malcode.

En el trabajo de Christopher Richardson [23], se utilizan algoritmos de aprendizaje automatizado para la detección del malware basado en su comportamiento. En un entorno de diez máquinas virtuales se simula el uso de la computadora de forma real para generar datos de entrenamiento con una mejor aproximación. Se generan dos conjuntos de datos de entrenamiento:

a) Induciendo tanto comportamiento normal como malicioso al infectar el equipo con virus
b) Capturando datos del equipo sin infectar para determinar que corresponde a una anomalía

Obteniendo los datos se genera el modelo con el clasificador Bayesiano después de pre procesar los datos con ventanas de Parzen y consiguiendo un rendimiento de 92% para el primer enfoque y de 62% en el segundo.

En la investigación realizada por Raymond J. [24], se estudia el problema de detección y clasificación de procesos maliciosos usando un análisis del registro de llamadas al sistema para inferir comportamiento malicioso. Al igual que en [23] se utiliza un entorno virtual pero con 14 máquinas virtuales con 55 mil muestras de malware que obtuvieron un mínimo de 1500 llamadas al sistema, por otro lado los datos de Whiteware fueron generados con 1000 días de 43 computadoras de igual manera sobre los procesos que cumplían con un mínimo de 1500 llamadas al sistema obteniendo un total de 4 millones de procesos benignos, considerando los siguientes algoritmos para la detección de procesos maliciosos: Detector basado en firmas, máquinas de soporte vectorial, regresión logística y la prueba de índice

de probabilidad logarítmica, en la siguiente tabla se muestran los diferentes enfoques con llamadas al sistema, siendo esta una de las fuentes de datos más recurridas por los investigadores de malware:

Estrategia	Datos de entrada	Algoritmos de Detección	Evaluación
Forrest et al.[25]	Secuencias de WinAPI calls	Firmas	Detección de Intrusos
Hofmeyr et al. [26]	Secuencias de WinAPI calls	Firmas	Detección de Intrusos
Warrender et al. [27]	Secuencias de WinAPI calls, Frecuencias de WinAPI calls	HMM, Rule learner	Detección de Intrusos
Liao and Vemuri [28]	Frecuencias de WinAPI calls	Nearest Neighbor	Detección de Intrusos
Kang et al. [29]	Frecuencias de WinAPI calls	SVM, LR, Clasificador Bayesiano,Árbol de desición, Rule Learner	Detección de Intrusos
Xin and Xu [30]	Secuencias de WinAPI calls	Modelos de Markov, Temporal Diference Learning	Detección de Intrusos
Burguera et al. [31]	Frecuencias de WinAPI calls	Clustering	3 Benignos,5 Maliciosos
Martignoni et al. [32]	Frecuencias de WinAPI calls	Graph matching	11 Benignos,7 Maliciosos
Tokhtabayev et al. [33]	Activiades	Firmas	210 Benignos,31 Maliciosos
Mehdi et al. [34]	Secuencias de WinAPI calls	Rule learner, SVM, Árbol de decisión	72 Benignos,72 Maliciosos
Kirda et al. [35]	Activiades	Firmas	18 Benignos,33 Maliciosos
Kolbitsch et al. [36]	Flujo de Datos	Graph matching	5 Benignos,563 Maliciosos (6 familias)
Pfoh et al. [37]	Secuencias de WinAPI calls	SVM	285 Benignos,1943 Maliciosos
Xiao and Stibor [38]	Secuencias de WinAPI calls	Latent Dirichlet allocation	168 Benignos,2880 Maliciosos
Canali et al. [39]	Secuencias de WinAPI calls, Tuplas, Cúmulo de WinAPI calls, Activiades, Argumentos	Firmas	363k Benignos,7k Maliciosos
Lanzi et al. [40]	Secuencias de WinAPI calls, Activiades	Firmas	362k Benignos, 10k Maliciosos

Tabla 1. Recopilación de trabajo relacionado en detección de malware basado en llamadas al sistema [24]

En la presente investigación a diferencia de [23] dónde se lleva a cabo un experimento con tres diferentes muestras de malware de diferente familia, la exactitud presentada es en promedio mayor al 90%, para llevar a cabo la clasificación se utilizaron 15 minutos de registros de Contadores de Rendimiento de Windows a manera de comportamiento normal,

y después de ese lapso de tiempo se ejecuta cada muestra para obtener su respectivo reporte de contadores generando los datos correspondientes de la clase. Por otro lado, se menciona que son diez veces las que se repite el registro de contadores, en los tres experimentos de esta investigación se utilizó el software Weka para obtener la exactitud del modelo. Por otro lado, toma el fenómeno como antes de ejecutar el malware y después de ejecutar el malware. En nuestro caso como datos de clase se usaron las repeticiones en ejecución de las cuatro diferentes muestras de malware de dos familias, en lo cual tuvimos menor exactitud, pero con un mayor número de registros. El problema con el planteamiento del escenario de este experimento radica en no considerar que al ser un censo de variables físicas convendría repetirlo más de 10 veces. Así mismo, no se menciona la tarea de Hardening a la Sandbox ahí creada. El utilizó un gradiente para discretizar sus datos en lugar del promedio. Además de que no detalla los contadores de rendimiento que fueron utilizados para su experimento. Y no especifica las condiciones en que su comportamiento normal fue muestreado. Sólo se muestran los resultados del Naive Bayes de Weka sin comparar la exactitud de su base de conocimiento con otros algoritmos de aprendizaje supervisado.

En [24] no ahonda en la minería de datos, sólo considera los resultados de clasificación, y al igual que en [23] para calcular el modelo se empleó Weka. Por otro lado, su infraestructura ofrece una mayor cantidad de generación de datos y contó con una mayor cantidad de malware para realizar sus experimentos. Se enfoca totalmente a la clasificación de malware y no considera la oportunidad de poder diferenciar malcode usando ese mismo modelo. Realiza su investigación con n-gramas de una cardinalidad máxima de 5 además de que los reportes de process monitor que utilizan son de mínimo 1000 llamadas al sistema por cada muestra. En nuestro caso el número máximo de cardinalidad de n-gramas es de 10, además en nuestro segundo experimento se realizó minería de datos sobre los n-gramas más relacionados con el comportamiento de una muestra de malware.

1.2 Hipótesis

Se podrá predecir el comportamiento malicioso, además de la clasificación del malware al emplear algoritmos de aprendizaje supervisado en los contadores de rendimiento de Windows y de manera independiente las llamadas al sistema operativo Windows.

1.2.1 Hipótesis específicas

Al usar los reportes de contadores de Windows, generados con la ejecución de las cuatro diferentes muestras de malware y el algoritmo Naive Bayes, se podrá determinar la presencia de malware según los patrones de comportamiento obtenidos con los valores de Épsilon y Score.

Al aplicar el algoritmo de aprendizaje supervisado de máquinas de soporte vectorial a las bases de conocimiento de n-gramas resultantes de variar su tamaño de uno a 10 y los cuales se obtienen de las Win API calls de cada muestra, como datos de entrenamiento se podrá clasificar: Troyanos, Gusanos, Virus, Troyanos Espía, Puertas Traseras y Rootkits y Whiteware, además de mostrar que modelo de n-grama presenta una mejor exactitud de clasificación. Así mismo dichas bases de conocimiento nos podrán servir para determinar qué n-gramas están más relacionados con procesos maliciosos después de realizar un proceso de minería de datos con nuestra implementación de Naive Bayes, así como

determinar la existe de correlación de n-gramas con mayor tamaño manteniendo estructuras de menor dimensión.

1.3 Objetivos

El objetivo general consiste en determinar los patrones de comportamiento en las fuentes de información que estén relacionados con la presencia de malware, además de obtener el rendimiento que representa emplear algoritmos de aprendizaje supervisado como una alternativa al proceso que llevan a cabo los analistas de malware para identificar que una muestra sea maliciosa y la familia a la que pertenece.

1.3.1 Objetivos específicos

Determinar la viabilidad de aplicar el algoritmo Naive Bayes a los contadores de rendimiento de Windows para la detección de malware y la extracción de variables con su valor correspondiente correspondiendo a una alta probabilidad de presencia de malware en el sistema.
Clasificar: Troyanos, Gusanos, Virus, Troyanos Espía, Puertas Traseras y Rootkits y Whiteware. A través de sus Win API calls empleando un modelo de lenguaje basado en n-gramas variando su tamaño de uno a diez. Así como determinar el modelo de n-gramas que presenta la mejor exactitud de clasificación.
Obtener los n-gramas más relacionados con procesos maliciosos empleando Naive Bayes sobre cada variante de las diez bases de conocimiento de n-gramas que contemple el mismo número de registros de malware del mismo tipo y whiteware para su análisis.

1.4 Aportes

1.4.1 Aportes a la Industria

Los datos que se utilizan para generar el modelo de aprendizaje supervisado pueden obtenerse con un script y/o programa con un bajo consumo en procesador a nivel host en comparación con las soluciones antivirus. Por otro lado, al obtener los datos en tiempo, idealmente podrá generarse una alerta no con una diferencia en tiempo de actualización de la base de datos de malware sino en el momento en que se detecte un comportamiento malicioso, debido a que cómo ya se ha mencionado, la mayoría de programas anti-virus en su forma autónoma funcionan con detección de firma y de las modificaciones en directorios de otros programas. Sin embargo, el primer enfoque no es suficiente debido al ofuscado y/o malware polimórfico; y en el segundo, este genera falsos positivos y sólo monitorea rutas conocidas.

1.4.2 Aporte Académico

En la literatura revisada ninguna manifiesta algún proceso de Hardening a su Sandbox. Esto es fundamental, ya que como se ha mencionado anteriormente el comportamiento malicioso no puede ser observado si el malware detecta que está siendo analizado (en ciertas muestras). En nuestro caso, utilizamos el programa Pafish como referencia para llevarla a cabo.

Por otro lado, el algoritmo de Naive Bayes con el que trabajamos cuenta con una técnica de Coarse Grain la cual nos permite convertir una variable de valor continuo en una de valor

categórico. Al mismo tiempo empleamos el suavizado de Laplace para una mejora en el rendimiento del clasificador.

En la presente investigación, a diferencia de muchos artículos en la literatura que realizan el análisis de comportamiento con reportes de syscalls utilizando modelos de secuencias y/o los resultados del análisis estático realizados por la herramienta SandBox [4], [7], [13]-[41], usamos las llamadas al sistema del tipo WinAPI y un modelo de lenguaje basado en n-gramas variando su tamaño de uno a diez, además de determinar que n-gramas de Win API calls están relacionados con la presencia de malware para cada una de las bases de conocimiento, y con ello analizar si existe una correlación entre los n-gramas de cada una.

CAPÍTULO II

« My definition implies that whether a program is malware depends not so much on its capabilities but, instead, on how the attacker uses it… Behind malicious software there is usually a human or organization that is making use of its capabilities for malicious purposes. »
Lenny Zeltser

Malware, análisis estático y dinámico

Contextualizando los ataques informáticos sin importar su vector de ataque, donde su impacto operacional es a través de la instalación de malware de cualquier tipo y cuyo objetivo es el sistema operativo Windows sin considerar su defensa según la taxonomía AVOIDIT (Attack Vector, Operational Impact, Defense, Information Impact, and Target) [42] que se muestra en la figura 1, este tiene lugar en un sin fin de intrusiones e incidentes de seguridad en cómputo. Cualquier software que cause daño a un usuario, computadora, o red puede ser considerado de este tipo, incluyendo virus, troyanos, gusanos, rootkits, spyware, etc. El objetivo del análisis de malware es usualmente proveer la información necesaria para responder a una intrusión en la red, para lograrlo se realizan fundamentalmente análisis de tipo estático y dinámico. Cuando se analiza la muestra, el objetivo es: determinar que comportamiento tiene, cómo detectarlo en la red además de cómo medir y resguardar el daño que pueda causar. [43]

2.1 Malware y su clasificación

El malware, también conocido como código malicioso y software malicioso, se refiere a un programa que se inserta en un sistema, por lo general de forma encubierta, con la intención de comprometer la confidencialidad, integridad o disponibilidad de los datos de la víctima, de aplicaciones, o del sistema operativo, o de otro modo en molestar o perturbar a la víctima. [44] Las principales variantes de malware son las siguientes:

a) Rootkit: Programas maliciosos que se ocultan en el sistema a través de modificaciones en las herramientas del sistema, filtrando activamente información de estado del sistema de los usuarios, enmascarando la presencia de archivos, servicios y canales de comunicación maliciosos. [45]

Figura 1. Taxonomía de Ataques Informáticos AVOIDIT [42]

b) Backdoor: Las puertas traseras son un método externo en el proceso de autenticación o en otros controles de seguridad con el fin de acceder a un sistema de cómputo o a los datos contenidos en el mismo. [41]

c) Keylogger: Programa en el que un atacante puede robar las credenciales e información confidencial de la computadora de la víctima. [46]

d) Ransomware: Compromete las computadoras víctima y cifra los datos almacenados en ellas y luego pide rescate para conseguir la llave que descifre los datos. Mientras que este tipo de malware puede ser una molestia para una persona, a nivel empresarial resulta devastador. [47]

e) Browser Hijacker: Es un programa que modifica el comportamiento por default del navegador. Dependiendo del recurso que sea controlado, podemos distinguir en diferentes tipos de secuestradores de navegador. [48]

f) Rogue security software: Software cuya finalidad es la estafa, hace uso de ingeniería social para explotar el miedo de un usuario de computadora para revelar información sensible, perder datos importantes y o causar daños irreversibles en el hardware. [49]

g) Virus: Un programa que es usualmente oculto dentro de otro programa aparentemente inocuo y que produce copias de sí mismo e inserta otros programas y usualmente realiza una acción maliciosa como destruir datos. [47]

h) Gusano: Un programa usualmente pequeño que auto replica su contenido en sí mismo y que invade computadoras en una red y generalmente realiza acciones destructivas. [47]

i) Troyanos: Es un software malicioso que se empaqueta junto con una pieza útil del software o se hace pasar por una pieza de software útil. Una vez que el troyano es activado, que por lo general pasa desapercibido por el usuario, se libera una carga útil de ellos ya sea como un virus o una puerta trasera que puede permitir a un usuario acceder remotamente al sistema. [50]

j) Adware: aplicaciones de software sobre una computadora cliente que genera ventanas emergentes con avisos que no están relacionados o autorizados por los sitios web que alguien elige visitar. Estos avisos intrusivos pueden tomas más de una forma: anuncios emergentes u ocultos, banners, páginas web redirigidas, o incluso correos electrónicos tipo spam. [51]

k) Spyware: software que obtiene información de una persona u organización sin su conocimiento y que puede enviar tal información a otra entidad sin el consentimiento del cliente, o que impone el control sobre una computadora sin el conocimiento del cliente. [51]

l) Spam: se refiere a los mensajes masivos no solicitados distribuidos a través de los sistemas de mensajería electrónica. Los ejemplos incluyen anuncios, estafas, phishing, etc. Debido al bajo costo de operación, el spam electrónico ha crecido de manera constante y se extendió a muchos medios de comunicación, como SMS, motor de búsqueda web, blog y foro recientemente. [52]

m) Bomba lógica: ejecuta un conjunto de instrucciones para comprometer a un sistema de información basado en la lógica definida por su creador. Son usualmente programas que usan el tiempo o un evento como disparador. Cuando las condiciones se cumplen, se ejecuta el código de su carga útil. Generalmente es usado por empleados descontentos que planean vengarse de sus jefes o para ganancias financieras. [53]

n) Bot: Es un programa que hace cualquier acción basada en instrucciones recibidas de su maestro o controlador. Una red de este tipo de robots se llama Botnet. IRC es uno de los canales comunes que los controladores utilizan para comunicarse con redes de bots enteras. [53]

o) Scareware: se basa en ingeniería social para inducir miedo en el usuario infectado para comprar algo. Por lo general tiene una interfaz gráfica que lo hace parecer un programa de seguridad o de antivirus. Se informa a los usuarios que hay código malicioso en su sistema y que la única manera de deshacerse de él es para comprar su “software”, cuando en realidad, el software sólo quita el scareware. [43]

p) Downloader: descargan otros códigos maliciosos. Son comúnmente instalados por los atacantes cuando ellos obtienen acceso por primera vez al sistema, posteriormente se descargará e instalará el código malicioso adicional. [43]

q) Launcher: generalmente no utilizan técnicas tradicionales para ejecutar otros malcodes con el fin de obtener mayores privilegios en el sistema o no dejar evidencias. [43]

A menudo la clasificación del malware puede extenderse en diferentes categorías. Por ejemplo, un programa podría tener un keylogger que obtiene las contraseñas y un componente gusano que envía spam. Para ser precisos, no se debe clasificar el malware únicamente considerando su comportamiento.

El malware también puede ser clasificado en función de si el objetivo del atacante es masivo o selectivo. En el primer caso tomando como ejemplo el Scareware, adopta el enfoque de un arma que está diseñada para infectar al mayor número de máquinas posible. De los dos objetivos es el más común y suele ser menos sofisticado respecto a la detección y defensa contra este, porque el software de seguridad se dirige a él. Análogamente el malware dirigido, tomando como ejemplo un tipo avanzado de Backdoor, se adapta a una organización específica, lo cual representa una amenaza mayor para las redes respecto al malware con objetivo masivo, ya que una solución de seguridad difícilmente podrá defender contra este tipo de amenaza sin realizar un análisis detallado de este. [43]

2.2 Análisis Estático

En su forma básica consiste en examinar el archivo ejecutable sin ver las instrucciones actuales. Este análisis sirve para confirmar que el archivo es malicioso, además de proveer parte de la información de su funcionalidad. El análisis estático básico es sencillo y puede ser rápido, pero es en gran medida ineficaz contra el malware sofisticado, y se puede perder de comportamientos importantes. Por otro lado, el análisis estático avanzado consiste en aplicar ingeniería inversa a la muestra de malware al cargar el ejecutable dentro de un desensamblador y revisar las instrucciones del programa para descubrir lo que hace. Las instrucciones son ejecutadas por el CPU, entonces el análisis estático avanzado nos muestra lo que hace el programa. Sin embargo, este análisis tiene una curva de aprendizaje muy pronunciada y requiere conocimiento especializado en el desensamblado del código. [43]

2.2.1 Escaneo de la muestra con antivirus

Cuando primero analizamos un prospecto de malware, un buen primer paso es ejecutar un escaneo sobre la muestra con diferentes programas antivirus, de esta manera podemos saber si la muestra ya ha sido identificada como maliciosa. Pero las herramientas antivirus no son del todo perfectas. Se basan principalmente en una base de datos de muestras de código conocido como sospechoso (firmas de archivo), así como el análisis del comportamiento y de reconocimiento de patrones (heurística) para identificar los archivos sospechosos. Uno de los principales problemas con este primer análisis es que el creador de malware puede modificar fácilmente su código, cambiando de este modo la firma de su programa y evadir los antivirus.

Debido a que los diversos programas antivirus utilizan diferentes firmas y heurísticas, es útil ejecutar programas antivirus diferentes contra la misma pieza de malware sospechoso. Por otro lado, sitios web como VirusTotal (http://www.virustotal.com/) le permiten cargar un archivo para el escaneado de múltiples motores de antivirus. VirusTotal genera un informe que proporciona el número total de los motores que han marcado el archivo como malicioso, el nombre de malware, y, en su caso, información adicional sobre el programa malicioso. [43] En la figura 2 se muestra el reporte de la muestra Rootkit.Win32.Vandi.df.

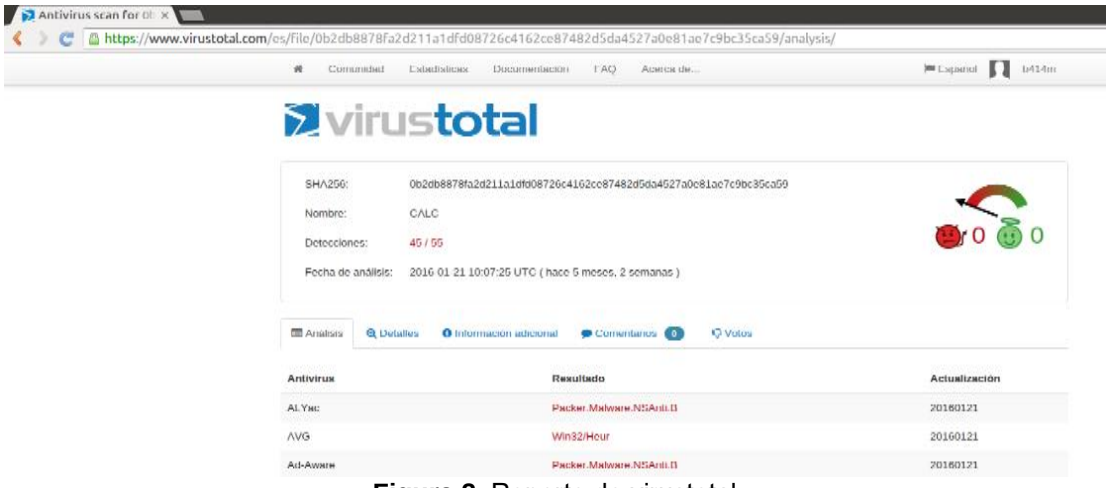

Figura 2. Reporte de virustotal

2.2.2 Firmas de malware con métodos Hash

El Hashing es un método comúnmente utilizado para identificar de forma exclusiva el malware. El software malicioso se ejecuta a través de un programa de hash que produce un hash único que identifica el malware (una especie de huella dactilar). La función hash Message-Digest Algorithm 5 (MD5) es el más común para el análisis de malware, aunque el algoritmo de hash seguro 1 (SHA-1) también es popular. [43]

Por ejemplo, utilizando el freeware md5deep para calcular el hash de la muestra utilizada para el reporte de virus total, generaría el siguiente resultado con la muestra Rootkit.Win32.Vanti.df (en la segunda línea, la columna izquierda corresponde a la firma md5 del archivo y la otra a la ruta absoluta del mismo):

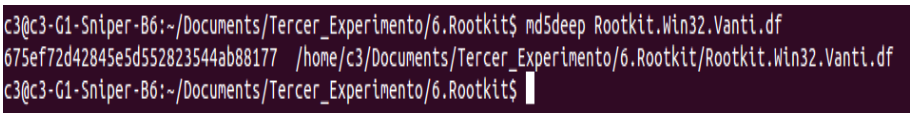

Figura 3. Firma MD5 con md5deep

2.2.3 Cadenas, funciones y cabeceras

Una cadena en un programa es una secuencia de caracteres tales como "esto es un ejemplo de cadena". Un programa contiene cadenas si se imprime un mensaje, se conecta a una dirección URL o copia un archivo a una ubicación específica. Buscando a través de las cadenas puede ser una forma sencilla de obtener pistas acerca de la funcionalidad de un programa. Por ejemplo, si el programa accede a una URL, entonces verá la URL visitada almacena como una cadena en el programa. En el contexto de Windows el término gran cadena de caracteres describe su implementación de cadenas Unicode, las cuales varían sutilmente respecto a los estándares Unicode. [43]

La siguiente figura muestra parte de las cadenas de la muestra Rootkit.Win32.Vanti.df:

