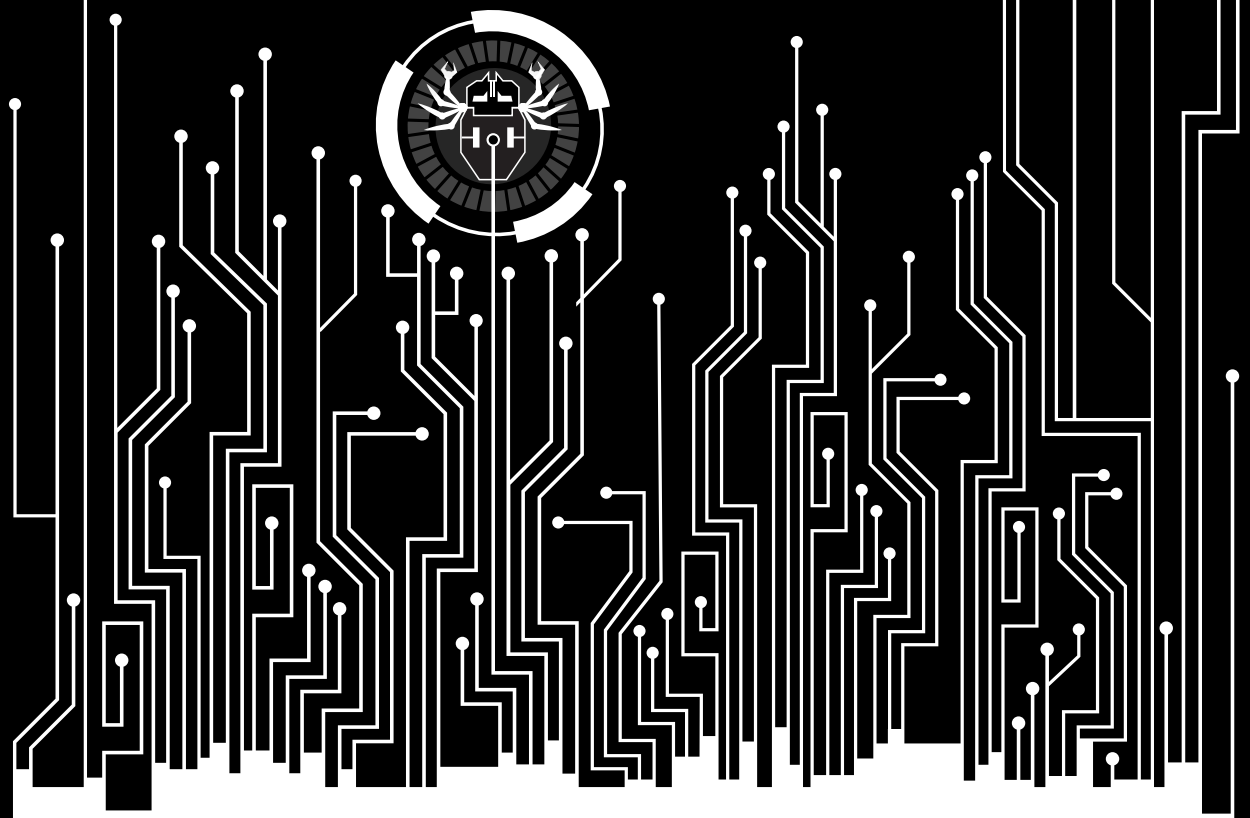
CHRISTIAN CAMILO URCUQUI LÓPEZ
MELISA GARCÍA PEÑA
JOSÉ LUIS OSORIO QUINTERO
ANDRÉS NAVARRO CADAVID



CIBERSEGURIDAD
UN ENFOQUE DESDE LA CIENCIA DE DATOS

UNIVERSIDAD ICESI
Editorial Universidad Icesi

Ciberseguridad: un enfoque desde la ciencia de datos

Ciberseguridad: un enfoque desde la ciencia de datos

Christian Camilo Urcuqui
Melisa García Peña
José Luis Osorio Quintero
Andrés Navarro Cadavid

Editorial Universidad Icesi, 2018

Ciberseguridad: un enfoque desde la ciencia de datos
© Christian Camilo Urcuqui L., Melisa García P., José Luis Osorio Q. y Andrés Navarro C.