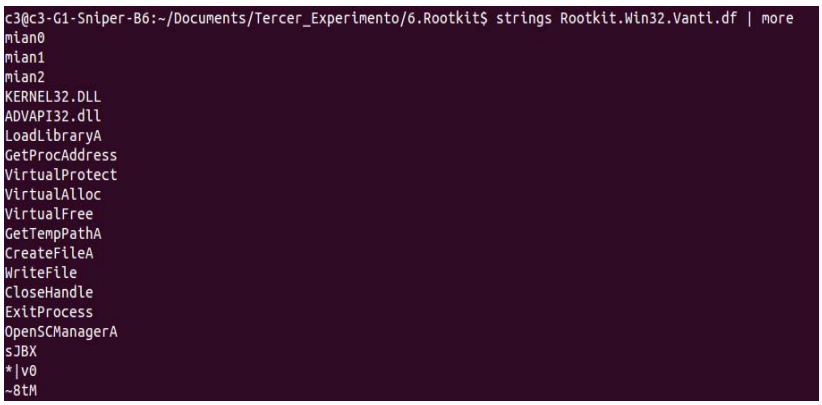

Figura 4. Resultado de cadenas con strings

2.2.4 Malware empaquetado y ofuscado

Los desarrolladores de malware suelen empaquetar u ofuscar sus archivos para hacerlos más difíciles de detectar y analizar. Los programas de ofuscado son aquellos dónde sus creadores pretenden ocultar su ejecución. Por otro lado, los programas de empaquetado son un subconjunto de programas de ofuscado en los cuales el programa malicioso es

comprimido y no puede ser analizado. Los programas legítimos incluyen siempre muchas cadenas en comparación al malware que es empaquetado u ofuscado. [43]

La siguiente figura muestra el empaquetado utilizado en la muestra Rootkit.Win32.Vanti.df:

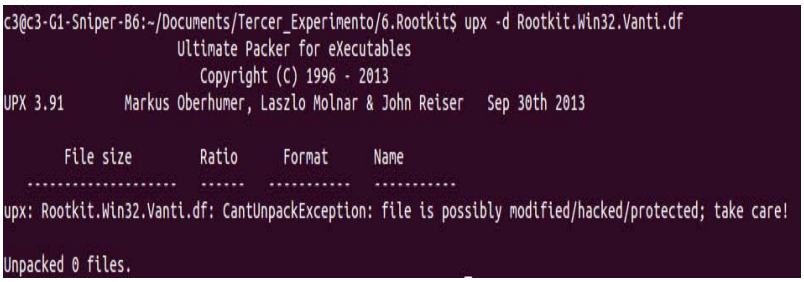

Figura 5. Desempaquetado UPX

2.2.5 Librerías Vinculadas y Funciones

Una de las partes más importantes de información que se pueden obtener de un ejecutable, es la lista de funciones que importa. Las directivas de import, son funciones usadas por un programa que está actualmente almacenado un programa diferente, tal como librerías que contienen funcionalidad común para muchos programas. El código de las librerías puede ser integrado al programa principal al vincularlo. [43]

2.3 Análisis Dinámico

Las técnicas en el análisis dinámico básico involucran ejecutar el malware y observar su comportamiento en el sistema para remover la infección, producir huellas efectivas de comportamiento, o ambas. Sin embargo, antes de poder ejecutar malware de forma segura se debe crear un entorno que permita estudiar la muestra sin el riesgo de daño al sistema o red. Por otro lado, las técnicas avanzadas de análisis dinámico usan un debugger para examinar el estado interno de un código malicioso mientras se ejecuta. Además de proveer otra forma de extraer información detallada de un ejecutable. Estas técnicas son las más útiles cuando se trata de obtener información que es difícil de obtener con otras técnicas. [43]

2.3.1 Monitor de Procesos

El monitor de procesos, o procmon es una herramienta avanzada de monitorio para Windows que provee una forma para monitorear ciertos registros, archivos del sistema, red, procesos y actividad. Combina y mejora la funcionalidad de dos herramientas: FileMon y RegMon. Aunque procmon captura muchos datos, no captura todo. Por ejemplo, puede perder la actividad del controlador de dispositivo de un componente en modo usuario comunicándose con un rootkit a través de los controles de entrada y salida del dispositivo, así como ciertas funciones GUI como SetWindowsHookEx. Aunque procmon puede ser una herramienta útil, usualmente no debe ser usada para registrar actividad de red, debido a que no presenta el mismo rendimiento en las diferentes versiones de Windows. procmon monitorea todas las llamadas al sistema que pueda obtener tan pronto como es ejecutado.

Debido a que muchas llamadas al sistema existen en máquinas Windows (algunas veces más de 50,000 eventos en un minuto), es usualmente imposible observarlas todas. Como un resultado, debido a que procmon usa RAM para registrar eventos hasta que el usuario lo detenga, puede colapsar una máquina virtual usando toda su memoria disponible.
Procmon muestra columnas configurables que contienen información sobre los eventos individuales, incluyendo el número de la secuencia de evento, fecha y hora, nombre del proceso que causa el evento, la operación de eventos, ruta usada por el evento, y resultado del evento. [43]

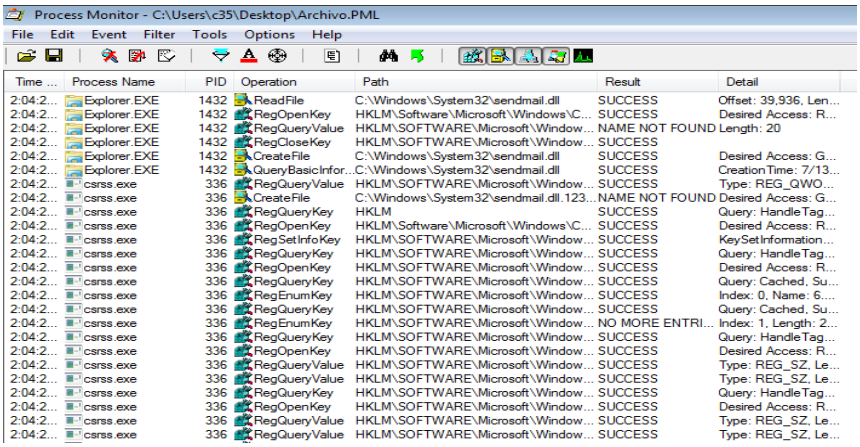

Figura 6. Monitoreo de procesos con procmon

2.3.2 Exploración de Procesos

El explorador de procesos, freeware de Microsoft, es un administrador de tareas muy útil que debe estar en ejecución cuando se está realizando un análisis dinámico. Éste puede proporcionar información valiosa sobre los procesos que se están ejecutando en un sistema. Además de poder utilizar el explorador de procesos para listar: los procesos activos, DLL cargadas por un proceso, varias propiedades del proceso, y la información general del sistema. También permite matar algún proceso, cerrar la sesión de usuarios, ejecutar y validar procesos. Process Explorer supervisa los procesos que se ejecutan en un sistema y los muestra en una estructura de árbol que muestra las relaciones secundarios y primarios. [43]

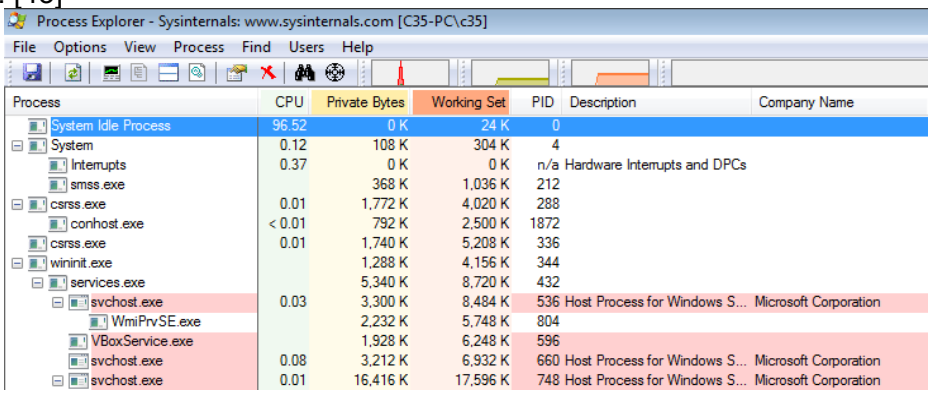

Figura 7. Monitoreo de procesos con Process Explorer

Process Explorer muestra cinco columnas: Proceso (nombre del proceso), PID (identificador de proceso), la CPU (uso de CPU), Descripción y Nombre de la compañía. La vista se actualiza cada segundo. De forma predeterminada, los servicios se destacan en rosa, los procesos en azul, los nuevos procesos en verde, y los procesos terminados en

rojo. Los elementos marcados con verde y rojo son temporales y se eliminan después que el proceso ha comenzado o terminado. [43]

2.3.3 Comparación de Registros con Regshot

Regshot es una herramienta freeware de comparación del registro que le permite tomar y comparar dos instantáneas del registro de Windows. Para utilizar Regshot en el análisis de malware, se toma la primera snapshot, luego se ejecuta el software malicioso (esperando a que termine de hacer cualquier cambio en el sistema para después analizarlo). Posteriormente se toma la segunda snapshot para finalmente comparar las dos snapshots creadas. [43]

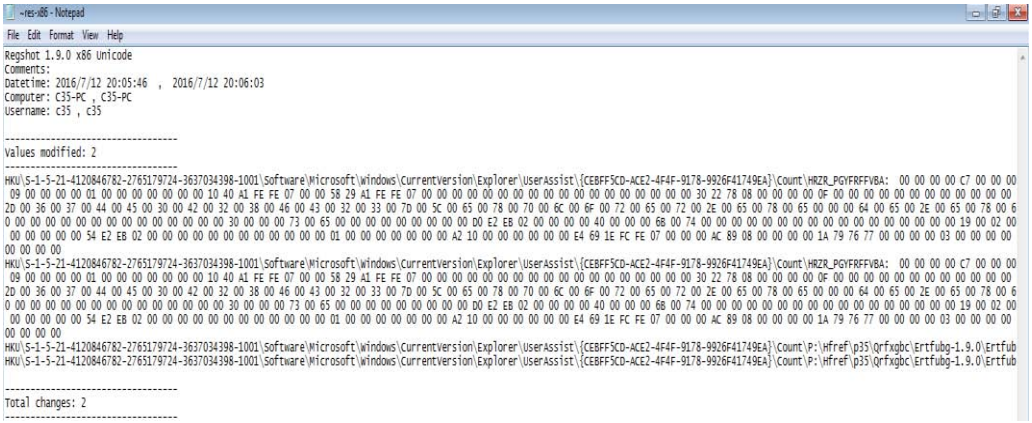

Figura 8. Comparación de captura de registros con RegShot

Como se puede apreciar, tras la ejecución de la muestra Rootkit.Win32.Vanti.df, se modificaron dos claves de registro (ambas con el siguiente prefijo de ruta):

HKU\S-1-5-21-4120846782-2765179724-3637034398-1001\Software\Microsoft\...

2.3.4 Análisis de tráfico de red

Wireshark es un sniffer de código abierto, una herramienta de captura de paquetes que intercepta y registra el de tráfico de red. Wireshark proporciona una visualización, análisis del flujo de paquetes, y un análisis en profundidad de los paquetes individuales. Se puede utilizar para analizar las redes internas y uso de la red, depurar problemas de la aplicación, y estudiar los protocolos de cada capa. Pero también puede ser utilizado para rastrear contraseñas, protocolos de red, ingeniería inversa, robar información sensible. En la siguiente figura se muestra la captura de paquetes por la muestra Net-Worm.Win32.Aspxor.ba. [43]

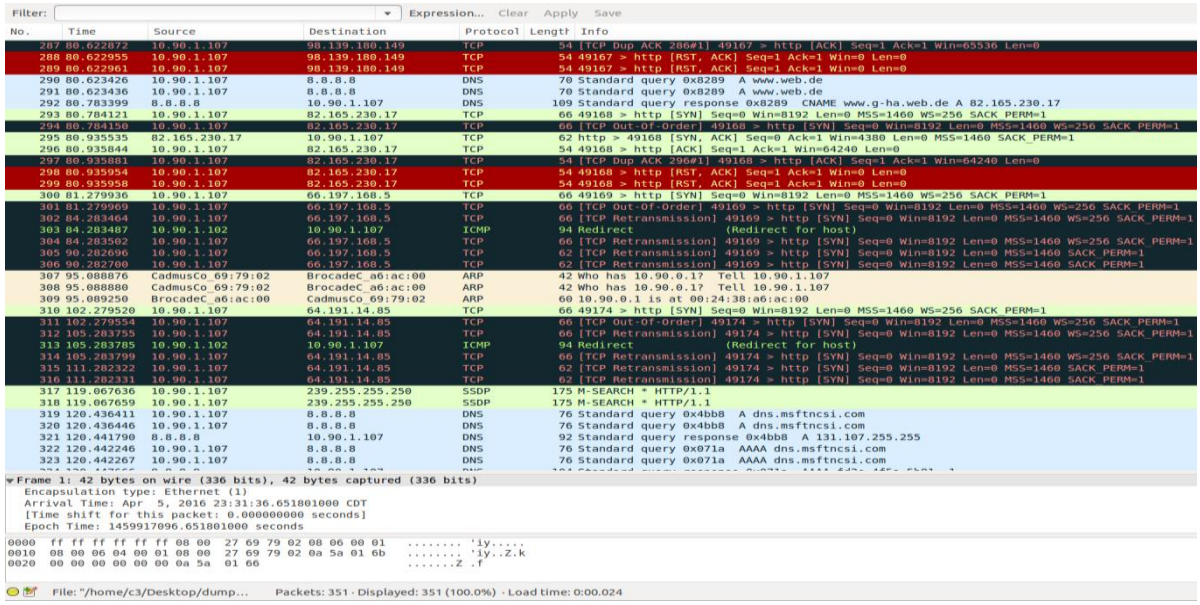

Figura 9. Captura de paquetes con Wireshark

Wireshark se sabe que tiene muchas vulnerabilidades de seguridad, así que asegúrese de ejecutarlo en un entorno seguro.

2.4 Herramientas de Sandboxing

Muchos de los productos integrales que se ofrecen como software de seguridad, se pueden utilizar para llevar a cabo un análisis dinámico básico, y los más populares usan el método de Sandboxing. El cual corresponde a un mecanismo de seguridad para la ejecución de los programas sospechosos en un ambiente seguro para evitar dañar los sistemas en producción. Además de comprender entornos virtualizados que a menudo simulan los servicios de red de alguna manera para asegurar que el software o software malicioso está probando funcionarán normalmente.

Sandbox como: Norman SandBox, GFI Sandbox, Anubis, Joe sandbox, ThreatExpert, BitBlaze, etc. Analizan el malware de forma gratuita. Actualmente, Norman SandBox y GFI Sandbox (anteriormente CWSandbox) son las más populares, entre las soluciones profesionales de seguridad informática. Estas cajas de arena o Sandbox, proporcionan resultados fáciles de entender, y son ideales para un análisis preliminar del malware, siempre y cuando se esté dispuesto a someter el malware a los sitios web que funcionan como Sandbox.

Desventajas de Sandbox:

a) Ejecuta el programa, sin opciones de línea de comandos. Si el malcode requiere opciones de línea de comandos, no va a ejecutar cualquier código que se ejecuta sólo cuando se proporciona una opción. Además, si la muestra está a la espera de un paquete de control y comando para ser devuelto antes de lanzar una puerta trasera, la puerta trasera no se ejecutará en la sandbox.

b) La sandbox también puede no registrar todos los eventos. Por ejemplo, si el malware está ajustado a ejecutarse en cierta fecha y hora, es posible que no se pueda registrar su comportamiento.

Otras desventajas potenciales incluyen los siguientes:

c) El malware a menudo se detecta cuando se está ejecutando en una máquina virtual, y si se detecta una máquina virtual, el malware podría dejar de funcionar o se comportan diferentemente. No todas las sandbox toman en cuenta esto.

d) Algunos malware requieren la presencia de ciertas claves de registro o archivos en el sistema que no se pueden encontrar en la sandbox. Estos podrían ser necesarias para contener los datos legítimos, tales como comandos o claves de cifrado.

e) Si el malware es un archivo DLL, no se invocan ciertas funciones exportadas adecuadamente, porque un DLL no se ejecutará tan fácilmente como un ejecutable.

f) El sistema operativo del entorno de sandboxing puede no ser correcta para el malware. Por ejemplo, el programa malicioso puede bloquearse en Windows XP, pero funcionar correctamente en Windows 7.

g) Una sandbox no puede decir lo que el malware hace. Es decir; informa la funcionalidad básica, pero no puede decir que el malware es un volcado de hash del Administrador de cuentas (también conocido como SAM) o un backdoor de keylogger cifrado, por ejemplo.

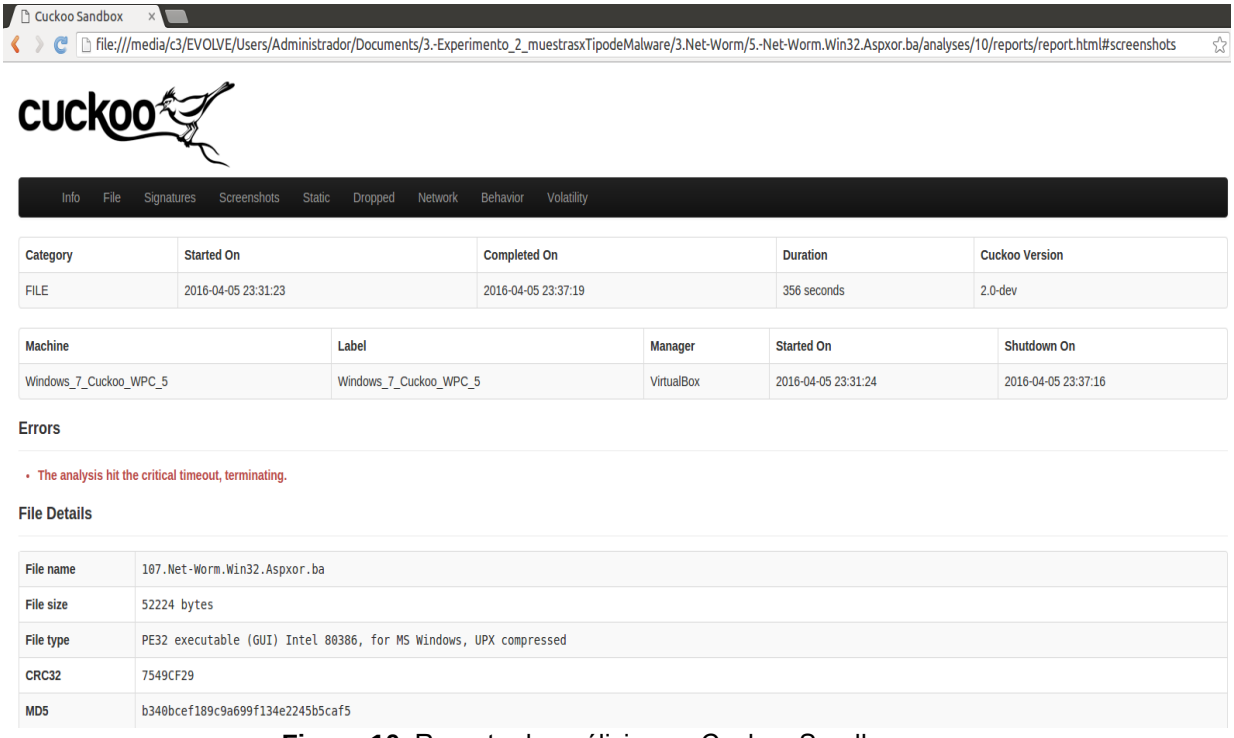

Figura 10. Reporte de análisis con Cuckoo Sandbox

2.4.1 Cuckoo Sandbox

Como se describe en su sitio web oficial (http://www.cuckoosandbox.org/), Cuckoo es una herramienta del tipo sandboxing que tiene aplicaciones prácticas desde el punto de vista del análisis dinámico de malware. En lugar de analizar estáticamente el archivo binario, el fichero es ejecutado y monitoreado en tiempo real. En resumen, Cuckoo es un sistema de análisis de malware automatizado de código abierto que permite llevar a cabo el análisis de malware en un recinto de seguridad. Cuckoo comenzó como un proyecto de Google Summer of Code en 2010 dentro del Honeynet Project. Después del trabajo inicial que ocurrió en el verano de 2010, la primera versión beta se publicó el 5 de febrero de 2011, cuando Cuckoo fue anunciado públicamente por primera vez y distribuido en público.

Originalmente diseñado y desarrollado por Claudio "nex" Guarnieri, quien sigue siendo el principal desarrollador, además de coordinar todos los esfuerzos de los desarrolladores y colaboradores de ese proyecto. Cuckoo se utiliza para ejecutar y analizar automáticamente los archivos, además de obtener resultados completos de análisis que describen lo que hace el software malicioso, mientras este se ejecuta dentro de un sistema operativo Windows aislado. [54]

Cuckoo está diseñado para su uso en el análisis de los siguientes tipos de archivos:

a) Ejecutables genéricos de Windows
b) Archivos DLL
c) Documentos PDF
d) Documentos de Microsoft Office
e) URL
f) scripts PHP

Cuckoo también puede producir los siguientes tipos de resultados:

a) La secuencia de llamadas a la API de Win32 realizadas por todos los procesos generados por el malware.
b) Los archivos que se crean, suprimen, y descargados por el programa malicioso durante su ejecución.
c) Volcados de memoria de los procesos del malware.
d) El registro del tráfico de red en formato PCAP.
e) Capturas de pantalla del escritorio de Windows tomadas durante la ejecución del malware.
f) Volcados de memoria completa de las máquinas.

Cuckoo Sandbox consiste de un software de gestión central, que se ocupa de las ejecuciones de muestras de malware y también de su análisis. Cada análisis es realizado en la snapshot de máquina virtual aislada. La infraestructura de Cuckoo está compuesta por un ordenador central (software de gestión) y una serie de equipos hosts (máquinas virtuales para el análisis).

Los hosts ejecutan el componente principal de la sandbox que gestiona todo el proceso de análisis, mientras que los hosts se encuentran en entornos aislados, de manera que el malware se analiza y ejecuta de forma segura.

La siguiente figura muestra la arquitectura de Cuckoo:

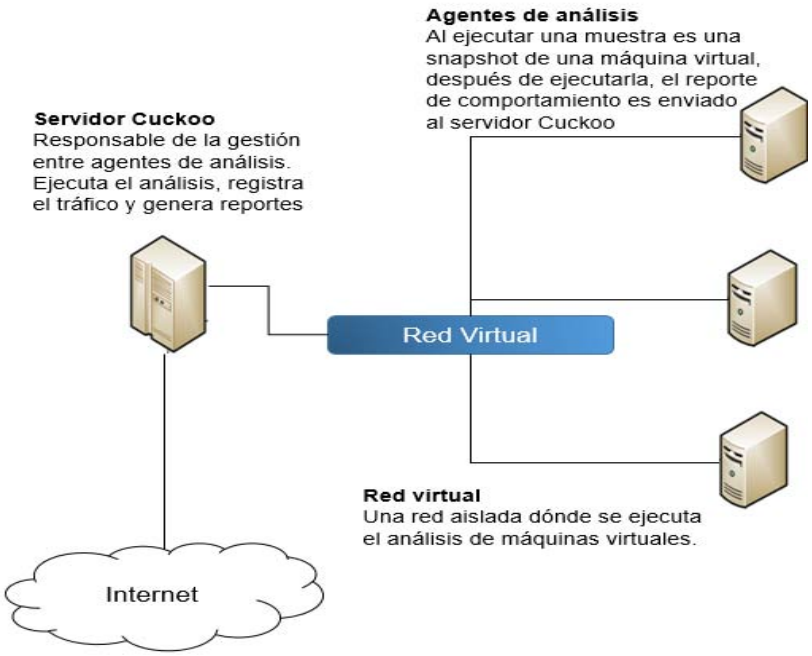

Figura 11. Reporte de análisis con Cuckoo Sandbox

CAPÍTULO III

« The question of whether a computer can think is no more interesting than the question of whether a submarine can swim. »
Edsger W. Dijkstra

Algoritmos de aprendizaje supervisado

El campo del aprendizaje automatizado está encaminado al tema de construir programas que automáticamente mejoren con el conocimiento empírico (asociación de vectores característicos a respuestas concretas). [55] Haciendo una analogía del aprendizaje, podemos pensar en él como un conjunto de herramientas y métodos que tratan de inferir patrones y generar intuición a partir de un histórico del mundo observable. Por ejemplo, si estamos tratando de enseñar a una máquina a reconocer los códigos postales escritos enfrente de los sobres, nuestros datos pueden consistirse a partir de fotografías de los sobres asociados con un registro del código postal al cual el sobre estaba dirigido. Es decir, dentro de algún contexto se puede tener un registro de las acciones de nuestros sujetos, aprender de estos registros, y luego crear un modelo de estas actividades que servirán de base a nuestra comprensión de este contexto en el futuro. [56] Contextualizando de esta forma el conjunto de entrenamiento consta de aquellos datos a los cuales se aplica cierto algoritmo de aprendizaje supervisado, el cual consiste de un conjunto de parejas (x, y), llamadas muestras de entrenamiento.

Las parejas se explican a continuación:

a) x: es un vector de valores, algunas veces llamado vector característico, cada valor o característica, puede ser categórico o numérico.
b) y: es la etiqueta, la clasificación o valores de regresión para los valores de x.

El objetivo del proceso de aprendizaje automatizado es descubrir una función $y = f(x)$ que tenga una mejor predicción del valor y asociado con cada valor de x.
El tipo de x en principio es arbitrario, pero hay casos comunes.

a) y: es un número real. El problema de aprendizaje automatizado es llamado regresión.
b) y: es un valor booleano verdadero o falso, comúnmente escrito como $+1$ y -1, respectivamente. Para este caso el problema es llamado clasificación binaria.
c) y: es un miembro de un conjunto finito de elementos llamados clase, el problema es llamado de clasificación multiclase.

d) y: es un miembro de un conjunto potencialmente infinito, por ejemplo, un árbol de análisis sintáctico para x, el cual es interpretado como una sentencia. [57]

Principalmente existen tres estilos de aprendizaje o modelos de aprendizaje que un algoritmo puede tener, los cuales son: supervisado, no supervisado y semi-supervisado. Esta taxonomía o forma de organizar algoritmos de aprendizaje automatizado es útil porque obliga a pensar en los papeles de los datos de entrada y el proceso de preparación del modelo además de seleccionar el más adecuado para cierto problema con el fin de obtener el mejor resultado.

Aprendizaje supervisado

Los datos de entrada se denominan datos de entrenamiento, y tienen una etiqueta o un resultado conocido, por ejemplo: spam / no - spam. Un modelo se crea mediante un proceso de entrenamiento en el que se requiere hacer predicciones y se corrige cuando esas predicciones son erróneas. El proceso de entrenamiento continúa hasta que el modelo alcanza un nivel deseado de precisión en los datos de entrenamiento. Los problemas que resuelve esencialmente son de clasificación y regresión.

Aprendizaje no supervisado

Los datos de entrada no se encuentran marcados y no tienen un resultado conocido. Un modelo se crea mediante la deducción de estructuras presentes en los datos de entrada. Esto puede ser a través de la extracción de reglas generales, también a través de un proceso matemático para reducir sistemáticamente la redundancia o, puede ser para organizar los datos por similitud. Los problemas que resuelve son de agrupación, reducción de dimensionalidad y reglas de asociación.

Aprendizaje semi-supervisado

Los datos de entrada son una mezcla de ejemplos etiquetados y no etiquetados. El problema subyacente es de predicción, pero el modelo debe aprender las estructuras para organizar los datos, así como hacer predicciones. Los problemas que resuelve son de clasificación y regresión. Ejemplos de algoritmos son extensiones a otros métodos flexibles que hacen suposiciones acerca de cómo modelar los datos no etiquetados.

Algoritmos agrupados por similitud

Los algoritmos son a menudo agrupados por similitud en términos de su funcionamiento. Este es un método de agrupación útil como puede apreciarse en la figura 12, pero no es perfecto. Todavía hay algoritmos que podrían encajar con la misma facilidad en varias categorías como aprendizaje de cuantificación vectorial que es a la vez un método inspirado red neuronal y un método basado en la instancia. También hay categorías que tienen el mismo nombre que describe el problema y la clase de algoritmo como la regresión y el Clustering. [58] En la presente investigación se empleó el estilo de aprendizaje supervisado con los algoritmos de Máquinas de Soporte Vectorial y el Clasificador Bayesiano, los cuales se explican a continuación.

Figura 12. Clasificación por Similitud de los Principales Algoritmos de Aprendizaje Automatizado [58]

3.1 Clasificador Bayesiano

Uno de los métodos Bayesianos de aprendizaje más prácticos es el Naive Bayes, algunas veces llamado el clasificador Naive Bayes. En algunos dominios su rendimiento ha sido mostrado para ser comparado con los métodos de aprendizaje cómo redes neuronales y árboles de decisión [55].

Esencialmente el Naive Bayes funge como un clasificador binario. Dónde a partir de una muestra poblacional N que cumple con un vector de características $X = < x_1, x_2, x_3, \dots, x_{311} >$ (las cuales deben ser probabilísticamente independientes) y a partir de estas la muestra poblacional pertenece a un grupo denominado clase C y/o análogamente no pertenece a la misma, es decir a la no clase $\neg C$. Luego entonces desde el punto de vista frecuentista N_{x_i} se define cómo el número de individuos que cumplen con alguna de las características del vector X.

Por lo tanto, aquellos individuos que pertenecen a la clase se definen cómo la intersección del conjunto N con el conjunto C, es decir N_C. Dicho lo anterior y a partir de la definición de probabilidad de un evento que pertenezca a la clase $P(C)$dada una muestra poblacional se escribe cómo: $\frac{N_C}{N}$ y la probabilidad de cada característica $P(x_i)$ en el vector X cómo $\frac{N_{x_i}}{N}$. (Véase Obtención de fórmulas frecuentistas).

Finalmente, el objetivo del clasificador es determinar a modo de frecuencia: las características en X con una mayor probabilidad que están relacionadas con N_C, a través del cálculo de la probabilidad condicional $P(C|x_i)$, para lo cual utilizamos el Teorema de Bayes: [59]

$$P(C|x_i) = \frac{P(x_i|C) * P(C)}{P(x_i)} \dots (1)$$

3.1.1 Función de probabilidad en términos de frecuencia

Suponiendo que el vector X sólo consta de dos elementos x_1 y x_2, luego entonces:

$$P(C|x_1, x_2) = \sum_{x_1 x_2} P(C|x_1, x_2) * P(x_1, x_2) = \sum_{x_1 x_2} \left(\frac{N_{C_{x_1 x_2}}}{N_{x_1 x_2}}\right) * \left(\frac{N_{x_1 x_2}}{N}\right)$$

$$= \frac{N_{C_{x_1 x_2}}}{N}$$

Considerando que ambos elementos constituyen la clase:

$$\boldsymbol{P(C) = \frac{N_C}{N} \dots (2)}$$

Entonces el Teorema de Bayes puede escribirse en términos de frecuencia cómo:

$$P(C|x_1, x_2) = \frac{N_{C_{x_1 x_2}}}{N} =$$

$$\boldsymbol{P(C|x_i) = \frac{N_{C_{x_i}}}{N_{x_i}} \dots (3)}$$

Análogamente:

$$\boldsymbol{P(x_i) = \frac{N_{x_i}}{N} \dots (4)}$$

3.1.2 Definición de la variable Épsilon

En el cálculo de contribución de probabilidad para cada característica en X, existen variables que se encuentran más relacionadas con la clase, para lo cual para cada una de nuestras N_{x_i} calculamos la relación que existe entre su probabilidad condicional y su desviación estándar (considerando que existen en una distribución binomial para después aproximarlos a una distribución normal), la cual se define como: [60]

$$\sigma = \sqrt{\left(\frac{1}{n-1}\right)\sum_{r=1}^{n}(r_i - \bar{r})} \dots (5)$$

A partir de la definición de la distribución binomial:

$$p(n,N) = p^n(1-p)^{N-n} = C_1{}_n^N p^n(1-p)^{N-n} \dots (6)$$

Dónde $C_1 = \frac{N!}{n!(N-n)!}$

$$\sum_{n=0}^{N}\left(C_1{}_n p^n(1-p)^{N-n}\right) \dots (7)$$

Y el valor esperado

$$\langle x\rangle = \sum_x xp(x)\,;\langle f(x)\rangle = \sum_x f(x)p(x)$$

$$\langle n\rangle = \sum_{n=0}^{N} np(n,N) \dots (8)$$

Luego entonces, sustituyendo (7) en (8):

$$\sum_{n=0}^{N} nC_1{}_n^N x^n y^{N-n} = x\frac{d}{dx}\left(\sum_{n=0}^{N} nC_1{}_n^N x^n y^{N-n}\right)$$

A partir de la definición del binomio de Newton:

$$= x\frac{d}{dx}(x+y)^N = N*(x+y)^{N-1}*x$$

Dado que $x = p, y = (1-p)$:

$$\sum_{n=0}^{N} nC_1{}_n^N x^n y^{N-n} = N*p \dots (9)$$

Por otro lado:

$$\sum_{n=0}^{N} n^{2N}C_1{}_n^N x^n y^{N-n} = x\frac{d}{dx}x\frac{d}{dx}\left(\sum_{n=0}^{N} n^{2N}C_1{}_n^N x^n y^{N-n}\right)$$

$$= x\frac{d}{dx}x\frac{d}{dx}(x+y)^N = x\frac{d}{dx}(N*(x+y)^{N-1})$$
$$= N*(x+y)^{N-1} + Nx^2(N-1)(x+y)^{N-2}$$

$$= N*p + N(N-1)p^2 \ldots (10)$$

Considerando que $\sigma^2 = < n > -(< n >)^2; Dónde\ (< n >)^2 = (N*p)^2$

$$\sigma^2 = N*p + N(N-1)p^2 - N^2*p^2 = N*p + N^2*p^2 - N^2*p^2 - N^2*p^2 =$$

$$= N*p*(1-p)$$

$$\therefore \sigma = \big(N*p*(1-p)\big)^{\frac{1}{2}} \ldots (11)$$

Asumiendo que $N = N_{x_i}, y\ p = P(C)$ en (10), luego entonces el valor de Épsilon se define como:

$$\boldsymbol{\varepsilon = \frac{N_{x_i} * \big(P(C|x_i) - P(C)\big)}{\Big(N_{x_i} * P(C) * (1 - P(C))\Big)^{\frac{1}{2}}} \ldots (12)}$$

3.1.3 Definición de la variable Score

Definida como la medida de confiabilidad en términos de probabilidad en que cada característica de X pertenezca a la clase C, es decir la relación de probabilidad de la clase dado el vector de características sobre la probabilidad de la no clase a partir del mismo vector.

$$P(C|X) = \frac{\big(\prod_{i=1}^{N} P(x_i|C) * P(C)\big)}{P(X)} \ldots (13)$$

$$P(\neg C|X) = \frac{\big(\prod_{i=1}^{N} P(x_i|\neg C) * P(\neg C)\big)}{P(X)} \ldots (14)$$

Al dividir (13) en (14):

$$\frac{P(C|X)}{P(\neg C|X)} = \frac{\prod_{i=1}^{N} P(x_i|C) * P(C)}{\prod_{i=1}^{N} P(x_i|\neg C) * P(\neg C)}$$

Aplicamos logaritmo natural en virtud de las multiplicaciones:

$$\frac{P(C|X)}{P(\neg C|X)} = ln\left(\frac{P(C|X)}{P(\neg C|X)}\right) = \ln\left(\frac{\prod_{i=1}^{N} P(x_i|C) * P(C)}{\prod_{i=1}^{N} P(x_i|\neg C) * P(\neg C)}\right)$$

Luego entonces debido al productorio en los miembros del cociente, resulta que:

$$S(C|X) = \sum_{i=1}^{N} \ln\left(\frac{P(x_i|C)}{P(x_i|\neg C)}\right) + \ln\left(\frac{P(C)}{P(C\neg)}\right);$$

Luego entonces

$$\sum_{i=1}^{N} S(C|x_i) + \ln\left(\frac{P(C)}{P(C\neg)}\right);$$

$$\boldsymbol{S(C|x_i) = \ln\left(\frac{P(x_i|C)}{P(x_i|C\neg)}\right) \ldots (15)}$$

3.1.4 Suavizado de Laplace

En estadística el suavizado de adición, también llamado suavizado de Laplace, es una técnica usada para suavizar datos categóricos. Dada una observación $x = (x_1, x_2, \ldots, x_n)$ de una distribución multinomial con N intentos y vector de parámetros $\theta = (\theta_1, \theta_2, \theta_n)$, una versión suavizada de los datos nos ofrece el siguiente estimador:

$$\hat{\theta}_i = \frac{x_i + \alpha}{N + \alpha d} \qquad para\ i = 1,2, \ldots, n$$

Dónde el pseudo-conteo $\alpha > 0$ es el parámetro de suavizado ($\alpha = 0$ corresponde a un conjunto de datos sin suavizado). El suavizado de adición es un estimador de encogimiento, como la estimación resultante será entre la estimación empírica de $\frac{x_i}{N}$, y la probabilidad uniforme $\frac{1}{d}$. Usando la regla de sucesión de Laplace, algunos autores han argumentado que el valor de α debe ser 1 (también llamado suavizado de adición uno), aunque en la práctica se elige un valor más pequeño.

Desde un punto de vista Bayesiano, este corresponde al valor esperado después de la distribución, usando una distribución simétrica Dirichlet con parámetro α a priori. En el caso especial donde el número de categorías es 2, esto es equivalente a usar una distribución Beta como el conjugado a priori para los parámetros de la distribución binomial. [61]

3.1.5 Técnica de Coarse Grain

En el contexto de minería de datos y concretamente con el clasificador Bayesiano, esta técnica consta de la reducción efectiva del número de grados de libertad de los datos, la cual está asociada con la reducción de la cantidad de puntos de datos para el número de parámetros en la familia de modelos que se utilizan. De esta forma podemos convertir una variable de valor continuo a una de tipo discreto, lo cual brinda la posibilidad de agrupar los valores de los datos y obtener un valor de probabilidad por grupo, sacrificando precisión de la misma respecto al valor original. [62]

Para ello existen tres enfoques principales:

1) Eliminar los datos que no se comportan de manera homogénea respecto a la población total.
2) Obtener los rangos al tomar el mayor y el menor y dividir el resultado en una constante (generalmente se usa 10).
3) Obtener los rangos al tomar el mayor y el menor y dividir en rangos con el objetivo de que todos tengan el mismo número de valores posible.

A continuación, se presenta el pseudocódigo correspondiente al proceso de entrenamiento, dónde A es la matriz que representa la base de conocimiento.

$EntrenamientoNaiveBayes(A)$
1. $N \leftarrow ContarRegistros(A)$
2. $N_c \leftarrow ContarRegistrosEnLaClase(A)$
3. $N_{NOC} \leftarrow (N - N_C)$
4. $P(C) = \frac{N_c}{N}$
5. $P(NOC) = 1 - P(C)$
6. $\boldsymbol{for\ each}\ Column \in A$
7. $\boldsymbol{do}\ N_X \leftarrow ContarValorRepetido(A)$
8. $N_{XC} \leftarrow ContarValorRepetidoEnLaClase(A)$
9. $N_{XNOC} \leftarrow (N_X - N_{XC})$
10. $\boldsymbol{if}\ N_{XC} \leftarrow 0 \,||\, N_{XNOC} \leftarrow 0$
11. $\boldsymbol{then}\ cont \leftarrow cont + 1$
12. $\boldsymbol{else}$
13. $P(C|X) \leftarrow \frac{N_{XC}}{N_X}$
14. $P(X|C) \leftarrow \frac{N_{XC}}{N_C}$
15. $P(X|NOC) \leftarrow \frac{N_{XNOC}}{N_{NOC}}$
16. $\varepsilon \leftarrow N_X * \big(P(C|X) - P(C)\big) / Sqrt\big(N_X * P(C) * (1 - P(C))\big)$
17. $\alpha \leftarrow 2/cont$
18. $Score \leftarrow log((N_{XC} + \alpha) * (N_{NOC} + 2)/((N_C + 2) * (N_{XNOC} + 2)))$
19. $\boldsymbol{return}\ Score, \varepsilon$

3.2 Máquinas de Soporte Vectorial

Las máquinas de soporte vectorial es un algoritmo de clasificación que puede aplicarse a problemas de datos lineal y no linealmente separables. Una noción central en el desarrollo de SVM es el producto punto entre un x_i "vector de soporte" y un vector x del espacio de datos de entrada. Es fundamental mencionar que los vectores de soporte consisten en un pequeño subconjunto de puntos de los datos extraídos por el algoritmo de aprendizaje de la propia muestra de entrenamiento. De hecho, es a causa de esta propiedad central que el algoritmo de aprendizaje, involucrado en la construcción de una máquina de vectores de soporte, también se conoce como un método de kernel. [63]
Además, se basa en la siguiente suposición: Si dos clases de datos no pueden ser divididos por un hiperplano, entonces después de mapear los datos fuente a una dimensión lo suficientemente alta, debe existir el hiperplano de separación óptima. [57]

3.2.1 Hiperplano óptimo para patrones linealmente separables

Linealmente separable significa que un conjunto de datos puede ser dividido en las instancias con una ecuación lineal con la entrada del vector de entrenamiento. Considere la muestra de entrenamiento $\{x_i, d_i\}_{i=1}^{N}$, dónde $\mathrm{x_i}$ es el patrón de entrada para el i-ésimo ejemplo y $\mathrm{d_i}$ es la respuesta deseada correspondiente (la salida objetivo). Para comenzar, asumimos que el patrón (clase) representada por el subconjunto $\mathrm{d_i} = +1$ y el patrón

representado por el subconjunto $\mathrm{d_i} = -1$ son "linealmente separables". La ecuación de una curva de decisión en la forma de un hiperplano que hace la separación es:

$$\boldsymbol{w^T * x + b = 0} \ldots (\mathbf{16})$$

Dónde x es un vector de entrada, w es un vector de peso ajustable, y b es un umbral. Podemos escribir

$$\begin{matrix} w^T * x_i + b \geq 0 & para\ d_i = +1 \\ w^T * x_i + b < 0 & para\ d_i = -1 \end{matrix} \ldots (17)$$

Para un vector de peso dado w y un umbral b, la separación entre el hiperplano definido en ecu. (16) y el punto de datos más cercano se llama el margen de separación, denotado por p. El objetivo de una máquina de soporte vectorial es encontrar el hiperplano particular para el cual se maximiza el margen de separación p. Bajo esta condición, la superficie de decisión se nombra hiperplano óptimo. La ecuación (16) ilustra la construcción geométrica de un hiperplano óptimo para un espacio de entrada de dos dimensiones.
Sean w_o y b_o quienes denotan los valores óptimos del vector de pesos y de umbrales respectivamente. Correspondientemente, el hiperplano óptimo, lo que representa una superficie lineal de decisión multidimensional en el espacio de entrada, se define por

$$w_o^T * x + b_o = 0 \ldots (18)$$

La cual es rescrita de la ecuación (1). La función discriminante

$$g(x) = w_o^T * x + b_o \ldots (19)$$

Dada la medida algebraica de distancia de x al hiperplano óptimo. La forma más fácil de concebir la función objetivo es expresar x como:

$$x = x_p + r\left(\frac{w_o}{\|w_o\|}\right)$$

Dónde x_p es la proyección normal de x sobre el hiperplano óptimo y r es la distancia algebraica deseada; r es positivo si x está en el lado positivo del hiperplano óptimo y negativo si x está en el lado negativo. Debido a que por definición $g(x_p) = 0$, entonces:

$$g(x) = w_o^T * x + b_o = r * \|w_o\|$$

O equivalentemente,

$$r = \frac{g(x)}{\|w_o\|} \ldots (20)$$

La siguiente figura muestra una Ilustración sobre la idea de un hiperplano óptimo para los patrones linealmente separables: Los puntos de datos sombreados en azul son los vectores de soporte.

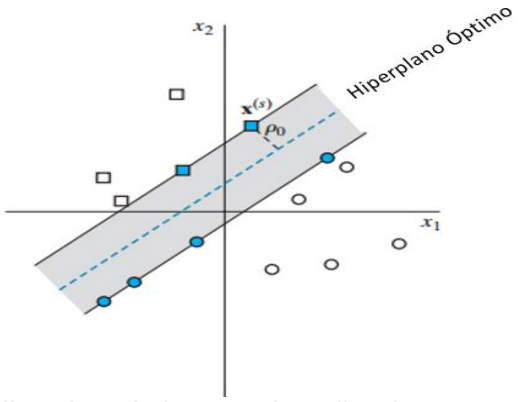

Figura 13. Hiperplano óptimo con datos linealmente separables

En particular, la distancia desde el origen (es decir, $x = 0$) al hiperplano óptimo está dado por $\frac{b_o}{\|w_o\|}$. Si $b_o > 0$, el origen está en el lado positivo del hiperplano óptimo; Si $b_o < 0$, está en el lado negativo. Si $b_o = 0$, el hiperplano óptimo pasa a través del origen. Puede apreciarse una interpretación geométrica en la Figura 13.
El problema en mano es encontrar los parámetros w_o y b_o para el hiperplano óptimo, dado el conjunto de entrenamiento $\varphi = \{(x_i, d_i)\}$. Cómo puede apreciarse en la figura 14 vemos que la pareja (w_o, b_o) debe satisfacer las siguientes restricciones:

$$\begin{matrix} \boldsymbol{w_o^T * x_i + b_o \geq 1} & \boldsymbol{para\ d_i = +1} \\ \boldsymbol{w_o^T * x_i + b_o < -1} & \boldsymbol{para\ d_i = -1} \end{matrix} \ldots (\mathbf{21})$$

Notar que si la ecuación (17) se mantiene –lo cual involucra que los patrones sean linealmente separables- podemos cambiar la escala de w_o y b_o tal que la ecuación (21) se mantenga; esta operación de escala no afecta a la ecuación (18).
Los puntos de datos particulares (x_i, d_i) para los que la primera o segunda línea de la ecuación (21) se satisface con el signo de igualdad son llamados vectores de soporte, de ahí el nombre de "máquinas de soporte vectorial." Todos los ejemplos restantes en la muestra de entrenamiento son completamente irrelevantes. Debido a su propiedad de distinción, los vectores de soporte juegan un papel destacado en la operación de esta clase de algoritmos de aprendizaje automatizado. En términos conceptuales, los vectores de soporte son aquellos puntos de datos que se encuentran más cerca del hiperplano óptimo y por lo tanto son los más difíciles de clasificar. Como tal, tienen una influencia directa en la ubicación óptima de la superficie de decisiones.
Considerar un vector de soporte $x^{(s)}$ para el cual $d^{(s)} = +1$. Entonces por definición:

$$g\left(x^{(s)}\right) = w_o^T * x^{(s)} + b_o = \mp 1 \qquad para\ d^{(s)} = \mp 1 \ldots (22)$$

En la siguiente figura se muestra una interpretación geométrica en dos dimensiones, de las distancias algebraicas de puntos al hiperplano óptimo.