1 ed. Cali, Colombia. Universidad Icesi, 2018
86 p., 19x24 cm
Incluye referencias bibliográficas
ISBN: 978-958-8936-55-0
https://doi.org/10.18046/EUI/ee.4.2018

1. Machine learning 2. Intelligent systems 3. Cyber security I.Tit
006 – dc22

© Universidad Icesi, 2018
Facultad de Ingeniería

Rector: Francisco Piedrahita Plata
Decano Facultad de Ingeniería: Gonzalo Ulloa Villegas
Coordinador editorial: Adolfo A. Abadía

Producción y diseño: Claros Editores SAS.
Editor: José Ignacio Claros V.
Diseño de portada: Jhoan Sebastian Pérez.
Impresión: Carvajal Soluciones de Comunicación.

Impreso en Colombia / *Printed in Colombia.*

El contenido de esta obra no compromete el pensamiento institucional de la Universidad Icesi ni le genera responsabilidades legales, civiles, penales o de cualquier otra índole, frente a terceros.

Editorial Universidad Icesi

Calle 18 #122-135 (Pance), Cali-Colombia
editorial@icesi.edu.co
www.icesi.edu.co/editorial
Teléfono: +57(2) 555 2334

Christian Camilo Urcuqui López

Máster en Informática y Telecomunicaciones e Ingeniero de Sistemas con énfasis en Administración e Informática de la Universidad Icesi (Cali, Colombia), y Especilista en *Deep Learning*. Docente investigador, miembro del Grupo de Investigación en Informática y Telecomunicaciones (i2t) y Coordinador del Club de Hacking de la Universidad Icesi. Sus áreas de interés profesional son: ciberseguridad, ciencia de datos y *machine learning*. ulcamilo@gmail.com

Melisa García Peña

Ingeniera de Sistemas de la Universidad Icesi (Cali, Colombia) actualmente vinculada al Banco de Occidente (Cali). Durante la preparación de su trabajo de grado "Sistemas Open Source para la detección de ataques a páginas web", fue monitora en el Grupo de Investigación en Informática y Telecomunicaciones (i2t), donde participó en el proyecto Sniff, primordialmente en la elaboración de estado del arte sobre *antidefacement*. megape82@hotmail.com

José Luis Osorio Quintero

Ingeniero de Sistemas de la Universidad Icesi (Cali, Colombia), actualmente vinculado al Banco de Occidente (Cali). Durante la preparación de su trabajo de grado "Sistemas Open Source para la detección de ataques a páginas web", fue monitor en el Grupo de Investigación en Informática y Telecomunicaciones (i2t), donde participó en el proyecto Sniff, primordialmente en la elaboración del estado del arte sobre *antidefacement*. josquin@outlook.com

Andrés Navarro Cadavid

Doctor Ingeniero en Telecomunicaciones de la Universidad Politecnica de Valencia (España), Máster en Gestión Tecnológica e Ingeniero Electrónico de la Universidad Pontificia Bolivariana (Medellín, Colombia). Es profesor de tiempo completo y Director del Grupo de Investigación en Informática y Telecomunicaciones (i2t) de la Universidad Icesi (Cali, Colombia). Es Investigador Senior (Colciencias); miembro senior del IEEE y presidente del capítulo Comunicaciones del IEEE Colombia; consultor internacional; y miembro de Grupo de Estudio 1 de la Unión Internacional de Telecomunicaciones. anavarro@icesi.edu.co

Tabla de contenido

Resumen **17**
Presentación **19**
Ciberseguridad y ciencia de datos **21**
Introducción **21**
Ciberseguridad **21**
Ciencia de datos **23**
Machine learning **26**
Ciencia de datos y ciberseguridad **30**
Ciberseguridad en Android **31**
Estado del arte **35**
Metodología **40**
Framework de análisis estático **40**
Generador de características **41**
Análisis de permisos **44**
Trabajo futuro **46**
Ciberseguridad en aplicaciones web **49**
Estado del arte **52**
Metodología **57**
Fase 1. Generación de datasets **58**
Fase 2. Preprocesamiento de los datos **59**
Fase 3. Procesamiento de los datos **60**
Fase 4. Selección de algoritmos y de medidas de evaluación **63**
Experimento **63**

Resultados **64**
Análisis **70**
Trabajo futuro **71**
A partir de las lecciones aprendidas **73**
Aplicación de la ciencia de datos al análisis de ciberamenazas **74**
Conjuntos de datos **76**
Un camino prometedor **76**
Referencias **77**