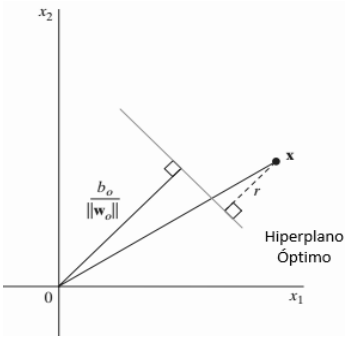

Figura 14. Distancias algebraicas de puntos al hiperplano óptimo

De la ecuación (20) la distancia algebraica del vector de soporte $x^{(s)}$ al hiperplano óptimo es:

$$r = \frac{g(x)}{\|w_o\|}$$

$$= \begin{cases} \frac{1}{\|w_o\|} & si\ d^{(s)} = +1 \\ -\frac{1}{\|w_o\|} & si\ d^{(s)} = -1 \end{cases} \quad \dots (23)$$

Dónde el signo positivo indica que $x^{(s)}$ yace en el lado positivo del hiperplano y análogamente el negativo indica que $x^{(s)}$ se ubica en el lado negativo del hiperplano. Sea p el valor óptimo del margen de separación entre las dos clases que constituyen el conjunto de entrenamiento φ. Entonces de (23), tenemos lo siguiente:

$$p = 2r$$

$$= \frac{2}{\|w_o\|} \dots (24)$$

La ecuación anterior establece que al maximizar el margen de separación entre clases binarias es equivalente a minimizar la norma Euclidiana del vector de pesos w.
En resumen, el hiperplano óptimo definido por ecu. (18) es único en tanto que el vector de peso w_o provea la separación máxima posible entre las muestras negativas y positivas. Esta condición óptima se obtiene de minimizar la norma Euclidiana del vector de pesos w.

3.2.2 Hiperplano óptimo para patrones linealmente no separables

Linealmente no separable significa ninguna de las ecuaciones lineales existe en el espacio con la misma dimensión que la del vector de entrenamiento. [57] En otras palabras, dada una muestra de datos de entrenamiento, no es posible construir un hiperplano de separación sin encontrar errores de clasificación. Sin embargo, nos gustaría encontrar un hiperplano óptimo que minimiza la probabilidad de error de clasificación, como media de la muestra de entrenamiento.
Se dice que el margen de separación entre clases es blando cuando un punto de los datos (x_i, d_i) no cumple la siguiente condición de la siguiente ecuación (25):

$$\boldsymbol{d_i(w^T * x_i + b) \geq 1} \qquad \boldsymbol{para\ i = 1, 2, \dots, N} \dots \boldsymbol{(25)}$$

Esta violación puede surgir de dos maneras:

a) Los puntos (x_i, d_i) , los cuales pertenecen a la clase α_1, representada por un cuadrado, caen dentro de la región de separación, y en el lado correcto de la superficie de decisión, cómo se ilustra en la figura 15.a.

b) Los puntos (x_i, d_i), los cuales pertenecen a la clase α_2, representada por un circulo, caen en el lado incorrecto de la superficie de decisión, cómo se ilustra en la figura 15.b.

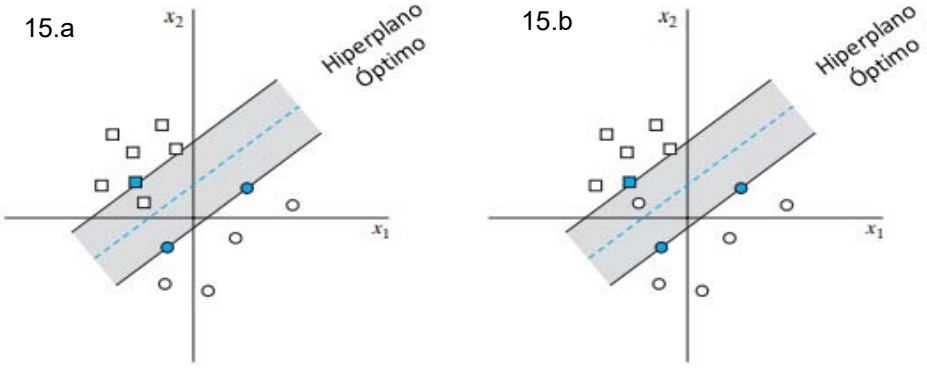

Figura 15. Gráficas que no cumplen con la ecuación (25)

Para establecer el escenario para un tratamiento formal de puntos linealmente no separables, se introduce un nuevo conjunto de variables escalares no negativos, $\{\varepsilon_i\}_{i=1}^{N}$ dentro de la definición del hiperplano de separación (es decir de la superficie de decisión), como sigue:

$$\boldsymbol{d_i(w^T * x_i + b) \geq 1 - \varepsilon_i \quad i = 1, 2, \ldots, N \ldots (26)}$$

Las ε_i son llamadas variables de holgura; las cuales miden la desviación de un punto de la condición ideal de patrón de separabilidad. Para $0 < \varepsilon_i \leq 1$, el punto cae dentro de la región de separación, pero en el lado correcto de la superficie de decisión, como se ilustra en la Fig. 15.a. Para $\varepsilon_i > 1$, el punto cae en el lado incorrecto de la superficie de decisión, cómo se ilustra en la figura 15.b. Los vectores de soporte son aquellos puntos particulares que satisfacen la ecuación (26) precisamente, incluso si $\varepsilon_i = 0$. Además, no puede haber vectores de soporte que satisfagan la condición $\varepsilon_i = 0$. Notar que si una muestra con $\varepsilon_i > 0$ se deja fuera de las muestras de entrenamiento, la superficie de decisión cambiará. Los vectores de soporte se definen entonces exactamente de la misma manera para ambos casos linealmente separables y no separables.
El objetivo es encontrar un hiperplano de separación por el cual el error de clasificación promediado sobre las muestras de entrenamiento es minimizado. Nosotros podemos hacer esto al minimizar la función

$$\phi(\varepsilon) = \sum_{i=1}^{N} I(\varepsilon_i - 1)$$

Con respecto al vector de pesos w, sujeto a la restricción se describe en la ecuación (26) y la restricción en la función $\|w\|^2$. La función $I(\varepsilon)$ es una función indicadora definida por:

$$I(\varepsilon) = \begin{cases} 0 & si\ \varepsilon \leq 0 \\ 1 & si\ \varepsilon > 0 \end{cases}$$

Desafortunadamente, la minimización de $\phi(\varepsilon)$ con respecto a w es una optimización convexa el cual es un problema NP completo. Para hacer el problema de optimización matemáticamente tratable, aproximamos la función $\phi(\varepsilon)$ al escribir:

$$\phi(\varepsilon) = \sum_{i=1}^{N} \varepsilon_i$$

Además, simplificamos la complejidad del algoritmo al formular la función para ser maximizada con respecto al vector de pesos de la siguiente forma:

$$\boldsymbol{\phi(w,\varepsilon) = \frac{1}{2} w^T * w + C * \sum_{i=1}^{N} \varepsilon_i} \ \ldots (27)$$

Al igual que antes, al reducir al mínimo el primer término de la ecuación (27) está relacionado con las máquinas de soporte vectorial. Por otro lado, el segundo término $\sum \varepsilon_i$, se trata de un límite superior en el número de errores de prueba.

El parámetro C controla el equilibrio entre la complejidad del algoritmo y el número de puntos linealmente no separables; por lo tanto, puede ser vista como el recíproco de un parámetro comúnmente referido como el parámetro de “regularización". Cuando el parámetro C se le asigna un valor grande, la implicación es que el diseñador de la máquina de soporte vectorial tiene una alta confianza en la calidad de la muestra de entrenamiento φ. Análogamente, cuando C se le asigna un valor pequeño, la muestra de entrenamiento φ se considera que es ruidoso, y por lo tanto se debe hacer menos hincapié en él.

En cualquier caso, el parámetro C tiene que ser seleccionado por el usuario. Se puede determinar experimentalmente mediante el uso estándar de una muestra de entrenamiento (método de validación), que es una forma cruda de re muestreo.

En cualquier caso, la función $\phi(w,\varepsilon)$ es optimizada con respecto a w y $\{\varepsilon_i\}_{i=1}^{N}$, sujeto a la restricción descrita en la ecuación (26), y $\varepsilon_i \geq 0$. De este modo, la norma de w se trata como una cantidad que se minimiza de forma conjunta con respecto a los puntos no separables y no como una restricción impuesta por la minimización de la cantidad de puntos no separables.

El problema de optimización para los patrones linealmente no separables incluye al problema de optimización para patrones linealmente separables como un caso especial. Específicamente al establecer $\varepsilon_i = 0$ para toda i en las ecuaciones (26) y (27) las reduce a la forma correspondiente para el caso linealmente separable. Podemos ahora formalmente situar el problema principal del caso linealmente no separado como sigue:

Dada la muestra de entrenamiento $\{(x_i, d_i)\}_{i=1}^{N}$, encontrar los valores óptimos del vector de pesos w y umbral b tal que satisfagan la siguiente restricción:

$$d_i(w^T * x_i + b) \geq 1 - \varepsilon_i \qquad para\ i = 1,2,\ldots,N$$
$$\varepsilon_i \geq 0 \qquad para\ toda\ i$$

Y tal que el vector de pesos w y las variables de holgura ε_i minimicen el costo de la función.

$$\phi(w,\varepsilon) = \frac{1}{2} w^T * w + C * \sum_{i=1}^{N} \varepsilon_i$$

Dónde C es un parámetro positivo especificado por el usuario.

Usando el método de multiplicadores de Lagrange y procediendo de forma similar a la descrita en el hiperplano para patrones linealmente separables, podemos formular el problema dual para patrones no separables como sigue:

Dada la muestra de entrenamiento $\{(x_i, d_i)\}_{i=1}^{N}$, encontrar los multiplicadores de Lagrange $\{(\alpha_i)\}_{i=1}^{N}$que maximicen la función objetivo:

$$\boldsymbol{Q(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j d_i \, d_j x_i^T x_j}$$

Sujeto a las restricciones:

$$\sum_{i=1}^{N} \alpha_i d_i = 0$$

$$0 \leq \alpha_i \leq C \qquad para\ i = 1,2, \dots, N$$

Notar que ni las variables de holgura ε_i, ni sus propios multiplicadores de Lagrange aparecen en el problema dual. El problema dual para el caso de los patrones no separables, es similar al caso simple de los patrones linealmente separables, a excepción de una menor, pero importante diferencia. La función objetivo $Q(\alpha)$ se maximiza en ambos casos. El caso no separable difiere del separable en que la restricción $\alpha_i \geq 0$ es remplazada con una restricción más rigurosa $0 \leq \alpha_i \leq C$. Exceptuando esta modificación, la optimización restringida para el caso de los no separables y los cálculos de los valores óptimos del vector de pesos w y el umbral b se procede en la misma forma que el caso de los linealmente separables. Notar también que las máquinas de soporte son definidas de la misma forma para ambos casos.

3.2.3 Máquinas de Soporte Vectorial Vistas como una Máquina de Kernel

Kernel de Producto Punto

Sea x un vector extraído del espacio de entrada de dimensión m_0. Sea $\{\varphi_j(x)\}_{j=1}^{\infty}$un conjunto de funciones no lineales que, entre ellos, transforman el espacio de entrada de dimensión m_0 a un espacio de dimensionalidad infinita. Dada esta transformación, nosotros podemos definir un hiperplano actuando como la superficie de decisión de acuerdo a la fórmula:

$$\sum_{j=1}^{\infty} w_j \, \varphi_j(x) = 0 \ldots (28)$$

Dónde $\{w_j\}_{j=1}^{\infty}$denota un conjunto infinitamente grande de pesos que transforman el espacio característico al espacio de salida. Esto es en el espacio de salida donde la decisión es tomada de si el vector de entrada x pertenece a una de las posibles clases, positiva o negativa. Por conveniencia de presentación se establece el umbral a cero en la ecuación (28). Usando la notación matricial podemos reescribir la ecuación en su forma compacta:

$$w^T \phi(x) = 0 \ldots (29)$$

Dónde $\phi(x)$ es el vector caracterìstico y w es el vector de pesos correspondiente.

Análogamente a “la separabilidad de los patrones transformados” en el espacio característico. En ese sentido, podemos adoptar la ecuación

$$w = \sum_{i=1}^{N_s} \alpha_i d_i \phi(x_i) \ \ldots (30)$$

Dónde el vector característico es expresado como:

$$\phi(x_i) = [\varphi_1(x_i), \varphi_2(x_i), \dots]^T \dots (31)$$

Y N_s es el número de vectores de soporte. Luego entonces, al sustituir la ecuación (29) en (30), podemos expresar la superficie de decisión en el espacio de salida como:

$$\sum_{i=1}^{N_s} \alpha_i d_i \phi^T(x_i)\phi(x) = 0 \dots (32)$$

Ahora vemos que el término escalar $\phi^T(x_i)\phi(x)$ de la ecuación (32) representa un producto punto. De acuerdo con ello, al dejar este término de producto punto ser escrito como escalar

$$k(x, x_i) = \phi^T(x_i)\phi(x)$$
$$= \sum_{j=1}^{\infty} \varphi_1(x_i), \varphi_2(x) \qquad i = 1,2, \dots, N_s$$

Correspondientemente, podemos expresar la superficie de decisión óptima (hiperplano) en el espacio de salida como:

$$\sum_{i=1}^{N_s} \alpha_i d_i k(x, x_i) = 0 \dots (33)$$

La función $k(x, x_i)$ es llamada el kernel de producto punto, o simplemente el kernel, el cual es formalmente definido como: una función que calcula el producto punto de las imágenes producidas en el espacio característico bajo la incrustación ϕ de dos puntos en el espacio característico. Siguiendo la definición de kernel, podemos decir que tiene dos propiedades básicas:

Propiedad 1: La función $k(x, x_i)$ es simétrica en el centro del punto x_i, esto es:

$$k(x, x_i) = k(x_i, x)\ para\ toda\ x_i$$

Además de alcanzar su valor máximo en $x = x_i$, notar sin embargo que no existe un máximo necesario; por ejemplo, $k(x, x_i) = x^T x_i$ dónde el kernel no tiene un máximo.

Propiedad 2: El volumen total bajo la superficie de la función $k(x, x_i)$ es una constante.

Si el kernel $k(x, x_i)$ se escala apropiadamente para hacer que la constante bajo propiedad 2 sea igual a la unidad, entonces tendrá propiedades similares a las de la función de densidad de probabilidad de una variable aleatoria.

Examinando la ecuación (33), ahora haremos dos importantes observaciones:

1. En cuanto a la clasificación de patrones en el espacio de salida, al especificar el kernel $k(x, x_i)$ es suficiente; en otras palabras, nunca necesitamos explícitamente calcular el vector de pesos w_o; es por esta razón que la aplicación de la ecuación (32) es comúnmente referida como el truco del kernel.
2. A pesar de que hemos supuesto que el espacio característico podría ser de dimensión infinita, la ecuación lineal (33), definiendo el hiperplano óptimo, consiste de un número finito de términos que es igual al número de patrones de entrenamiento usados en el clasificador.

A través de la observación 1, podemos decir que la máquina de soporte vectorial también se denomina una máquina de kernel. Para la clasificación de patrones, el algoritmo está parametrizado por un vector N-dimensional cuyo término i-ésimo se define por el producto $\alpha_i d_i \; para \; i = 1,2, \dots, N$. Podemos ver a $k(x, x_i)$ como el ij-ésimo elemento de la matriz simétrica NxN.

$$\boldsymbol{K} = \{\boldsymbol{k}(\boldsymbol{x_i}, \boldsymbol{x_j})\}_{\boldsymbol{i,j=1}}^{\boldsymbol{N}} \dots (\mathbf{34})$$

La matriz K es una matriz no negativa llamada la matriz kernel; que también puede ser encontrada en la literatura como Gram. Es no definida negativa o definida positiva cuando satisface la condición.

$$\boldsymbol{a^T K a \geq 0}$$

Para cualquier vector $\boldsymbol{a}$ en los reales cuya dimensión es compatible con la de $\boldsymbol{K}$.

Teorema de Mercer

La expansión de la ecuación (32) para el Kernel simétrico $k(x, x_i)$ es un caso importante del Teorema de Mercer, el cual dice lo siguiente:

Sea $k(x, x')$ un kernel continuo simétrico que está definido en el intervalo cerrado $a \leq x \leq b$, y lo mismo para x'. El kernel $k(x, x')$ puede ser expandido en las series:

$$k(x, x') = \sum_{i=1}^{\infty} \lambda_i \varphi_i(x), \varphi_i(x') \; \dots (35)$$

Con coeficientes positivos $\lambda_i \; > 0 \; para \; toda \; i.$ Para que esta expansión sea válida y para que converjan absoluta y uniformemente, es necesario y suficiente que la condición

$$\int_b^a \int_b^a k(x, x') \, \psi(x) \psi(x') dx dx' \geq 0 \dots (36)$$

Se mantenga para todo $\psi(\cdot)$, por lo cual tenemos

$$\int_b^a \psi^2(x) dx < \infty \dots (37)$$

Dónde a y b son las constantes de integración.

Las características $\varphi_i(x)$ son llamadas eigen funciones de la expansión, y los números λ_i son llamados eigen valores. El hecho de que todos los eigen valores sean positivos significa que el kernel $k(x, x')$ es definido positivo. Esta propiedad, en cambio significa que tenemos un problema complejo que puede ser resuelto de forma eficiente para el vector de pesos. Notar sin embargo que el teorema de Mercer nos dice que sólo si un kernel candidato es actualmente un kernel de producto punto en algún espacio y entonces admisible para el uso de una máquina de soporte vectorial. Sin embargo, el teorema de Mercer es importante porque pone un límite en el número de kernels admisibles. Tomar en cuenta también que la expansión de la ecuación (32) es un caso especial del teorema de Mercer, ya que todos los valores propios de esta expansión son la unidad. Es por esta razón por la que un kernel de producto punto, también se conoce como un núcleo de Mercer.

3.2.4 Ejemplos de SVM

El requerimiento en el kernel $k(x, x_i)$ es satisfacer el teorema de Mercer. Dentro de este requerimiento hay cierta libertad en cómo se escoge el kernel. En la Tabla 6.1 asumimos los kernels para tres tipos comunes de máquinas de soporte vectorial, kernel de aprendizaje polinomial, redes de funciones de base radial, y perceptrones de dos capas.

Tipo de Máquina de Soporte Vectorial	**Kernel Mercer** $\boldsymbol{k(x, x_i), i = 1, 2, \ldots, N}$	**Comentarios**
Máquina de Aprendizaje Polinomial	$(x^T * x_i + 1)^p$	La potencia p es especificada a priori por el usuario.
Redes de Funciones de Base Radial	$exp\left(-\frac{1}{2\sigma^2} * \|x - x_i\|^2\right)$	El tamaño de σ^2, común a todos los kernels, es especificado a priori por el usuario.
Perceptron de dos capas	$\tanh(\beta_0 x^T * x_i + \beta_1)$	El teorema de Mercer se cumple solo para algunos valores de β_0 y β_1

Tabla 2. Ejemplos de Kernel

CAPÍTULO IV: DETECCIÓN DE MALWARE, ENFOQUE BASADO EN LOS CONTADORES DE RENDIMIENTO DE WINDOWS Y UN CLASIFICADOR BAYESIANO

«Conoce a tu enemigo y conócete a ti mismo
y en cien batallas nunca estarás en peligro.»
Sun Tzu

Resumen

En el primer experimento obtuvimos los valores de los contadores más relacionados con la presencia de malware, al inducir actividad maliciosa y utilizando un clasificador Bayesiano. Para ello se utilizaron dos variantes de dos familias de Gusanos de red: Net-Worm.Win32.Kolab y Net-Worm.Win32.Kido además de Whiteware (software benigno que sirvió para diferenciar su comportamiento del malware) contenido en el directorio de System32. Las muestras fueron ejecutadas con la SandBox y por cada una se obtuvo los contadores de rendimiento cada segundo durante tres minutos. Después de obtener 494 reportes se obtuvo su promedio en los tres minutos de cada contador de cada reporte, como parte del pre procesamiento de los datos y debido a que estos presentaban un comportamiento continuo se empleó la técnica de Coarse Grain para poder hacerlos discretos. Finalmente, se generó el modelo de Score utilizando un 70% de los datos como entrenamiento y su complemento fue utilizado para probar el modelo, obteniendo una exactitud del 90%.

4.1 Contadores de Rendimiento de Windows

Los contadores de rendimiento son segmentos de código que monitorean, miden o cuentan eventos en el software, lo cual nos permite ver patrones a partir desde una perspectiva general. Están registrados en el sistema operativo durante la instalación del software, lo que permite verlos a cualquier persona con los permisos adecuados. Este concepto no es muy diferente de la gestión de un restaurante. Tomando como ejemplo uno que intente brindar su servicio 24/7, pero que es poco práctico e incluso imposible saber todo lo que está pasando. Al colocar monitores en partes clave del restaurante tales como el número de personas por hora, el número de pedidos en cola, la cantidad de los suministros de alimentos, y así sucesivamente, el administrador puede dar un paso atrás y puede identificar problemas, ver patrones emergentes y predecir las tendencias futuras.
Cuando un desarrollador escribe el software, puede ser difícil medir el rendimiento del software cuando está siendo escrito y cuando se limita a la estación de trabajo de un desarrollador. El desarrollador sin duda puede recorrer el código con un depurador, pero esto induce pausas frecuentes que no son parte del flujo normal de ejecución. Por otra

parte, una vez que el software está en la producción, el desarrollador tiene muy pocas formas de monitorear las rutas de código tomadas por el software. Por lo tanto, los contadores de rendimiento pueden ayudar a medir las piezas clave de software.
Al igual que todo el software, la confianza depende de la calidad de código y factores ambientales. En la mayoría de los casos, los contadores de rendimiento son muy fiables una vez que se entienda qué y cómo se están midiendo. Dicho esto, incluso los contadores de rendimiento mejor escritos pueden llegar a ser ridículamente inexactos por algo tan simple como un problema de reloj con el procesador.
Además, la virtualización de computadoras puede sesgar las mediciones de los contadores relacionados con el procesador, que no necesariamente lo causa un código mal ejecutado, pero si por la forma en que se programan entre el equipo virtual, el hipervisor, y el hardware. [64]

4.1.1 Rutas de los contadores

Una ruta de contador es una línea de texto que define uno o más instancias de contador. Es similar a una ruta de la convención universal de nombres (\\servidor\contribución) pero tiene una sintaxis ligeramente diferente. La ruta puede incluir un nombre de equipo como el origen de los datos del contador. Si se omite la parte de la computadora del registro, ejemplo: "\Procesador(_Total)\% Tiempo de Procesador", el cual corresponde al equipo local. Un asterisco (*) es usado como comodín para instancias de contadores (ejemplo: "\Procesador (*)\% Tiempo de Procesador"), para nombres de contadores (ejemplo: "\Procesador(_Total)*"), o ambos (ejemplo: \Procesador(*)*).

La sintaxis de una ruta de contador en forma general es:

\\Equipo\Objeto(Instancia)\Contador

Ejemplo:

\\Web01\Procesador(0)\% Tiempo de Procesador

Dónde:
Equipo: Es el equipo donde existe el contador. Si no se especifica una computadora, se asume que es la computadora local.
Objeto: Una agrupación lógica de contadores como: el Procesador y la Memoria.
Contador: El nombre del contador como: %Tiempo del Procesador. Un asterisco (*) indica que se engloban todos los contadores.
Instancia: Una instancia específica de un contador en el equipo como: _Total o 0. Un asterisco (*) indica que se engloban todas las instancias.

4.1.2 Características de los contadores

Los valores de los contadores son del tipo doble sin signo, lo cual es un valor decimal que no puede contener números negativos.

Cada instancia de un contador debe tener un nombre único. La herramienta de Monitor de Rendimiento trata con esto al agregar un sufijo de #x a cada instancia de contador del mismo tipo, donde x es el número más bajo disponible a partir de 1. El objeto de proceso a menudo tiene varias instancias con el mismo nombre y comúnmente se utiliza esta convención de nombres. Por ejemplo, si hay tres instancias de svchost.exe, entonces se tendrán svchost, svchost#1, y svchost#2. Si el proceso asignado a la instancia de svchost#1 termina y un nuevo svchost comienza, entonces se le asignará svchost#1 incluso cuando svchost#2 este activo. Esta técnica minimiza el número de instancias de contadores. El

sufijo no tiene nada que ver con el identificador de proceso PID, pero sirve para hacer un identificador único.

Algunas instancias de contadores tienen propósitos especiales. Las siguientes instancias especiales de contadores pueden encontrarse en Windows y Windows Server.

a) <Todas las instancias>: Este es un mecanismo de interfaz de usuario representando todas las instancias del objeto de contador seleccionado.
b) _Total: Esta instancia es encontrada en muchos de los objetos de contadores principales del sistema operativo. Por ejemplo, representa el total de todas las instancias del contador “Transferencias Disco Lógico/seg”, pero representa un promedio de todas las instancias del contador “Procesador % Tiempo de Procesador”.
c) _Global_: Esta instancia es comúnmente encontrada en los objetos de contadores relacionados con .NET y representa la suma de todas las instancias de contadores del contador respectivo.
d) PID en Procesos: Windows XP, Windows Server 2003, el cual cambia el comportamiento del Perfmon para desplegar las instancias objeto del proceso con el nombre del proceso y su ID desplegado como <NombreProceso>_<IDProceso>.

4.2 Diseño del Experimento

En el experimento se utilizaron dos muestras de dos familias de malware obtenidas de malwr.com [65]. Dichas familias corresponden a las siguientes:

1. Net-Worm.Win32.Kolab: es un gusano que funciona como un bot de IRC, lo que permite al atacante utilizar el ordenador de la víctima como un proxy o como una herramienta para la realización de ataques de denegación de servicio (DoS). Kolab también puede robar información del ordenador infectado, descargar malware adicional, y dar al atacante el control parcial de la computadora. [66]

2. Net-Worm.Win32.Kido: es un gusano que intenta atacar a los hosts de red a través de los puertos TCP 445 o 139, usando la vulnerabilidad de MS Windows MS08-067. El malware intenta acceder a sitios web con el fin de conocer la dirección IP externa del host infectado. [67]

Las muestras de malware fueron ejecutadas a través de la Sandbox Cuckoo (instalada en Ubuntu 14.04 de 64bits) por tres minutos.

Cada máquina virtual de Windows 7 después de cargar su estado de Snapshot, ejecuta un script en Powershell el cual obtiene los valores de los contadores de rendimiento en un archivo .csv correspondiente a una muestra por segundo durante los tres minutos que corresponden a nuestro tiempo de monitoreo. El esquema experimental puede verse en la Figura 16.

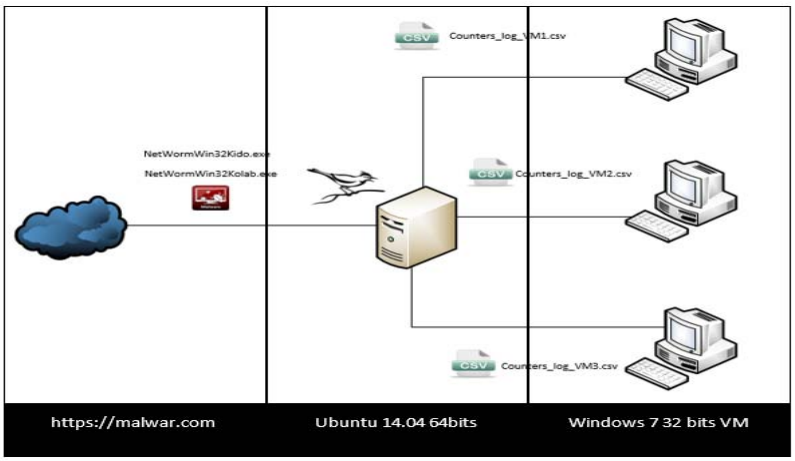

Figura 16. Diagrama de arquitectura de análisis de malware

El proceso de ejecución fue repetido setenta y cinco veces por muestra de malware y por otro lado en el whiteware (malware benigno) se utilizaron 300 de los programas ubicados en la carpeta de System32 teniendo un total de 600 ejecutables, de los cuales se utilizaron 599 para la generación del modelo debido a que el resto no obtenía el mismo número de campos en los archivos de csv.

4.2.1 Pre-procesamiento de los datos

Después de obtener los reportes correspondientes en formato separado por comas con los 180 registros, se calculó el promedio de cada campo a través de un script en bash, y debido a que los valores de los contadores de rendimiento en general presentan una tendencia continua se utilizó una técnica llamada Coarse Grain, la cual nos permite discretizar los valores de la muestra, en nuestro caso para llevarlo a cabo se realizaron los siguientes pasos:

1. Obtener el valor máximo y mínimo para cada variable
2. Obtener diez intervalos iguales entre los valores del paso anterior.
3. Calcular el rango correspondiente de cada valor para cada variable.

Después de aplicar la técnica de coarse grain resultó un archivo con 600 registros y 311 campos, el cual corresponde a nuestro conjunto de datos, del cual se deriva el conjunto de entrenamiento y el conjunto de prueba con un porcentaje de datos del 70% y 30% correspondientemente, seleccionando los datos de cada conjunto de manera aleatoria. Posteriormente al generar el modelo se utilizó un script en bash que incorpora el clasificador bayesiano.

4.3 Clasificador Bayesiano y Modelo de Contadores de Rendimiento de Windows

Un clasificador muy práctico basado en la teoría de Bayes es el clasificador Naive Bayes. A pesar de su simplicidad su rendimiento ha demostrado que es comparable a la de técnicas más sofisticadas, tales como las redes neuronales y aprendizaje por árbol de decisión [55]. De un total de 105 categorías de contadores de rendimiento, en nuestro escenario, dónde utilizamos Windows 7 Ultimate Service Pack 1, con la versión 1 de Powershell, en virtud de que algunas variables necesariamente requieren de Software extra como SQL Server,

Active Directory, etc. Además de que algunas categorías no arrojaban resultados en el reporte de contadores, se determinaron como conjunto base el siguiente conjunto de objetos con todas sus instancias:

1. Archivo de paginación
2. Ciclos de Actividad de Red por Procesador
3. Disco Físico
4. Disco Lógico
5. ICMP
6. Información del Procesador
7. Interfaz de red
8. IPv4
9. IPv6
10. Medidor de Energía
11. Memoria
12. Objetos de Trabajo
13. Objetos
14. Procesador
15. Sistema
16. TCPv4
17. UDPv4
18. UDPv6

Por cada objeto de la lista anterior hay diferentes contadores asociados, los cuales en suma constan de 311 valores para cada variable al monitorear el sistema. El conjunto de variables se puede representar por el vector $X = < x_1, x_2, x_3, \dots, x_{311} >$, el cual al tener 600 valores asociados a cada elemento puede representarse por una matriz, dónde para cada valor de x tenemos 600 valores correspondientes a diferentes instancias del problema, debido a que cada valor representa el comportamiento promedio durante tres minutos de cada variable, y en virtud de que los datos de entrenamiento constituyen el 70% de las muestras, entonces la matriz se representa de la siguiente forma:

$$X = \begin{pmatrix} \bar{x}_{1,1} & \cdots & \bar{x}_{1,311} \\ \vdots & \ddots & \vdots \\ \bar{x}_{420,1} & \cdots & \bar{x}_{420,311} \end{pmatrix}$$

Con el fin de determinar las variables con sus valores correspondientes, asociados con la presencia de malware se utilizó la ecuación de ε, vista en el capítulo 3, correspondiente a:

$$\varepsilon = \frac{N\bar{x}_i\big(P(c|\bar{x}_i) - P(c)\big)}{\sqrt{(N\bar{x}_i * P(c))(1 - P(c))}}, \bar{x}_i \in X$$

Dónde la variable c define la clase, en nuestro caso correspondiente a la presencia de malware y $\bar{x}_i$ es el valor característico a considerar. Esta es una prueba binomial con la hipótesis nula de que la presencia de malware no está relacionada con la variable $\bar{x}_i$ y entonces aparece con una frecuencia $P(c)$. El resultado de esta prueba binomial puede ser interpretada en términos de la prueba de hipótesis estándar.

Si $|\varepsilon| > 1.96$, entonces en el límite dónde la distribución binomial puede ser aproximada a la distribución normal, podemos rechazar la hipótesis nula con un grado de confianza mayor a 95% y entonces la variable $\bar{x}_i$ está relacionada con ella, de esta manera se dice que la variable está relacionada con la presencia de malware. Con el fin de obtener la contribución de probabilidad para todos los valores de X usamos la aproximación de Naive Bayes, dónde $P(X|c) = \prod_{i=1}^{N} P(\bar{x}_i|C) * P(C)$ y similarmente para la probabilidad del complemento de c (la no clase).

En este caso, nosotros podemos usar la siguiente función de Score como una medida de probabilidad indicando que el sistema ha sido infectado con malware:

$$S(c|\bar{x}) = \sum_{i=1}^{N} \ln\left(\frac{(P(\bar{x}_i|c)}{(P(\bar{x}_i|c')}\right) + \ln\left(\frac{P(c)}{P(c')}\right), \bar{x}_i \in X$$

Dónde para cada variable, la contribución de probabilidad fue calculada con la siguiente expresión:

$$S(c|\bar{x}_i) = \ln\left(\frac{P(\bar{x}_i|c)}{P(\bar{x}_i|c')}\right), \bar{x}_i \in X$$

Mientras mayor sea la suma de valores de $S(c|\bar{x}_i)$ en cada variable, dicho registro está más relacionado con la clase y análogamente mientras menor sea el valor de $S(c|\bar{x}_i)$ entonces dicha instancia está más relacionada con la no clase.

4.4 Variables predictores y la función de Score

Considerando que para un valor teórico de $\varepsilon = 1.96$, se encuentra el 95% de la clase, y análogamente $\varepsilon = -1.96$, para el 95% de la no clase; procedemos a depurar el modelo eliminando todos los valores dónde ε se encuentren en el intervalo abierto: $-1.96 < \varepsilon < 1.96$. La siguiente tabla muestra las primeras 10 variables predictoras ordenadas por su valor de ε:

#	Variable	Valor	ε	Intervalo
1	*UDPv4\Datagramas sin puerto/s*	2	12.2278	$0.0637 \leq \bar{x}_i < 0.1274$
2	*TCPv4\Conexiones establecidas*	6	10.6755	$3.966 \leq \bar{x}_i < 4.373$
3	*Ciclos de actividad de red por procesador\Ciclos de indicación de recepción de NDIS por segundo*	2	8.0520	$10585646.88 \leq \bar{x}_i < 21111061.8$
4	*Ciclos de actividad de red por procesador\Ciclos DPC de interrupción por segundo*	2	7.9899	$27178627.2 \leq \bar{x}_i < 54271195.61$
5	*Información del Procesador\DPC en cola/s*	2	7.9274	$582.14 \leq \bar{x}_i < 1137.36$
6	*Interfaz de Red\Paquetes recibidos/s*	2	7.8746	$449.403 \leq \bar{x}_i < 898.292$
7	*Ciclos de actividad de red por procesador\Ciclos de indicación de recepción de pila por segundo*	2	7.8112	$9895084 \leq \bar{x}_i < 19733318$
8	*Ciclos de actividad de red por procesador\Ciclos de interrupción por segundo*	2	7.6031	$8936810 \leq \bar{x}_i < 17859354$
9	*IPv4\Datagramas recibidos descartados*	3	7.5562	$44.6 \leq \bar{x}_i < 65.40$
10	*Interfaz de Red\Paquetes de no unidifusión enviados/s*	10	7.2680	$1.37 \leq \bar{x}_i < 1.52$

Tabla 3. Primeras 10 variables predictores ordenadas por su valor ε

Retomando que como parte del pre procesamiento de los datos con el Coarse Grain, para cada variable se calcularon diez intervalos con una diferencia constante empleando como extremos el valor mayor y menor de cada una, luego de ordenarlos de manera ascendente se etiquetaron con valores de 1 a 10 dependiendo del intervalo al que pertenezcan, para

llevar a cabo dicha tarea se incluyeron tanto las muestras de Whiteware como las de Malware. A continuación, se muestra la descripción de las variables anteriores:

a) *UDPv4\Datagramas sin puerto/s*: es la velocidad a la que se recibieron datagramas UDP para los cuales no existía ninguna aplicación en el puerto de destino.
b) *TCPv4\Conexiones establecidas*: es el número de conexiones TCP para las cuales el estado actual es ESTABLISHED o CLOSE-WAIT.
c) *Ciclos de actividad de red por procesador\Ciclos de indicación de recepción de NDIS por segundo*: es la velocidad media, en ciclos por segundo, a la que NDIS procesó una llamada de indicación de recepción de una interfaz.
d) *Ciclos de actividad de red por procesador\Ciclos DPC de interrupción por segundo*: es la velocidad media, en ciclos por segundo, a la que NDIS (Network Driver Interface Specification API) procesó una llamada a procedimiento diferida (DPC) de una interfaz.
e) *Información del Procesador\DPC en cola/s*: es la velocidad promedio, en incidentes por segundo, a la que se agregaron llamadas a procedimiento diferidas (DPC) a la cola de DPC del procesador. Las DPC son interrupciones que se ejecutan con una prioridad inferior a la de las interrupciones estándar. Este contador mide la velocidad a la que las DPC se agregan a la cola, no el número de DPC en la cola.
f) *Interfaz de Red\Paquetes recibidos/s*: es la velocidad a la que se reciben paquetes en la interfaz de red.
g) *Ciclos de actividad de red por procesador\Ciclos de indicación de recepción de pila por segundo*: es la velocidad media, en ciclos por segundo, a la que la pila procesó una llamada de indicación de recepción de una interfaz.
h) *Ciclos de actividad de red por procesador\Ciclos de interrupción por segundo*: es la velocidad media, en ciclos por segundo, a la que NDIS procesó las interrupciones de hardware de una interfaz.
i) *IPv4\Datagramas recibidos descartados*: es el número de datagramas IP de entrada que se descartaron, aunque no se encontraron problemas que impedían su procesamiento (por ejemplo, por falta de espacio en el búfer). Este contador no incluye ninguno de los datagramas descartados cuando estaban en espera de ser re-ensamblados.
j) *Interfaz de Red\Paquetes de no unidifusión enviados/s*: es la velocidad a la que los protocolos de nivel superior solicitan la transmisión de paquetes a direcciones que no son de unidifusión (es decir, difusiones de subred o multidifusiones de subred). Esta velocidad incluye los paquetes que se descartaron o que no se enviaron.

Previo al proceso de entrenamiento, puede apreciarse que los datos más relacionados con la clase a través de su valor de ε corresponden al comportamiento del malware en red, y a este nivel de interpretación dónde las variables son físicas sugiere que el malware hace un reconocimiento en el equipo de los puertos que están abiertos como lo indica Net-Worm.Win32.Kido y/o intentan comunicarse con el atacante en virtud de que la muestra Net-Worm.Win32.Kolab tiene comportamiento de un bot, los cuales utilizan generalmente el protocolo UDP para llevar a cabo el “Command and Control”. Por otro lado, el número de conexiones TCP e IPv4 corresponde al comportamiento común de un gusano de red, los cuales una vez infectado el equipo y realizado el mapeo de red infectan los equipos de la red. Finalmente, como consecuencia de estas conexiones en las capas de red y transporte, en la capa física los contadores relacionados con la Interfaz de red y la información del

procesador relacionada con la red, también son involucrados en dicho comportamiento malicioso.

Posteriormente con los datos dónde $\varepsilon \leq -1.96$ ^ $\varepsilon \geq 1.96$, se calcularon los valores de $S(c|\bar{x}_i)$, como resultado de dicho proceso se obtuvo la siguiente tabla:

#	Variable	Valor	$S(c\|\bar{x}_i)$	Intervalo	$N(\bar{x}_i\|c)$
1	*UDPv4\Datagramas sin puerto/s*	2	2.7246	$0.06 \leq \bar{x}_i < 0.12$	153
2	*TCPv4\Errores de conexión*	7	2.1726	$56.62 \leq \bar{x}_i < 66.05$	41
3	*TCPv4\Errores de conexión*	4	2.0968	$28.31 \leq \bar{x}_i < 37.74$	37
4	*UDPv4\Datagramas enviados/s*	4	2.0764	$0.19 \leq \bar{x}_i < 0.25$	36
5	*Interfaz de Red\Paquetes de no unidifusión enviados/s*	2	2.0514	$0.15 \leq \bar{x}_i < 0.3$	46
6	*Información del Procesador\% de tiempo en prioridad*	2	2.0169	$7.82 \leq \bar{x}_i < 15.31$	51
7	*Ciclos de Actividad de Red por Procesador\Ciclos de interrupción por segundo*	2	2.0048	$8936810 \leq \bar{x}_i < 17859354$	58
8	*Objetos\Semáforos*	9	2.0048	$677.88 \leq \bar{x}_i < 692.25$	29
9	*UDPv4\Datagramas enviados/s*	9	2.0007	$0.5076 \leq \bar{x}_i < 0.571$	36
10	*UDPv4\Datagramas enviados/s*	10	1.9838	$0.571 \leq \bar{x}_i < 0.6345$	34

Tabla 4. Primeras 10 variables ordenadas por su valor $S(c|\bar{x})$

A continuación, se muestra la descripción de las variables anteriores (no definidas en la tabla 3):

- **a)** *TCPv4\Errores de conexión*: es el número de veces que las conexiones TCP realizaron una transición directa al estado CLOSED desde el estado SYN-SENT o SYN-RCVD, sumado al número de veces que las conexiones TCP realizaron una transición directa al estado LISTEN desde el estado SYN- RCVD.
- **b)** *Información del Procesador\% de tiempo en prioridad*: es el porcentaje de tiempo que emplea el procesador en ejecutar subprocesos que no son de baja prioridad. Para calcularlo se mide el porcentaje de tiempo que emplea el procesador en ejecutar subprocesos de baja prioridad o el subproceso inactivo y después se resta este valor de 100%. (Cada procesador tiene un subproceso inactivo en el que se acumula tiempo cuando no hay ningún otro subproceso preparado para ejecutarse.) Este contador muestra el porcentaje promedio de tiempo ocupado observado durante el intervalo de muestra sin incluir el trabajo de baja prioridad en segundo plano. Debe tenerse en cuenta que el cálculo de si el procesador está inactivo se realiza en un intervalo de muestreo interno del reloj del sistema. Por tanto, % de tiempo en prioridad puede infravalorar el uso del procesador, ya que es posible que el procesador emplee mucho tiempo en atender subprocesos entre el intervalo de muestreo del reloj del sistema. Las aplicaciones de temporizador basadas en cargas de trabajo son un ejemplo de aplicaciones que se pueden medir de forma inexacta, puesto que los temporizadores se señalan justo después de tomar la muestra.
- **c)** *Ciclos de Actividad de Red por Procesador\Ciclos de interrupción por segundo*: es la velocidad media, en ciclos por segundo, a la que NDIS procesó las interrupciones de hardware de una interfaz.

d) *Objetos\Semáforos*: es el número de semáforos existentes en el equipo en el momento de la recopilación de datos. Se trata de un recuento instantáneo y no de un promedio a lo largo de un intervalo de tiempo. Los subprocesos usan semáforos para obtener el acceso exclusivo a las estructuras de datos que comparten con otros subprocesos.

e) *UDPv4\Datagramas enviados/s*: es la velocidad a la que se envían datagramas UDP desde la entidad.

Cómo puede apreciarse en la tabla anterior después del entrenamiento, la variable *UDPv4\Datagramas sin puerto/s,* se mantiene como la más representativa ya que de los 420 registros, 259 están relacionados con la clase en los primeros diez valores de $S(c|\bar{x}_i)$ y es precisamente el whiteware quien posee todos los valores de 1 el cual va de 0 a 0.06. Además de lo ya mencionado en la tabla 3, la variable de *Objetos\Semáforos* aparece con uno de los valores más altos, y nos indica que las muestras tienen un mayor número de procesos y subprocesos respecto al whiteware.