Índice de Tablas

Tabla 1. Métodos para el análisis de amenazas cibernéticas 22
Tabla 2. Ciclo de vida de la analítica de datos en Big Data 25
Tabla 3. Medidas de evaluación de la eficacia de los algoritmos de machine learning 28
Tabla 4. Medidas de confusión para problemas de dos clases 28
Tabla 5. Medidas de desempeño para problemas de dos clases 28
Tabla 6. Algoritmos de clasificación 29
Tabla 7. Arquitectura de Android 31
Tabla 8. Desempeño de los clasificadores de Naïve Bayes 42
Tabla 9. Desempeño de los clasificadores de Bagging 42
Tabla 10. Desempeño de los clasificadores de KNN 42
Tabla 11. Desempeño de los clasificadores de SVM 43
Tabla 12. Desempeño de los clasificadores de SGD y DT 43
Tabla 13. Desempeño individual de los seis clasificadores 43
Tabla 14. Permisos accedidos por las aplicaciones - Dataset I 44
Tabla 15. Permisos accedidos por las aplicaciones descargadas de Google Play.. 45
Tabla 16. Desempeño en la prueba de generalización 46
Tabla 17. OWASP Top Ten de los riesgos para la seguridad 2017 50
Tabla 18. Características de la capa de aplicaciones 54

Tabla 19. Características de la capa de red 55
Tabla 20. Características - Capa de aplicación 60
Tabla 21. Características - Capa de red 61
Tabla 22. Ejemplo de matriz de datos 62
Tabla 23. Ejemplo matriz con variables dummy 62
Tabla 24.Frecuencia de los datos no numéricos de la capa de aplicación 64
Tabla 25. Promedio de los datos numéricos de la capa de aplicación 65
Tabla 26. Promedio de los datos numéricos de la capa de red 65
Tabla 27. Resultados de los algoritmos por cada capa (k=10) 69
Tabla 28. Resultados de los algoritmos para las tres características obtenidas por los métodos subset e infogain (k=10) 69
Tabla 29. Resultados de los algoritmos para la matriz de datos completa 70

Índice de Figuras

Figura 1. Android Software Stack 32
Figura 2. Arquitectura de Safe Candy 34
Figura 3. Marco de trabajo para el análisis estático 40
Figura 4. Resultados: área bajo la curva 44
Figura 5. Generalización: área bajo la curva 46
Figura 6. Marco de trabajo para detección de páginas maliciosas 58
Figura 7. Correlación de los datos benignos de la capa de red 66
Figura 8. Correlación de los datos maliciosos de la capa de red 67
Figura 9. Correlación de los datos benignos de la capa de aplicación 68
Figura 10. Correlación de los datos maliciosos de la capa de aplicación 68
Figura 11. Proceso de aplicación de la ciencia de datos en ciberseguridad 74

Acrónimos

APK	Android Application Package
ASEF	Android Security Evaluation Framework
ASUM-DM	Analytics Solutions Unified Method for Data Mining
AUC	Area Under Curve
AVD	Android Virtual Devices
CDA	Confirmatory Data Analysis
CFG	Control Flow Graph
CRISP-DM	Cross-Industry Standard Process for Data Mining
DoS	Denegation of Services
DT	Decision Tree
EDA	Exploratory Data Analysis
FN	Falsos Negativos
FP	Falsos Positivos
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IDS	Intrusion Detection System
IoT	Internet of Things
JAR	JAva Archiver

JVM	Java Virtual Machine
KNN	K-Nearest Neighbor
KPI	Key Performance Index
LIBSVM	Library for Support Vector Machines
MLP	Multi-Layer Perceptron
NB	Naïve Bayes
NN	Neural Networks
OWASP	Open Web Application Security Project
PCA	Principal Components Analysis
RF	Random Forest
RFI	Remote File Inclusion
ROC	Receiver Operating Characteristics
SAAF	Static Android Analysis Framework
SEMMA	Sample, Explore, Modify, Model, and Assess
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
TIC	Tecnologías de la Información y las Comunicaciones
URL	Uniform Resource Locator
VN	Verdaderos Negativos
VP	Verdaderos Positivos
XSS	Cross-Site Scripting
XXE	Entidades Externas XML

Los proyectos base de este documento: "Safe Candy: plataforma de análisis, validación y configuración de seguridad en aplicaciones Android" y "Ciberseguridad 2.0: Mejoramiento de la seguridad de la información a los sitios web de las entidades públicas, por medio de una arquitectura de control informático", fueron implementados gracias al financiamiento del Departamento Administrativo de Ciencia, Tecnología e Innovación (Colciencias), Password Consulting Services y la Univerisidad Icesi.