4.5 Resultados

Después de generar el modelo con los datos de entrenamiento; es decir obtener el valor de $S(c|\bar{x}_i)$ para cada valor; posteriormente se sustituyeron en los datos de prueba. Si definimos cada elemento de los datos de prueba como $t_i = S(c|\bar{x}_i)$ para cada variable y asumiendo que el número de registros es igual a 180 tendremos el conjunto de datos de prueba con sus valores correspondientes de score definida de la siguiente forma:

$$T = \begin{pmatrix} t_{1,1} & \cdots & t_{311,1} \\ \vdots & \ddots & \vdots \\ t_{1,180} & \cdots & t_{311,180} \end{pmatrix}$$

Como resultado de la suma en cada renglón de la matriz T, tenemos el vector de scores $Sc = < Sc_1, Sc_2, Sc_3, \dots, Sc_{311} >$ dónde cada elemento se define de la siguiente forma:

$$Sc_j = \sum_{i=1}^{311} t_i = \sum_{i=1}^{311} S(c|\bar{x}_i)$$

Después de calcular los valores correspondientes con la ecuación anterior y ordenar los valores descendentemente respecto a su valor de Sc, se concluye la siguiente tabla:

j	Clase	Sc_j	j	Clase	Sc_j
1	1	90.9120	171	0	−17.8750
2	1	86.3634	172	0	−17.9103
3	1	84.6929	173	1	−18.7164
4	1	83.8129	174	0	−19.0676
5	1	78.4409	175	0	−19.1843
6	1	77.1692	176	0	−19.3879
7	1	76.6869	177	0	−19.7506
8	1	76.0579	178	0	−22.0367
9	1	73.4041	179	0	−24.0859
10	1	68.736	180	0	−24.3722

Tabla 5. Primeros y últimos 10 registros ordenados por su valor Sc_j

A través de la tabla anterior se muestra que en el caso de los 10 primeros valores de $\boldsymbol{Sc_j}$ existe una correspondencia de 100% con el valor de la clase, y de 90% respecto a los últimos 10 de la no clase. Idealmente el modelo separaría todos los valores de la clase y de la no clase tomando como referencia su valor de $\boldsymbol{Sc_j}$, sin embargo, para hallar el punto de corte óptimo del valor de $\boldsymbol{Sc_j}$ y con esto diferenciar dónde comienzan los valores de ambas, en primera instancia se calcularon los valores de $Sensibilidad$ y $1 - Especificidad$. Tal como se muestra en la siguiente tabla:

$\boldsymbol{j}$	$\boldsymbol{Sc_j}$	$\boldsymbol{Sensibilidad}$	$\boldsymbol{1 - Especificidad}$	$\boldsymbol{j}$	$\boldsymbol{Sc_j}$	$\boldsymbol{Sensibilidad}$	$\boldsymbol{1 - Especificidad}$
1	90.912	0.011	0	**171**	−17.875	0.988	0.915
2	86.363	0.023	0	**172**	−17.910	0.988	0.926
3	84.692	0.035	0	**173**	−18.716	1	0.926
4	83.812	0.047	0	**174**	−19.067	1	0.936
5	78.44	0.058	0	**175**	−19.184	1	0.947
6	77.169	0.07	0	**176**	−19.387	1	0.957
7	76.686	0.082	0	**177**	−19.75	1	0.968
8	76.057	0.094	0	**178**	−22.036	1	0.979
9	73.404	0.105	0	**179**	−24.085	1	0.989
10	68.736	0.117	0	**180**	−24.372	1	1

Tabla 6. Primeros y últimos 10 registros ordenados por su valor Sc_j

Dónde:
$\boldsymbol{j}$: La posición de $\boldsymbol{Sc}$ luego de ordenarlos valores ascendentemente
$\boldsymbol{Sc_j}$: El valor de $\boldsymbol{Sc}$ correspondiente a $\boldsymbol{j}$
$\boldsymbol{Sensibilidad}$: $\frac{\boldsymbol{TP}}{\boldsymbol{TP+FN}}$, corresponde a la probabilidad de predecir correctamente dónde existió presencia de malware.
$\boldsymbol{Especificidad}$: $\frac{\boldsymbol{TN}}{\boldsymbol{TN+FP}}$, corresponde a la probabilidad de predecir correctamente dónde no existió presencia de malware.
$\boldsymbol{TN}$: El número de Verdaderos Negativos corresponde a las predicciones correctas donde no existió presencia de malware.
$\boldsymbol{FN}$: El número de Falsos Negativos corresponde a las predicciones incorrectas donde no existió presencia de malware.
$\boldsymbol{TP}$: El número de Verdaderos Positivos corresponde a las predicciones correctas donde existió presencia de malware.
$\boldsymbol{FP}$: El número de Falsos Positivos corresponde a las predicciones incorrectas donde existió presencia de malware.

Es importante mencionar que para el cálculo de $Sensibilidad$ y $Especificidad$ de los puntos en la curva ROC se tomaron como umbral de clase cada uno de los valores de $\boldsymbol{Sc_j}$**.** Al graficar cada uno de los puntos y tomando como dominio los valores de $1 - Especificidad$ además de análogamente la $Sensibilidad$ en el eje de las abscisas tenemos la curva ROC. Considerando el criterio ROC01,el siguiente paso para obtener el punto de corte óptimo de $\boldsymbol{Sc_j}$, es obtener la distancia euclidiana de todos los puntos a $i(0,1)$, el cual representa el punto de un clasificador perfecto [68]. En la siguiente gráfica además de apreciar la curva ROC, se muestra el punto de corte óptimo $o(0.138,0.882)$, el cual representa el valor más cercano a $i(0,1)$.

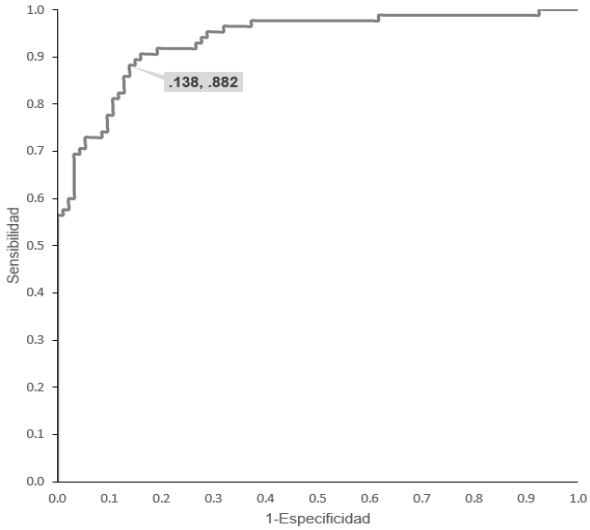

Figura 17. Curva ROC

Al calcular el área bajo la curva obtenemos $AUC = 0.935$, el cual sugiere que el modelo tiene un valor de predicción muy bueno, considerando que el clasificador ideal se define con $AUC = 1$. Posteriormente ubicamos que el valor de $Sc = -6.3233$ corresponde con el punto de corte óptimo. Retomando los valores de TP, TN, FP y FN correspondientes a ese valor de Sc, que pueden apreciarse en la siguiente tabla, procedemos a calcular otras métricas frecuentemente utilizadas en aprendizaje automatizado que nos permitan ponderar el rendimiento de nuestro modelo.

Matriz de Confusión			
		Predicho	
		Negativo	Positivo
Actual	Negativo	81	13
	Positivo	10	75

Tabla 7. Matriz de Confusión en el Punto de Corte Óptimo

$$\boldsymbol{Sensibilidad}: TPR = \frac{TP}{TP + FN} = 0.882$$

$$\boldsymbol{Precisión}: PPV = \frac{TP}{TP + FP} = 0.882352941$$

$$\boldsymbol{Especificidad}: SPC = \frac{TN}{FP + TN} = 0.862$$

$$\boldsymbol{Exactitud}: ACC = \frac{TP + TN}{P + N} = 0.87150838$$

$\boldsymbol{Coeficiente\ de\ Correlación\ de\ Mathews}$:

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} = 0.743218373$$

En primera instancia se muestra que la sensibilidad y especificidad son similares; esto significa que nuestro modelo predice en el mismo grado de confianza (a nivel decimal) tanto individuos de la clase como de la no clase, y dichos resultados sugieren una correlación con el coeficiente de correlación de Mathews el cual describe la relación entre los elementos de la clase y la no clase con base en la matriz de confusión a través de un valor máximo de

uno. Por otro lado, la exactitud o razón de predicciones correctas es buena considerando que el valor máximo es uno.

4.6 Comparación con otros algoritmos

Finalmente, para este experimento se comparó nuestra implementación del Naive Bayes* con la del software Weka, además de otros algoritmos de aprendizaje supervisado, con la finalidad de comparar la exactitud. Para llevar a cabo dicha tarea se utilizó la misma tabla de muestras, además del mismo porcentaje para el conjunto de datos de entrenamiento y de prueba, con los valores por default de Weka, después de realizar diez ejecuciones (variando los datos, pero manteniendo la relación de conjunto de entrenamiento en 70% y de prueba en 30%) de cada algoritmo los resultados se muestran en la siguiente tabla:

	Logit Boost	**Naive Bayes**	**Naive Bayes***	**Naive Bayes Multinomial**	**Random Forest**	**SVM RBF Kernel**	**SVM Poly Kernel**	**SVM Poly Kernel Normalized**
Exactitud Promedio	96	84.2	70.33	80.1	94.8	82.4	95.1	92.5
Exactitud Desviación Estándar	33.7	31.9	13.29	33.7	13.7	27.4	17.4	19.7

Tabla 8. Matriz de comparación de exactitud promedio $\overline{acc}$ con otros algoritmos y su desviación estándar

Como puede apreciarse en la tabla anterior, en la cual se obtienen dos medidas de la exactitud de cada algoritmo después de sus ejecuciones correspondientes, y nos permite saber por un lado cual es su tendencia central y a su vez que tanto varía ésta. En conclusión, el caso ideal sería que la exactitud promedio fuese de 100% y su desviación estándar de 0%.

Nuestra implementación del Naive Bayes ocupa el último lugar respecto a la exactitud promedio, sin embargo, es quien presenta una menor desviación estándar. Así mismo quien presenta una mejor relación de ambas variables es el algoritmo de Random Forest ocupando el segundo lugar exactitud y desviación estándar de la misma.

CAPÍTULO V: DETECCIÓN Y CLASIFICACIÓN DE MALWARE CON MODELO DE LENGUAJE

«But it must be recognized that the notion "probability of a sentence" is an entirely useless one, under any known interpretation of this term. »
Noam Chomsky

Resumen

En este experimento se utilizó la Sandbox Cuckoo para llevar a cabo el análisis de seis tipos de malware: Troyanos, Gusanos, Virus, Troyanos Espía, Puertas Traseras y Rootkits, además de un conjunto de Whiteware con el mismo número de muestras para cada tipo. Utilizando como fuente de información los n-gramas resultantes de las llamadas al sistema del tipo Win API, por los procesos ejecutados durante la ejecución de cada muestra y variando el tamaño de los n-gramas de uno a diez, se obtuvieron 10 diferentes bases de conocimiento correspondientes en cada modelo, a la relación de presencia de cada n-grama en cada muestra de malware, resultando diez tablas binarias con dicha información.

Esencialmente se usaron las bases de conocimiento para dos experimentos:

a) Clasificación de malware utilizando Máquinas de Soporte Vectorial (SVM) con kernel polinomial como algoritmo de aprendizaje supervisado y la variación del tamaño de n-gramas para saber cual tiene mayor exactitud acc. Dónde la mejor relación de exactitud promedio y su desviación estándar, fue generada con la base de conocimiento de 3gramas, con un valor de 75.5 y 1.26 correspondientemente. [69]

b) Minería de datos con el clasificador Bayesiano sobre los n-gramas de Win API con malware y whiteware para conocer que n-gramas están más relacionados con procesos maliciosos. Siguiendo la metodología, se realizaron diez iteraciones con diferentes algoritmos de clasificación, en la cual puede apreciarse que nuestra implementación de Naive Bayes obtuvo los mejores resultados de exactitud considerando todos los n-gramas y concretamente con el modelo de 3gramas obtuvo su mejor resultado con 99.31% y con una desviación estándar de 1.

5.1 Win API y Native API

La API de Windows es un amplio conjunto de funciones que rige la forma en que el malware interactúa con las bibliotecas de Microsoft. Gran parte de la API de Windows utiliza sus propios nombres para representar tipos de C. Normalmente no se utilizan tipos estándar de

C como *int*, *short*, y *unsigned int*. Así mismo usa la notación húngara para los identificadores de función de la API.
Los tres tipos principales:

a) *word*: representan enteros de 32 bits sin signo.
b) *dword:* representan enteros de 16 bits sin signo.
c) handle: una referencia a un objeto, son elementos que se han abierto o creado en el sistema operativo: una ventana, proceso, módulo, menú, archivo, etc. Los handles son como punteros en lo que se refiere a un objeto o ubicación en memoria en otro lugar. Sin embargo, a diferencia de los punteros, los handles no se pueden utilizar en las operaciones aritméticas, y no siempre representan la dirección del objeto.

La función *CreateWindowEx* contiene un ejemplo sencillo de un handle. Devuelve un HWND, que es un handle para una ventana. Cada vez que desee hacer algo con esa ventana, como la llamada DestroyWindow, se tiene que usar ese handle.

5.1.1 Funciones en el Sistema de Archivos

Una de las formas más comunes en que el malware interactúa con el sistema es mediante la creación o modificación de archivos y nombres de archivo distintos o cambios en los nombres de archivo existentes pueden ser buenos indicadores basados en host. Las principales funciones de la API en este para acceder al sistema de archivos son:

a) *CreateFile*: se usa para crear y abrir archivos. Puede abrir archivos existentes, pipes, streams, dispositivos de entrada y salida, además de crear nuevos archivos. El parámetro *dwCreationDisposition* controla si la función *CreateFile* crea un archivo nuevo o abre uno existente.
b) *ReadFile y WriteFile*: se utilizan para la lectura y la escritura a los archivos. Ambos operan en archivos como un stream. La primera vez que se llama a *ReadFile*, se leen los próximos bytes de un archivo; la próxima vez que se llame, lee los siguientes bytes después de los anteriories. Por ejemplo, si se abre un archivo y se hace la llamada *ReadFile* con un tamaño de 40, la próxima vez que la llame, se leerá comenzando con el cuadragésimo primer byte. Como se puede imaginar, sin embargo, ninguna función hace que sea especialmente fácil de saltar alrededor dentro de un archivo.
c) *CreateFileMapping* y *MapViewOfFile*: los mapeos de archivo son comúnmente utilizados por los creadores de malware, ya que permiten que un archivo se cargue en memoria y se manipula fácilmente. La función *CreateFileMapping* carga un archivo del disco en memoria. La función *MapViewOfFile* devuelve un puntero a la dirección base del mapeo, la cual se puede utilizar para acceder al archivo en memoria. El programa al llamar estas funciones puede usar el puntero retornado de *MapViewOfFile*.

5.1.2 Funciones Comunes en el Registro

El malware a menudo utiliza las funciones de registro con el fin de modificarlo para ejecutarse automáticamente cuando se inicia el sistema. Las siguientes son las funciones más comunes del registro:

a) *RegOpenKeyEx*: abre un registro para edición y consulta. Hay funciones que permiten consultar y editar una clave de registro sin necesidad de abrirlas primera, pero la mayoría de los programas utilizan *RegOpenKeyEx* de todos modos.
b) *RegSetValueEx*: añade un nuevo valor en el registro y establece sus datos.
c) *RegGetValue*: devuelve los datos para una entrada de valor en el registro.

5.1.3 APIs de Interconexión

El malware se basa con frecuencia en funciones de red para llevar a cabo su comportamiento malicioso. Para lo cual existen principalmente dos APIs de red:

- Compatibles Sockets Berkeley:

De las opciones de red de Windows, el malware utiliza comúnmente los sockets compatibles Berkeley, cuya funcionalidad es casi idéntica en los sistemas Windows y UNIX.
La funcionalidad de red de los sockets compatibles Berkeley en Windows se implementa en las bibliotecas de Winsock, principalmente en ws2_32.dll. Las funciones más comunes son las siguientes:

a) *socket*: crea un socket
b) *bind*: permite agregar un socket a un puerto en particular, antes de aceptar la llamada.
c) *listen*: indica que un socket estará escuchando las conexiones entrantes
d) *accept*: abre una conexión a un socket remoto y acepta la conexión
e) *connect*: abre una conexión a un socket remoto, el cual debe estar en espera de conexión
f) *recv*: recibe datos del socket remoto
g) *send*: envía datos del socket remoto

- La API WinINet

Además de la API de Winsock, hay una API de alto nivel llamada API WinINet, cuyas funciones se almacenan en Wininet.dll. Si un programa importa funciones de esta DLL, está usando APIs de red de alto nivel. El API WinINet implementa protocolos, como FTP y HTTP, en la capa de aplicación. El malware puede usarlo para conectarse de manera remota a un servidor y obtener instrucciones futuras para su ejecución. Sus principales funciones son las siguientes:

a) *InternetOpen*: se usa para iniciar una conexión a internet
b) *InternetOpenUrl*: se usa para conectar a una URL (la cual puede ser una página HTTP o un recurso FTP).
c) *InternetReadFile*: trabaja similar a la función *ReadFile*, permitiendo al programa leer los datos de un archivo descargado de internet.

5.1.4 Procesos

El malware también puede ejecutar código fuera del programa actual mediante la creación de un nuevo proceso o modificar uno existente. Un proceso es un programa que está siendo ejecutado por Windows. Cada proceso gestiona sus propios recursos, como los handles

abiertos y la memoria. Un proceso contiene uno o más hilos que se ejecutan por la CPU. Tradicionalmente, malware ha consistido de su propio proceso independiente, pero los nuevos programas maliciosos comúnmente ejecutan su código como parte de otro proceso. La función más comúnmente usada por el malware para crear un nuevo proceso es:

a) *CreateProcess*: Esta función tiene muchos parámetros, y la persona que llama tiene un gran control sobre la forma en que se creará. Por ejemplo, el malware podría llamar a esta función para crear un proceso para ejecutar su código malicioso, con el fin de eludir los firewalls basados en host y otros mecanismos de seguridad. O podría crear una instancia de Internet Explorer y luego usar ese programa para acceder a contenido malicioso. El malware utiliza comúnmente *CreateProcess* para crear un sencillo shell remoto con una sola llamada a la función. Uno de los parámetros a la función *CreateProcess*, es la estructura STARTUPINFO, la cual incluye un handle de la entrada estándar, salida estándar y salida de error estándar para un proceso. Un programa malicioso puede establecer estos valores a un socket, de modo que cuando el programa escribe en la salida estándar, en realidad está escribiendo en el socket, lo que permite a un atacante ejecutar un shell remoto sin correr alguna otra llamada distinta de *CreateProcess*.

5.1.5 Hilos

Los procesos son el contenedor para su ejecución, pero los hilos son aquellos que ejecuta el sistema operativo Windows. Los threads son secuencias independientes de instrucciones que son ejecutadas por la CPU sin esperar a otros hilos. Un proceso contiene uno o más hilos, que ejecutan parte del código dentro de un proceso. Los hilos dentro de todo un proceso de compartir el mismo espacio de memoria, pero cada uno tiene sus propios registros del procesador y la pila. Para crear un nuevo hilo se usa la siguiente función:

a) *CreateThread*: cuando se llama la función se específica una dirección de inicio, que a menudo se llama la función de arranque. La ejecución comienza en la dirección de inicio y continúa hasta que la función retorne, aunque la función no necesita retornar, y el hilo se puede ejecutar hasta que finalice el proceso. Al analizar código que llama *CreateThread*, se tendrá que analizar la función de arranque además de analizar el resto del código de la función que llama *CreateThread*.

5.1.6 Servicios

Otra forma en que el software malicioso pueda ejecutar código adicional es instalándolo como un servicio. Windows permite que las tareas se ejecuten sin sus propios procesos o hilos mediante el uso de los servicios que se ejecutan como aplicaciones en segundo plano; el código es agendado y dirigido por el Windows service manager sin intervención del usuario. El uso de servicios tiene muchas ventajas para el creador de malware. Una de ellas es que los servicios se ejecutan normalmente como SYSTEM u otra cuenta privilegiada. Esto no es una vulnerabilidad porque necesita acceso administrativo para instalar un servicio, pero es conveniente para los creadores de malware, debido a que la cuenta SYSTEM tiene más acceso que las cuentas de administrador o de usuario. Las funciones principales son las siguientes:

a) *OpenSCManager*: devuelve un identificador para el administrador de control de servicio, que se utiliza para todas las llamadas a funciones relacionadas con los

servicios posteriores. Todo el código que interactuará con los servicios llamará a esta función.

b) *CreateService*: añade un nuevo servicio para el administrador de control de servicios, y permite especificar si el servicio se iniciará automáticamente al arrancar el sistema o se debe iniciar manualmente.
c) *StartService*: Inicia un servicio, y sólo se utiliza si el servicio está configurado para iniciarse manualmente.

5.1.7 Modos Usuario y Kernel

Windows utiliza dos niveles de privilegio: procesador en modo kernel y modo de usuario. Todas las funciones descritas en este capítulo hasta ahora son funciones en modo de usuario, pero hay formas equivalentes de hacer lo mismo en modo kernel. Casi todo el código se ejecuta en modo usuario, excepto los controladores del sistema operativo y hardware, los cuales se ejecutan en modo kernel. En el modo de usuario, cada proceso tiene su propia memoria, permisos de seguridad, y los recursos. Si un programa en modo de usuario ejecuta una instrucción no válida y colapsa, Windows puede recuperar todos los recursos y terminar el programa.
Normalmente, el modo de usuario no puede acceder directamente al hardware, y está restringido a sólo un subconjunto de todos los registros y las instrucciones disponibles en la CPU. Con el fin de manipular el hardware o cambiar el estado en el núcleo, mientras que, en el modo de usuario, debe depender de la API de Windows.

5.1.8 La API Nativa

La API nativa es una interfaz de bajo nivel para interactuar con Windows que rara vez se utilizan en los programas no maliciosos, pero es muy popular entre los creadores de malware. Llamar a funciones de la API nativa evitan el API de Windows normal.
Cuando se llama a una función de la API de Windows, la función normalmente no realiza la acción solicitada directamente, porque la mayoría de las estructuras de datos importantes se almacenan en el kernel, el cual no es accesible por el código externo al kernel (código en modo usuario). Microsoft ha creado un proceso de múltiples etapas por el que las aplicaciones de usuario pueden lograr la funcionalidad necesaria. La siguiente figura ilustra la forma en que esto funciona para la mayoría de llamadas a la API.

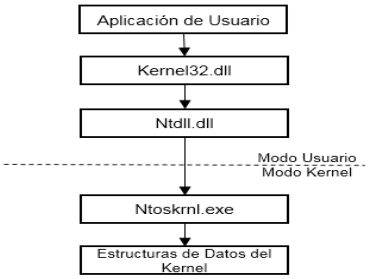

Figura 18. Modos Usuario y Kernel

Las aplicaciones de usuario se les da acceso a las API de modo usuario tales como kernel32.dll y otros archivos DLL, los cuales llaman a ntdll.dll, el cual es una DLL especial que gestiona las interacciones entre el espacio de usuario y el kernel. El procesador entonces cambia al modo kernel y ejecuta una función en el kernel, que normalmente se

encuentra en ntoskrnl.exe. El proceso es complicado, pero la separación entre las API del kernel y de usuario permite a Microsoft cambiar el kernel sin afectar las aplicaciones existentes.

5.2 N-gramas en la API de Windows

En los campos de probabilidad y lingüística computacional, un n-grama es una secuencia contigua de n elementos de una determinada secuencia de texto o de habla. Los elementos pueden ser fonemas, silabas, letras de acuerdo a su aplicación. Los n-gramas típicamente se recolectan de un corpus de texto o de habla. Cuando los elementos son palabras los n-gramas también pueden ser llamadas "shingles" [70].

Un n-grama de tamaño 1 se nombra "unigrama", el de tamaño 2 "bigrama", 3 "trigrama". Los n-gramas de mayor tamaño son nombrados por el valor de n, es decir; "4-grama", "5-grama", etc.

La siguiente tabla muestra diferentes ejemplos de secuencias y su modelo de secuencia de n-grama presentada en [71]:

Campo	Unidad	Ejemplo de secuencia	Secuencia 1-grama	Secuencia 2-grama	Secuencia 3-grama
Secuencia de Proteinas	Amino ácido	…Cys-Gly-Leu-Ser-Trp …	…, Cys, Gly, Leu, Ser, Trp, …	…, Cys-Gly, Gly-Leu, Leu-Ser, Ser-Trp, …	…, Cys-Gly-Leu, Gly-Leu-Ser, Leu-Ser-Trp, …
Secuencia DNA	Pares base	…AGCTTCGA…	…, A, G, C, T, T, C, G, A, …	…, AG, GC, CT, TT, TC, CG, GA, …	…, AGC, GCT, CTT, TTC, TCG, CGA, …

Tabla 9. Ejemplos de n-gramas

En la siguiente tabla se muestran ejemplos de los n-gramas obtenidos de las muestras de malware en cada modelo:

Modelo	n-grama de Win API Call
1grama	{NtOpenFile}
2grama	{LdrGetProcedureAddress-NtQueryInformationFile}
3grama	{LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress}
4grama	{OpenServiceW-RegQueryValueExA-CreateThread-RegCloseKey}
5grama	{NtDelayExecution-ZwMapViewOfSection,DeleteFileA-NtDelayExecution-DeleteFileA}

Tabla 10. Ejemplos de los primeros cinco modelos de Win API Calls

5.3 Diseño del experimento

En el experimento se descargaron las 900 muestras de los 6 diferentes tipos de malware de malwr.com [65] considerando diferentes familias de cada tipo [72], aunado a ello se utilizaron 234 ejecutables del directorio System32 como whiteware. Los tipos de malware corresponden a los siguientes: Backdoors, Troyanos, Troyanos Espía, Gusanos, Rootkits y Virus (definidos en capítulo 2) [69]:

1. **Puertas traseras o Backdoors:** son un método externo en el proceso de autenticación o en otros controles de seguridad con el fin de acceder a un sistema de cómputo o a los datos contenidos en el mismo [8].

2. **Troyanos:** es un tipo de software malicioso que se empaqueta junto con una pieza útil del software o se hace pasar por una pieza de software útil. Una vez que el troyano es activado, que por lo general pasa desapercibido por el usuario, se libera una carga útil de ellos ya sea como un virus o una puerta trasera que puede permitir a un usuario acceder remotamente al sistema [6].

3. **Troyanos Espía:** software que obtiene información de una persona u organización sin su conocimiento y que puede enviar tal información a otra entidad sin el consentimiento del cliente, o que impone el control sobre una computadora sin el conocimiento del cliente [3].

4. **Gusanos:** Un programa usualmente pequeño que auto replica su contenido en sí mismo y que invade computadoras en una red y generalmente realiza acciones destructivas [10].

5. **Rootkits:** programas maliciosos que se ocultan en el sistema a través de modificaciones en las herramientas del sistema, filtrando activamente información de estado del sistema de los usuarios, enmascarando la presencia de archivos, servicios y canales de comunicación maliciosos [9].

6. **Virus:** Un programa que está usualmente oculto dentro de otro programa aparentemente inocuo y que produce copias de sí mismo e inserta otros programas y usualmente realiza una acción maliciosa (como destruir datos) [7].

Siguiendo la metodología del experimento en el Capítulo 4, se utilizó el mismo escenario de Sandboxing para obtener las Win API de cada proceso ejecutado en las muestras, sin embargo, en este caso, cada muestra sólo se ejecuta una vez y por el tiempo que Cuckoo considera para su análisis.

5.3.1 Preprocesamiento de los datos

Después de obtener las funciones Win API de los procesos ejecutados en cada muestra a través de un script en bash (se repite lo mismo para cada variación en el tamaño de n-gramas):

1. Se genera un corpus por cada tipo de malware con las Win API calls
2. Se obtienen los n-gramas de cada corpus
3. Se realiza un compilado de los corpus para formar un diccionario con los n-gramas únicos
4. Se obtienen los n-gramas por cada muestra
5. Se genera la tabla de presencia de n-grama en cada muestra por tipo de malware
6. Se realiza un compilado de las tablas de cada tipo de malware

Finalmente, para cada base de conocimiento se tienen los siguientes números de n-gramas:

Malware/Modelo	1	2	3	4	5	6	7	8	9	10
Backdoors	110	1208	3148	5060	6721	8153	9413	10604	11802	13032
Troyanos	122	1894	6158	11115	15786	20080	23983	27693	31229	34576
Troyanos-Espía	110	1422	4428	8148	11848	15283	18256	20937	23389	25686
Gusanos	120	2576	10492	21882	34265	46996	59726	72581	85513	98838
Rootkits	50	218	414	567	684	779	860	940	1015	1082
Virus	115	1470	4641	8656	12619	16334	19625	22680	25624	28480
Whiteware	259	2271	3640	4415	4973	5347	5616	5836	5994	6106
Total	886	11059	32921	59843	86896	112972	137479	161271	184566	207800
Únicos	**294**	**5465**	**20015**	**41064**	**63977**	**86738**	**108818**	**130641**	**152311**	**174280**

Tabla 11. Número de n-gramas por tipo de malware y por modelo de n-grama

5.4 Modelo de clasificación de malware usando n-gramas de Win API calls y Máquinas de Soporte Vectorial de Kernel Polinomial

El objetivo del experimento es obtener la viabilidad de utilizar un modelo de n-gramas de las Win API calls para la clasificación de malware. Para la obtención de resultados se utilizó el software Weka en su versión 3.7.13 y se realizaron 10 separaciones aleatorias de datos de entrenamiento y datos de prueba con los valores predeterminados de la herramienta para los algoritmos que se utilizaron en este experimento, además de los siguientes parámetros del clasificador SVM:

Tipo de experimento: División porcentual de conjunto de entrenamiento y prueba (con datos extraídos de forma aleatoria).
Porcentaje de entrenamiento: 70, Clasificador: Función SMO
Tamaño del lote: 0, C: 1.0
Épsilon:1.0E-12, Tipo de Filtro: Normalizar los datos de entrenamiento
Kernel: PolyKernel -E 1.0 -C 250007
Número de cifras decimales: 6

5.4.1 Resultados

Al realizar diez iteraciones con diferentes algoritmos de clasificación mostrados en la Tabla 12, puede apreciarse que las SVM con Kernel Polinomial se posicionan en el segundo mejor promedio de exactitud después de Random Forest el cual tiene una desviación estándar del valor promedio de exactitud mayor (el cual se muestra en la Tabla 13), en el modelo de 3gramas que fue el mejor para la mayoría de algoritmos.

Algoritmo/Modelo	1	2	3	4	5	6	7	8	9	10
LogitBoost	67.31	71.04	72.20	69.68	70.04	68.38	67.07	67.62	69.11	68.27
NaiveBayes	60.76	62.65	61.40	60.95	60.00	58.89	60.29	59.93	61.27	62.06
NaiveBayesMultinomial	63.05	69.36	70.91	70.77	68.51	67.93	67.55	68.88	69.41	69.70
RandomForest	75.69	75.96	76.35	75.96	74.99	68.83	66.58	71.61	74.10	73.72
SVM RBF Kernel	67.46	70.84	69.64	68.94	68.44	67.64	65.66	66.73	68.75	67.58
SVM Poly Kernel	72.69	74.34	75.50	75.81	75.29	74.37	72.60	72.15	73.59	73.60
SVM Poly Kernel Normalized	72.22	73.87	72.70	72.30	71.94	71.93	70.56	70.89	72.68	72.64

Tabla 12. Exactitud promedio $\overline{acc}$ por modelo de n-grama y algoritmo

Algoritmo/Modelo	1	2	3	4	5	6	7	8	9	10
LogitBoost	1.67	1.45	1.60	1.50	1.21	1.53	2.45	1.25	1.95	1.32
NaiveBayes	2.81	1.95	1.40	2.89	2.68	1.74	1.35	2.05	1.00	1.78
NaiveBayesMultinomial	2.51	1.53	1.77	2.54	1.91	2.37	2.20	2.23	1.39	1.62
RandomForest	1.85	1.58	1.94	2.02	1.87	3.33	5.50	3.23	1.03	2.33
SVM RBF Kernel	2.34	1.50	1.25	1.68	1.90	2.67	1.88	1.85	1.35	1.32
SVM Poly Kernel	2.09	1.78	1.26	1.80	1.24	1.48	1.87	1.41	1.72	2.59
SVM Poly Kernel Normalized	2.05	1.23	1.43	1.90	1.54	2.17	1.33	1.32	2.17	1.28

Tabla 13. Desviación Estándar de $\overline{acc}$ por modelo de n-grama y algoritmo

5.5 Modelo de detección de malware empleando n-gramas de Win API calls y el clasificador Bayesiano

En este experimento el objetivo es determinar que n-gramas de Win API calls están relacionados con la presencia de malware para cada una de las bases de conocimiento, y con ello analizar si existe una correlación entre los n-gramas con un mayor valor de ε y $S(c|x)$. Para llevarlo a cabo se utilizó una implementación del clasificador Naive Bayes en un script de bash, el cual toma como entrada cada base de conocimiento y de manera aleatoria selecciona el 70% de cada una para formar el conjunto de entrenamiento y su complemento como conjunto de prueba, posteriormente genera el modelo obteniendo los valores de ε y $S(c|x)$.

Es importante mencionar que para llevar a cabo este experimento se utilizaron el mismo número de muestras de cada tipo de malware para equipar el de whiteware; es decir que teniendo un total de 234 registros de whiteware, se eligieron de manera aleatoria 39 muestras de cada tipo de malware, obteniendo un total de 468 registros por cada una de las diez bases de conocimiento.

Por cada n-grama único de las tablas vistas en la Tabla 11, el conjunto de variables se puede representar por el vector $X = < x_1, x_2, x_3, \dots, x_N >$, dónde N varía dependiendo de la base de conocimiento que se utilice para generar el modelo de probabilidad, el cual al tener 468 valores asociados a cada elemento puede representarse por una matriz, dónde para cada valor de x tenemos 468 valores correspondientes a diferentes instancias del problema, en virtud de que los datos de entrenamiento constituyen el 70% de las muestras.

5.5.1 Variables predictoras y función de Score

Siguiendo la metodología del experimento del Capítulo 4, depuramos cada modelo eliminando todos los valores dónde ε se encuentren en el intervalo abierto: $-1.96 < \varepsilon < 1.96$. Después de ordenar los valores ascendentemente por ε de cada modelo, se obtuvieron los primeros valores de cada uno, como puede apreciarse en tabla 15:

Modelo	n-grama	ε
1gramas	ZwMapViewOfSection	10.99
2gramas	LdrGetProcedureAddress-LdrGetProcedureAddress	12.02
3gramas	LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress	11.93
4gramas	LdrLoadDll-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress	11.84
5gramas	LdrLoadDll-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress	12.07
6gramas	LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress	11.53
7gramas	LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress	11.65
8gramas	LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress	11.71
9gramas	LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress	11.24
10gramas	LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress	11.36

Tabla 15. Primeras 10 variables predictores ordenadas por su valor ε de cada modelo de n-grama con valor 1

Posteriormente, con los modelos depurados se obtuvieron los primeros valores de $\boldsymbol{S(c|x)}$ de cada modelo ordenados descendentemente, cuyo resultado se muestra en la Tabla 16, la cual manifiesta que a excepción del modelo de 1gramas, en cuyo caso el n-grama con mayor valor de $\boldsymbol{\varepsilon}$ es *ZwMapViewOfSection*, la tendencia del n-grama malicioso de cardinalidad mayor, es una secuencia de la Win API call *LdrGetProcedureAddress*, dónde es preciso indicar que en los modelos de 5gramas y 6gramas es antecedido por *LdrLoadDll*. Además de que la relación de $\boldsymbol{N(x|c)}$ de los n-gramas respecto al total de muestras maliciosas es de más del 50%.

A continuación, se presenta la definición y características principales de las Win API calls ya mencionadas:

a) *ZwMapViewOfSection*: es una función de kernel encargada de mapear en el mismo espacio de memoria que el proceso del programa actual las DLLs externas. Ya que cuando una aplicación Windows se carga en memoria, todas las funciones externas requeridas en Librerías de Enlace Dinámico (DLLs), son cargadas en el mismo espacio de memoria y mapeadas en memoria como si fueran una parte integral de la aplicación. En el contexto de los desarrolladores de malware una vez que se hace el hooking (se conoce que la Win API call es ejecutada y se intercepta) de esta instrucción, el mapeo de funciones de DLLs puede ser alterado. Debido a que cada función agregada mientras se carga una DLL debe ser copiado al espacio de memoria del proceso llamado, el remplazo de funciones también necesita ser inyectado en este espacio de memoria y con esto llevar a cabo una inyección de proceso. [73]

b) *LdrGetProcedureAddress*: se utiliza para encontrar la dirección de una función en un módulo, y a diferencia de *GetProcAddress* el cual se encuentra en kernel32.dll, se ubica en ntdll.dll, y como se ha mencionado anteriormente es la interfaz entre el modo usuario y el modo kernel, lo cual permite a los desarrolladores de malcode hacer llamadas al sistema de manera directa al kernel y evitar las soluciones de seguridad que solo realizan el hooking en modo usuario. Esta técnica es muy recurrida por los creadores de rootkits. [74]
c) *LdrLoadDll*: sirve para cargar una DLL a un proceso, análogamente a la Win API call anterior es la versión ntdll.dll de LoadLibrary que se encuentra en kernel32.dll, la presencia de esta importación puede indicar que un programa está intentando ser oculto. [75]

Realizando un análisis con las características descritas anteriormente las funciones *LdrLoadDll* y *LdrGetProcedureAddress* permiten a un programa acceder a cualquier función en cualquier biblioteca en el sistema y debido a la persistencia de *LdrGetProcedureAddress* se puede inferir que se trata de un programa malicioso por la frecuencia de ese n-grama en las muestras maliciosas y su valor de $S(c|x)$ en todos los modelos, es importante mencionar que *ZwMapViewOfSection* al integrar varias DLL en un mismo espacio de trabajo se ajusta perfectamente a una de las tendencias actuales en el desarrollo de malware, la cual corresponde a inyección de DLLs maliciosas.

Modelo	n-grama	$S(c\|x)$	$N(x\|c)$
1gramas	ZwMapViewOfSection	4.11	120
2gramas	LdrGetProcedureAddress-LdrGetProcedureAddress	4.27	145
3gramas	LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress	4.26	150
4gramas	LdrLoadDll-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress	4.29	135
5gramas	LdrLoadDll-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress	4.30	124
6gramas	LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress	4.20	133
7gramas	LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress	4.22	134
8gramas	LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress	4.24	138
9gramas	LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress	4.15	128
10gramas	LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress	4.17	138

Tabla 16. Primeras 10 variables predictores ordenadas por su valor $S(c|x)$ de cada modelo de n-grama con valor 1

Después de obtener la correlación en cada modelo respecto al n-grama malicioso con el valor más alto de $\boldsymbol{S}(\boldsymbol{c}|\boldsymbol{x})$, se propuso hacer un análisis minucioso de los resultados a través de la frecuencia de los diez n-gramas con mayor valor de $\boldsymbol{S}(\boldsymbol{c}|\boldsymbol{x})$ para cada modelo, considerando los valores más altos en frecuencia de cada uno y omitiendo aquellos n-gramas que tengan la menor entropía respecto al n-grama con la cardinalidad más alta que los contenga y considerando tener un ejemplo de cada uno. Lo anterior se muestra en la siguiente tabla:

Modelo	n-grama	Frecuencia
2gramas	LdrGetProcedureAddress-LdrGetProcedureAddress	46
1gramas	ZwMapViewOfSection	18
3gramas	RegQueryValueExW-RegCloseKey-RegOpenKeyExW	16
6gramas	RegOpenKeyExW-RegQueryValueExW-RegCloseKey-RegOpenKeyExW-RegQueryValueExW-RegCloseKey	6
4gramas	NtCreateMutant-NtOpenSection-ZwMapViewOfSection-SetWindowsHookExA	6
7gramas	LdrGetProcedureAddress-RegOpenKeyExW-RegQueryValueExW-RegCloseKey-RegOpenKeyExW-RegQueryValueExW-RegCloseKey	4
5gramas	LdrLoadDll-NtCreateMutant-NtOpenSection-ZwMapViewOfSection-SetWindowsHookExA	4
9gramas	LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetDllHandle-LdrGetDllHandle-LdrGetProcedureAddress-RegOpenKeyExW-RegQueryValueExW	3
8gramas	NtOpenKey-LdrLoadDll-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress-LdrGetProcedureAddress	3

Tabla 17. Tabla de top de frecuencias de presencia de n-gramas con mayor $\boldsymbol{\varepsilon}$ y de valor 1

A continuación, se presenta la definición y características principales de las Win API calls en la tabla:

a) *RegQueryValueExW*: recupera el tipo de datos y un nombre de valor especificado asociado con una clave de registro abierto. [76]
b) *RegCloseKey*: esta Win API call libera el handle de la clave especificada, acepta un solo argumento, el handle de la clave abierta, y devuelve 0 si la llave se cerró con éxito. [76]

c) *RegOpenKeyExW*: esta función se utiliza para abrir un handle a una clave de registro para su lectura y la edición. Las claves de registro se escriben a veces como una forma de software para lograr persistencia en un host. El registro también contiene una gran cantidad de información de sistema operativo y configuración de la aplicación. [76]
d) *NtCreateMutant*: crea un objeto mutante y abre un handle. El objeto Mutant se usa para emular semáforos mutex OS / 2 2.0. Aunque Windows NT proporciona otras, más sencillas, capacidades para sincronizar el acceso a las secciones críticas, este objeto se ha incluido para permitir la emulación más eficiente las capacidades de los OS / 2 2.0. [77]
e) *NtOpenSection*: la apertura de una sección provoca un handle para el objeto que se abre de manera que una vista de la sección se puede mapear en el espacio de direcciones virtual del proceso sujeto. Un proceso no puede abrir un objeto de

sección a menos que los tipos de acceso deseados sean permitidos por la sección objeto ACL, y, si la sección está respaldada por un archivo de datos, también son compatibles con el modo de apertura del archivo de datos asociado. Además de los errores y los errores de cuota de administración de objetos asociados con objetos de apertura, los siguientes valores de estado pueden ser devueltos por la función [78]:

1) STATUS_NORMAL - Normal, finalización exitosa.
2) STATUS_INVALID_PARAMETER - Error, Fue especificado un parámetro inválido.

f) *SetWindowsHookExA*: esta función se utiliza para establecer una función de hooking que se llamará cada vez que un determinado evento se llama. De uso común con los keyloggers y spyware, esta función también proporciona una manera fácil de cargar un archivo DLL en todos los procesos de interfaz gráfica de usuario en el sistema. Esta función se agrega a veces por el compilador.
Se instala un procedimiento de hooking definido por la aplicación en una cadena de hooking. El cual se puede usar para supervisar ciertos eventos del sistema. Estos eventos están asociados ya sea con un hilo específico o con todos los hilos en el mismo escritorio como el subproceso de llamada. [79]

g) *LdrGetDllHandle*: encuentra los DLL inyectados asociados con el hooking del espacio de usuario en su espacio de proceso, es la versión de ntdll.dll cuya versión de kernel32.dll es GetModuleHandle. [80]

h) *NtOpenKey*: NtOpenKey y ZwOpenKey son dos versiones de la misma rutina de *Servicios del Sistema Nativo* Windows. La rutina NtOpenKey en el kernel de Windows no puede acceder directamente a los controladores de modo kernel. Sin embargo, los controladores en modo kernel pueden acceder a esta rutina indirectamente al llamar a la rutina ZwOpenKey la cual abre una clave de registro existente. [81]

Con la información anterior puede concluirse que en los modelos de 3gramas y 6gramas el malware crea y modifica claves de registro para ser persistente; es decir que inclusive después del reinicio este mantenga su funcionamiento. Por otro lado, el modelo de 4gramas muestra un n-grama que sugiere una inyección de proceso; y que por la última llamada usa una técnica de evasión en su detección ya que el hooking opera del lado de la muestra. Llevando este úlitmo n-grama al modelo de 5gramas sugiere que la inyección de proceso es a través de una DLL ya que el n-grama se mantiene antecedido por la llamada al sistema de *LdrLoadDll*, por otro lado, los modelos de 8gramas y 9gramas presentan el mismo comportamiento que se discutió en la Tabla 16.

5.5.2 Resultados

Al igual que el experimento del cap, se realizaron diez iteraciones con diferentes algoritmos de clasificación mostrados en la Tabla 18, en la cual puede apreciarse que nuestra implementación de Naive Bayes obtuvo los mejores resultados de exactitud considerando todos los n-gramas y concretamente con el modelo de 3gramas obtuvo su mejor resultado con 99.31% y con una desviación estándar de 1. Sin embargo, respecto a la desviación estándar de su exactitud obtuvo el penúltimo lugar. Y considerando esta relación quien obtuvo los mejores resultados fue Naive Bayes Multinomial, quien queda en segundo lugar de mejores resultados de exactitud y en primero de menor desviación estándar. Es

importante mencionar que aun así todos los algoritmos tienen una exactitud de predicción muy buena ya que es de más del 90%.

Algoritmo/n-grama	1	2	3	4	5	6	7	8	9	10
LogitBoost	99.50	99.36	98.01	97.37	97.11	96.02	95.94	95.49	95.73	94.55
NaiveBayes	99.08	96.37	95.52	94.59	93.42	92.98	92.69	91.70	93.11	82.79
Naive Bayes*	99.23	99.22	99.31	98.58	98.55	97.10	97.83	96.38	95.52	96.45
NaiveBayesMultinomial	99.50	100	99.72	98.65	97.25	96.74	96.24	95.12	95.58	93.13
RandomForest	99.15	99.86	99.01	98.44	97.54	96.81	96.82	95.63	94.83	93.50
SVM RBF Kernel	99.08	97.94	96.66	95.59	94.07	93.77	93.63	92.72	93.63	92.98
SVM Poly Kernel	99.43	99.93	99.08	98.37	96.89	96.45	95.58	94.32	94.90	93.34
SVM Poly Kernel Normalized	99.29	98.22	97.37	95.38	94.58	94.35	93.99	92.72	94.45	92.91

Tabla 18. Exactitud promedio $\overline{acc}$ por modelo de n-grama y algoritmo

Algoritmo/n-grama	1	2	3	4	5	6	7	8	9	10
LogitBoost	1.49	0.98	1.38	1.49	1.37	1.66	1.72	1.75	1.42	1.77
NaiveBayes	2.77	2.04	1.44	1.48	1.71	2.03	1.96	2.91	1.63	2.15
NaiveBayes*	1.58	1.43	0.99	1.76	1.56	3.12	1.86	3.12	2.23	3.87
NaiveBayesMultinomial	1.49	0	0.57	1.08	1.63	1.59	1.57	1.94	1.04	1.71
RandomForest	2.55	0.43	1.23	1.22	1.34	1.65	2.07	1.63	1.48	2.13
SVM RBF Kernel	2.77	1.78	1.82	1.29	1.64	1.94	1.84	2.45	1.33	1.41
SVM Poly Kernel	1.70	0.21	0.90	1.31	1.21	1.69	1.75	2.30	1.46	1.41
SVM Poly Kernel Normalized	2.13	1.50	1.56	1.85	1.62	1.76	1.57	2.64	1.56	1.58

Tabla 19. Desviación Estándar de $\overline{acc}$ por modelo de n-grama y algoritmo

CAPÍTULO VI

«¿A vos no te pasa que te despertás a veces con la exacta conciencia de que en ese momento empieza una increíble equivocación?»
Julio Cortázar

Conclusiones

6.1 Conclusiones y discusión de resultados

Una de las aplicaciones esenciales al utilizar un sistema informático, especialmente si nos conectamos a Internet desde él, es un sistema antivirus o software de seguridad. Sin embargo, un antivirus convencional con un motor basado en firmas no nos protege del malware "zero-day", el cual resulta de las vulnerabilidades que no han sido publicadas y por tanto no existe algún parche en la tecnología dónde se presenta, su nombre se debe a que se tiene cero días para arreglar esa vulnerabilidad, el tipo de malware más peligroso, especialmente en aquellas empresas, que necesitan un departamento de seguridad y una serie de herramientas de análisis forense (de un precio muy elevado) para ser capaces de protegerse de este malware y en consecuencia de las millones de variantes de otros tipos de malware que no han sido catalogados en las bases de firmas de antivirus.

En ese sentido la tarea en detección de malware se vuelve cada vez compleja, en la presente investigación se han mostrado esencialmente dos experimentos:

Detección de Malware, Enfoque Basado en los Contadores de Rendimiento de Windows y un Clasificador Bayesiano:

En este experimento se obtuvo una exactitud promedio del 70.33% con una desviación estándar de 13.29 tras realizar 10 ejecuciones de nuestra implementación del Naive Bayes, que como se muestra en la Tabla 8, al compararse con otros algoritmos incluyendo la implementación de Weka, representa la exactitud más baja, y su desviación estándar es la menor, lo cual significa que a pesar de tener un valor de 70% como mínimo, ésta no tendrá mucha variación independientemente del número de iteraciones que se realice.

Las ventajas que se presentan en este modelo son:

a) El muestreo de los contadores de rendimiento no representa un reto en términos de programación, ya que estos pueden ser accesibles ya sea a través de logman o con un script de powershell que el mismo sistema operativo desde su versión de Windows 7 ofrece como parte de sus programas predeterminados.

b) El proceso de minería de datos no demanda demasiado ya que la naturaleza de los datos es netamente numérica.
c) La complejidad algorítmica del algoritmo de Naive Bayes representa una de los más bajas en el contexto de los algoritmos de aprendizaje supervisado.
d) Al llevar este modelo a una implementación práctica en tecnologías como Splunk, y tener las variables más relacionadas con la presencia de malware es relativamente fácil implementar una solución del tipo generación de alertas con los resultados obtenidos y el monitoreo de las características más representativas con sus valores correspondientes.
e) A diferencia de las soluciones antivirus que llevan a cabo la comparación de firmas, el realizar un monitoreo en tiempo real se puede predecir comportamiento malicioso en tiempo.

Las desventajas que se presentan en este modelo son:
a) Se emplearon 4 muestras de malware de Gusanos de red.
b) El whiteware utilizado son los programas contenidos en la carpeta de System32.
c) Las naturalezas de las variables relacionadas con la presencia de malware son físicas; en virtud de ello se tuvo que repetir n veces la ejecución de malware para poder generar una base de conocimiento con el comportamiento promedio durante 3 min de malware y whiteware.
d) El tiempo de muestreo es de 1 segundo, lo cual impacta directamente en el proceso de almacenamiento y de tráfico en red si se lleva a una solución práctica de monitoreo.
e) La exactitud que ofrece el algoritmo Naive Bayes no es alta, con ello se pueden generar demasiados falsos positivos al llevar el modelo a una solución de monitoreo.
f) Sólo fue probado en Windows 7 de 32 bits.
g) Al emplearse un algoritmo de aprendizaje supervisado limita el alcance de la aplicación a sólo los patrones que están relacionados con las muestras definidas como malware.

Detección de Malware con Modelo de Lenguaje y su Clasificación:

En este experimento se generaron dos enfoques para minar las bases de conocimiento, los cuales son:

1) Modelo de clasificación de malware usando n-gramas de Win API calls y Máquinas de Soporte Vectorial de Kernel Polinomial:

Al realizar diez iteraciones con diferentes algoritmos de clasificación mostrados en la Tabla 12, puede apreciarse que las SVM con Kernel Polinomial se posicionan en el segundo mejor promedio de exactitud después de Random Forest el cual tiene una desviación estándar del valor promedio de exactitud mayor (el cual se muestra en la Tabla 13), en el modelo de 3gramas con 75.5% que fue el mejor para la mayoría de algoritmos y para el algoritmo de SVM Polinomial obtuvo una desviación estándar de 1.26 en ese mismo modelo.

Las ventajas que se presentan en este modelo son:
a) La exactitud que proporciona el algoritmo de SVM Polynomial es una de los mejores respecto a los que se comparan, además de que su desviación estándar no es alta.

b) A diferencia de otros análisis de secuencias de syscalls, la generación de n-gramas su complejidad computacional en tiempo y espacio es menor.
c) La clasificación de malware permite especificar el análisis por parte del analista.
d) La complejidad algorítmica de SVM es menor que los algoritmos de minado de secuencias.
e) Con las bases de conocimiento actual se determina que el modelo de 3gramas es el mejor para la clasificación de malware usando el algoritmo de SVM Polinomial.

Las desventajas que se presentan en este modelo son:

a) El número de muestras de malware que se usaron fue de 900.
b) La infraestructura con la que se realizó el experimento fue de máquinas virtuales.
c) A pesar de que SVM Polinomial tiene uno de los mejores desempeños, es bajo respecto al número de falsos positivos que pudieran derivar de este.
d) El hooking de las syscalls se llevó a cabo con la SandBox Cuckoo, sin embargo, al llevarlo a la práctica a una solución del tipo Splunk requiere conocimientos especializados sobre las Win API calls para su monitoreo en tiempo real.
e) El número de Win API calls que se generan es exponencial en breves periodos de tiempo, lo cual lleva nuevamente a problemas de tráfico y almacenamiento.
f) Sólo fue probado en Windows 7 de 32 bits.
g) Al emplearse un algoritmo de aprendizaje supervisado limita el alcance de la aplicación a sólo los patrones que reconoce están relacionados con las muestras definidas como malware.

2) Modelo de detección de malware empleando n-gramas de Win API calls y el clasificador Bayesiano:

En este experimento puede apreciarse que nuestra implementación de Naive Bayes obtuvo los mejores resultados de exactitud considerando todos los n-gramas y concretamente con el modelo de 3gramas obtuvo su mejor resultado que puede verse en la Tabla 18. Sin embargo, respecto a la desviación estándar de su exactitud obtuvo el penúltimo lugar respecto a los otros algoritmos. Y considerando esta relación quien obtuvo los mejores resultados fue Naive Bayes Multinomial, quien queda en segundo lugar de mejores resultados de exactitud y en primero de menor desviación estándar.

Las ventajas que se presentan en este modelo son:

a) La exactitud de nuestra implementación de Naive Bayes se mantiene excelente para la mayoría de modelos, arriba del 90%.
b) Al realizar minería de datos sobre las 10 bases de conocimiento nos permitió hallar correlaciones de n-gramas sobre las Win API calls que pueden representar procesos maliciosos.
c) La implementación de un sistema de este tipo en soluciones de tipo Splunk permite la detección de procesos maliciosos en tiempo real.

Las desventajas que se presentan en este modelo son:

a) El número de muestras de malware que se usaron fue de 900.
b) El whiteware utilizado son los programas contenidos en la carpeta de System32.

c) El hooking de las syscalls se llevó a cabo con la SandBox Cuckoo, sin embargo, al llevarlo a la práctica a una solución del tipo Splunk requiere conocimientos especializados sobre las Win API calls para su monitoreo en tiempo real.
d) Sólo fue probado en Windows 7 de 32 bits.
e) Al emplearse un algoritmo de aprendizaje supervisado limita el alcance de la aplicación a sólo los patrones que reconoce están relacionados con las muestras definidas como malware.

Después de llevar a cabo ambos experimentos, se comprobó la hipótesis que problemas de seguridad informática como la detección y clasificación de malware pueden tratarse desde la perspectiva de inteligencia artificial con algoritmos de aprendizaje supervisado y usando como fuentes de datos: los Contadores de Rendimiento de Windows y las Syscallas de tipo Win API para predecir comportamientos maliciosos y por tanto detectarlos, además de poder clasificar entre seis diferentes tipos de malware. Dicha tarea que es llevada a cabo por los analistas de malware, consume tiempo y recursos. Sin perder de vista que el alcance de esta investigación no llega hasta la interpretación detallada del comportamiento malicioso. En vez de ello sugiere tendencias en los patrones de comportamiento del malware para poder predecir su presencia, en variables físicas y de funciones del sistema operativo.

6.2 Líneas de trabajo futuro

Finalmente, como trabajo futuro creemos que el rendimiento del enfoque actual se puede mejorar aumentando el número de muestras y de fuentes de información por cada tipo de malware para ambos experimentos, así como la infraestructura podría cambiarse a toda una topología de red con hosts reales y diferentes versiones del sistema operativo Windows incluyendo las versiones actuales del mismo, así como variar la generación de modelos con whiteware que no sólo sea del directorio System32 y que incluya comportamiento normal de usuario y/o dicho comportamiento se encuentre sujeto a políticas operativas de instituciones bancarias, empresariales, industriales, educativas, etc. Así mismo se pueden realizar análisis por tipo de malware para el primer experimento con el fin de determinar hasta qué punto el modelo puede acaparar diferentes tipos y/o familias de malware y entre ellos cuales tienen un mayor grado de predictibilidad. Todo lo anterior con el fin de ofrecer una solución fuera de la esfera académica y que realmente proporcione un enfoque pragmático en el sentido de su aplicación, lo cual nos lleva a que en la implementación se puedan resolver problemas como el almacenamiento y tráfico de red, eliminando todas aquellas variables que después de llevar a cabo el proceso de minería de datos no tengan contribución alguna en la tarea de predicción.

Referencias

[1] P. V. Shijo and a. Salim, “Integrated Static and Dynamic Analysis for Malware Detection,” *Procedia Comput. Sci.*, vol. 46, no. Icict 2014, pp. 804–811, 2015.

[2] A. Moser, C. Kruegel, and E. Kirda, “Limits of Static Analysis for Malware Detections.”

[3] H. Zhao, M. Xu, N. Zheng, J. Yao, and Q. Hou, “Malicious executables classification based on behavioral factor analysis,” in *IC4E 2010 - 2010 International Conference on e-Education, e-Business, e-Management and e-Learning*, 2010, pp. 502–506.

[4] M. Ahmadi, A. Sami, H. Rahimi, and B. Yadegari, “Malware detection by behavioural sequential patterns,” *Comput. Fraud Secur.*, vol. 2013, no. 8, pp. 11–19, 2013.

[5] V. S. Shcherbina and V. A. Zakharov, “USING ALGEBRAIC MODELS OF PROGRAMS FOR DETECTING METAMORPHIC MALWARES R . I . Podlovchenko , N . N . Kuzyurin ,” vol. 172, no. 5, pp. 740–751, 2011.

[6] C. Wang, J. Pang, R. Zhao, W. Fu, and X. Liu, “Malware detection based on suspicious behavior identification,” in *Proceedings of the 1st International Workshop on Education Technology and Computer Science, ETCS 2009*, 2009, vol. 2, pp. 198–202.

[7] R. Tian, R. Islam, L. Batten, and S. Versteeg, “Differentiating malware from cleanware using behavioural analysis,” in *Proceedings of the 5th IEEE International Conference on Malicious and Unwanted Software, Malware 2010*, 2010, pp. 23–30.

[8] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, “A survey on automated dynamic malware-analysis techniques and tools,” *ACM Computing Surveys*, vol. 44, no. 2. pp. 1–42, 2012.

[9] R. Islam, R. Tian, L. M. Batten, and S. Versteeg, “Classification of malware based on integrated static and dynamic features,” *Journal of Network and Computer Applications*, vol. 36, no. 2. pp. 646–656, 2013.

[10] P. Jonathan and B. Vázquez, “PoC : Captura de malware con el honeypot Dionaea - Parte I,” *Rev. .Seguridad UNAM CERT*, 2015.

[11] T. Sochor and M. Zuzcak, “Study of Internet Threats and Attack Methods Using Honeypots and Honeynets,” in *Computer Networks SE - 12*, vol. 431, A. Kwiecień, P. Gaj, and P. Stera, Eds. Springer International Publishing, 2014, pp. 118–127.

[12] R. Wa, G. Hunt, and D. Brubacher, “Detours: Binary Interception of Win32 Functions,” *Proc. 3rd USENIX Wind. NT Symp.*, pp. 135–143, 1999.

[13] U. Bayer, C. Kruegel, and E. Kirda, “TTAnalyze : A Tool for Analyzing Malware.”

[14] U. Bayer, A. Moser, C. Kruegel, and E. Kirda, “Dynamic analysis of malicious code,” pp. 67–77, 2006.

[15] G. Willems, T. Holz, and F. Freiling, “Toward automated dynamic malware analysis using CWSandbox,” *IEEE Secur. Priv.*, vol. 5, no. 2, pp. 32–39, 2007.

[16] C. Guarnieri, “CuckooSandbox,” 2015. [Online]. Available: http://www.cuckoosandbox.org/about.html. [Accessed: 01-Jun-2015].

[17] M. Gheorghescu, “An Automated Virus Classification System,” *Virus Bull. Conf.*, no. October, pp. 294–300, 2005.

[18] K. Rieck, P. Trinius, C. Willems, and T. Holz, “Automatic Analysis of Malware Behavior using Machine Learning,” pp. 1–30, 2011.

[19] G. Editor and S. Detection, “Understanding Anti-Virus Software,” no. March, 2011.

[20] I. Santos, Y. K. Penya, J. Devesa, and P. G. Bringas, “N-grams-based File Signatures for Malware Detection.,” *Proc. 11th Int. Conf. Enterp. Inf. Syst.*, pp. 317–320, 2009.

[21] A. Tang, S. Sethumadhavan, and S. J. Stolfo, “Unsupervised Anomaly-Based Malware Detection Using Hardware Features,” *Raid*, vol. 8688, pp. 109–129, 2014.

[22] C. Rossow, C. Dietrich, H. Bos, L. Cavallaro, M. Van Steen, F. C. Freiling, and N. Pohlmann, “Sandnet: Network traffic analysis of malicious software,” *BADGERS ’11 Proc. First Work. Build. Anal. Datasets Gather. Exp. Returns Secur.*, pp. 78–88, 2011.

[23] C. Richardson, “Virus detection with machine learning,” 2009.

[24] R. J. Canzanese, “Detection and Classification of Malicious Processes Using System Call Analysis,” 2015.

[25] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, “A sense of self for Unix processes,” *Proc. 1996 IEEE Symp. Secur. Priv.*, pp. 120–128, 1996.

[26] S. A. Hofmeyr, S. Forrest, and A. Somayaji, “Intrusion Detection using Sequences of System Calls,” *J. Comput. Secur.*, vol. 6, no. 3, pp. 151–180, 1998.
[27] C. Warrender, S. Forrest, and B. Pearlmutter, “Detecting Intrusions Using System Calls:Alternative Data Models,” *1999 IEEE Symp. Secur. Priv.*, vol. 0, no. c, pp. 133–145, 1999.
[28] Y. Liao and V. Vemuri, “Using Text Categorization Techniques for Intrusion Detection.,” *USENIX Secur. Symp.*, 2002.
[29] D. Kang, D. Fuller, and V. Honavar, “Learning Classifiers for Misuse Detection Using,” pp. 511–516, 2005.
[30] X. Xu, “Sequential anomaly detection based on temporal-difference learning: Principles, models and case studies,” *Appl. Soft Comput. J.*, vol. 10, no. 3, pp. 859–867, 2010.
[31] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, “Crowdroid: Behavior-Based Malware Detection System for Android,” *Proc. 1st ACM Work. Secur. Priv. smartphones Mob. devices - SPSM ’11*, p. 15, 2011.
[32] L. Martignoni, E. Stinson, M. Fredrikson, S. Jha, and J. C. Mitchell, “Behaviors,” pp. 78–97, 2008.
[33] A. Tokhtabayev, V. Skormin, and A. Dolgikh, “Dynamic, resilient detection of complex malicious functionalities in the system call domain,” *Proc. - IEEE Mil. Commun. Conf. MILCOM*, pp. 1349–1356, 2010.
[34] B. Mehdi, F. Ahmed, S. A. Khayyam, and M. Farooq, “Towards a theory of generalizing system call representation for in-execution malware detection,” *IEEE Int. Conf. Commun.*, 2010.
[35] M. Bailey, J. Oberheide, and J. Andersen, “Automated classification and analysis of internet malware,” *Recent Adv. Intrusion Detect.*, pp. 1–18, 2007.
[36] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X. Zhou, X. Wang, U. C. S. Barbara, and S. Antipolis, “Effective and Efficient Malware Detection at the End Host,” *System*, vol. 4, no. 1, pp. 351–366, 2009.
[37] “Machine Learning Project at the University of Waikato in New Zealand.” [Online]. Available: http://www.cs.waikato.ac.nz/ml/index.html. [Accessed: 16-Dec-2015].
[38] H. Xiao and T. Stibor, “A supervised topic transition model for detecting malicious system call sequences,” *Proc. 2011 Work. Knowl. Discov. Model. Simul. - KDMS ’11*, p. 23, 2011.
[39] D. Canali, A. Lanzi, D. Balzarotti, and C. Kruegel, “A quantitative study of accuracy in system call-based malware detection,” *Proc. ...*, pp. 122–132, 2012.
[40] A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda, “AccessMiner: Using System-Centric Models for Malware Protection,” *Proc. 17th ACM Conf. Comput. Commun. Secur. -- CCS’10*, pp. 399–412, 2010.
[41] C. Wysopal and T. Shields, “Static Detection of Application Backdoors Detecting both malicious software behavior and malicious,” *Info*.
[42] C. Simmons, C. Ellis, S. Shiva, D. Dasgupta, and Q. Wu, “AVOIDIT: A cyber attack taxonomy,” *9th Annu. Symp. Inf. Assur.*, pp. 12–22, 2014.
[43] M. Sikorski and A. Honig, *Pratical Malware Analysis*. 2012.
[44] P. Mell, K. Kent, and J. Nusbaum, “Guide to malware incident prevention and handling recommendations of the national institute of standards and technology,” *Nist Spec. Publ. 800-83*, p. 101, 2005.
[45] M. Skrzewski, “Monitoring System ’ s Network Activity for Rootkit Malware Detection,” pp. 157–165, 2013.
[46] M. Wazid, R. Sharma, A. Katal, R. H. Goudar, P. Bhakuni, and A. Tyagi, “Implementation and Embellishment of Prevention of Keylogger Spyware Attacks,” *Commun. Comput. Inf. Sci.*, vol. 377 CCIS, pp. 262–271, 2013.
[47] and A. A. Scott E. Donaldson, Stanley G. Siegel, Chris K. Williams, *Enterprise Cybersecurity - How to Build a Successful Cyberdefense Program Against Advanced Threats*. 2015.
[48] A. Stamminger, C. Kruegel, G. Vigna, and E. Kirda, “Automated spyware collection and analysis,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5735 LNCS, pp. 202–217, 2009.
[49] O. Tsigkas and D. Tzovaras, “Analysis of rogue anti-virus campaigns using hidden structures in k-partite graphs,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7712 LNCS, pp. 114–125, 2012.
[50] S. Bowles and J. Hernandez-Castro, “The first 10 years of the Trojan Horse defence,” *Comput. Fraud Secur.*, vol. 2015, no. 1, pp. 5–13, 2015.

[51] J. Aycock, “Spyware and adware,” *Adv. Inf. Secur.*, vol. 50, no. 2, pp. 1–142, 2011.
[52] P. P. K. Chan, C. Yang, D. S. Yeung, and W. W. Y. Ng, “Spam filtering for short messages in adversarial environment,” *Neurocomputing*, vol. 155, pp. 167–176, 2015.
[53] M. P. Simone, “Malware 101 - Viruses,” *Sans Inst.*, p. 27, 2009.
[54] D. Oktavianto and I. Muhardianto, *Cuckoo Malware Analysis*. 2013.
[55] T. M. Mitchell, *Machine Learning*. McGraw-Hill Science/Engineering/Math, 2009.
[56] D. Conway and J. M. White, *Conway - Machine Learning for Hacker - 2012*. .
[57] B. Makhabel, *Learn Data Mining with R*. .
[58] J. Brownlee, “A Tour of Machine Learning Algorithms,” pp. 1–31, 2013.
[59] Wolfram Math World, “Bayes’ Theorem.” [Online]. Available: http://mathworld.wolfram.com/BayesTheorem.html. [Accessed: 13-Jun-2015].
[60] Wolfram Math World, “Standard Deviation.” [Online]. Available: http://mathworld.wolfram.com/StandardDeviation.html. [Accessed: 13-Jun-2015].
[61] O. D. M. P. R. H. Schützeer, *An Introduction to Information Retrieval*, no. c. 2009.
[62] C. R. Stephens, “An Introduction To Data Minning,” pp. 455–484, 2006.
[63] S. O. Haykin, *Neural Networks and Learning Machines*. 2008.
[64] C. Huffman, *Windows Performance Analysis Field Guide*. 2015.
[65] Cuckoo Sandbox, “Malware repository.” [Online]. Available: https://malwr.com/. [Accessed: 02-Jun-2015].
[66] F-Secure, “Net-Worm:W32/Kolab,” 2015. [Online]. Available: https://www.f-secure.com/v-descs/net-worm_w32_kolab_qa.shtml. [Accessed: 02-Jun-2015].
[67] K. Labs, “How to clean a corporate network from Net-Worm.Win32.Kido,” 2014. [Online]. Available: http://support.kaspersky.com/viruses/solutions/4673. [Accessed: 02-Jun-2015].
[68] M. Seth, K. J. Drobatz, D. B. Church, and R. S. Hess, “Receiver Operating Characteristic,” *J Vet Intern Med*, 2011. [Online]. Available: https://en.wikipedia.org/wiki/Receiver_operating_characteristic. [Accessed: 13-Sep-2016].
[69] Alex I. Valencia-Valencia,Sofía N. Galicia-Haro, “Detección de malware con modelo de lenguaje y su clasificación mediante SVM,” *Advances in Natural Language Processing and Computational Linguistics Research in Computing Science*, México, p. 9, 9-18 ISSN 1870-4069 (2016).
[70] “Syntactic Clustering of the Web.” [Online]. Available: http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-TN-1997-015.pdf. [Accessed: 15-Dec-2015].
[71] G. Sidorov, “Syntactic Dependency Based N-grams in Rule Based Automatic English as Second Language Grammar Correction,” *Int. J. Comput. Linguist. Appl.*, vol. 4, no. 2, pp. 169–188, 2013.
[72] V. Bontchev, F. Software, and I. Thverholt, “Current Status of the CARO Malware Naming Scheme.”
[73] R. Vieler, *Professional Rootkits*. Wrox Press, 2007.
[74] KumaT, “bypass fle x AC,” 2010. .
[75] J. B. Anonymous, “Bypassing 3rd Party Windows Buffer Overflow Protection,” 2016. .
[76] K. Getz and M. Gilbert, *VBA Developer’s Handbook*. 2000.
[77] D. N. Cutler, “Windows NT Mutant Specification,” 1990.
[78] L. Perazzoli, “Windows NT Virtual Memory Specification,” 1993.
[79] P. Engineering, O. N. June, C. Here, G. Thedemand, S. Of, R. Engineer, O. U. R. Hands, and O. N. Training, “Using SetWindowsHook SetWindowsHookEx for DLL Injection on Windows Reverse Engineering,” 2016. .
[80] J. L. and U. Shamir, “Malware Discovered – SFG: Furtim Malware Analysis.” .
[81] Microsoft, “NtOpenKey,” 2016. .