Resumen

El desarrollo de las tecnologías de la información y las comunicaciones requiere de mecanismos de seguridad que garanticen la confidencialidad, integridad y disponibilidad de la información, y el desarrollo de habilidades para detectar y controlar oportunamente las nuevas amenazas. A pesar del gran desarrollo de las líneas de defensa, la creatividad y la creciente capacidad tecnológica de los cibercriminales hacen más compleja la tarea, por lo que las metodologías tradicionales no son suficientes.

Con base en los resultados de dos proyectos de investigación en seguridad informática, uno focalizado en la detección de malware en dispositivos móviles con sistema operativo Android, el otro en el control de *defacement* en páginas web, se exploró la viabilidad de la aplicación de la ciencia de datos para el desarrollo de soluciones a problemas en ciberseguridad. En el primer tema, se evaluaron seis clasificadores de *machine learning* para la detección de *malware* a partir de permisos accedidos, sus resultados mostraron la factibilidad de preparar algoritmos de clasificación capaces de generalizar a otro conjunto de datos, a partir de un *dataset* de entrenamiento. En el segundo, luego de generar *datasets* apropiados (con URL benignos y malignos), se seleccionaron algoritmos de clasificación y medidas de evaluación, se realizó un análisis exploratorio de los *datasets* (capas de aplicación y de red), y se revisaron las bondades y limitaciones de los clasificadores propuestos.

Con base en los resultados de ambas experiencias, se propone un procedimiento base para la construcción de un marco de trabajo, que muestra las actividades necesarias para: el entrenamiento y la evaluación de modelos de *machine learning* para la detección de *malware* en dispositivos con sistema operativo Android, y la identificación *a priori* de páginas web maliciosas.

Presentación

La ciencia de datos se ha venido aplicando en distintos campos gracias a su capacidad de proveer soluciones aproximadas a problemas no resolubles por sistemas convencionales. Ha tomado fuerza en la medida en que responde adecuadamente a los retos que representan los grandes volúmenes de información que se manejan hoy en día y a las robustas capacidades computacionales requeridas para su uso.

La ciberseguridad requiere de un esfuerzo constante para el desarrollo de soluciones que garanticen, no solo la integridad, disponibilidad y confidencialidad de la información, sino también su capacidad de detección de nuevas amenazas informáticas. Esto representa un reto, tanto para los sistemas, como para los investigadores, por la complejidad de sus variables, especialmente por el desarrollo acelerado de las tecnologías y la notable y creciente astucia y capacidad técnica de los cibercriminales.

La comunidad de desarrolladores e investigadores ha estado trabajando en técnicas, marcos de trabajo (*frameworks*) y soluciones para mejorar los niveles de seguridad de las distintas tecnologías. Entre las propuestas se encuentran: los análisis estático, dinámico e hibrido; la aplicación de la inteligencia artificial; y las metodologías de detección de amenazas cibernéticas, tales como: la identificación por firmas y el descubrimiento a través de anomalías. Un camino prometedor es la aplicación de la ciencia de datos para el desarrollo de soluciones de software, por ejemplo, el uso de modelos predictivos especializados para la detección de *malware* y la predicción de ciberataques web.

En este libro se aborda la aplicación de la ciencia de datos para el desarrollo de soluciones aproximadas a problemas en ciberseguridad a partir de un

recorrido que va desde los fundamentos teóricos de la ciencia de datos y la ciberseguridad, hasta su aplicación en proyectos de investigación. De estos últimos, se tratan dos investigaciones con enfoque de soluciones para defensa: la primera, orientada al análisis de software malicioso para dispositivos con sistema operativo Android; la segunda, a la detección de sitios web con contenido perjudicial.

Estas dos investigaciones surgieron de las necesidades, experiencias y resultados de dos proyectos de ciberseguridad desarrollados por el Grupo de Investigación en Informática y Telecomunicaciones de la Universidad Icesi (i2t) en asocio Password Consulting Services, con financiamiento del Departamento Administrativo de Ciencia, Tecnología e Innovación (Colciencias): "Safe Candy: plataforma de análisis, validación y configuración de seguridad de aplicaciones Android" y "Ciberseguridad 2.0: Mejoramiento de la seguridad de la información a los sitios web de las entidades públicas, por medio de una arquitectura de control informático". Las experiencias de estos dos proyectos han sido incluidas con el fin de resaltar por qué la ciencia de datos es un área importante para el análisis de amenazas cibernéticas.

En el texto se presentan además los estudios más recientes de la aplicación de ciencia de datos en ciberseguridad, las ventajas y desventajas de las técnicas de análisis para detección de software malicioso y los caminos pendientes de explorar en la investigación y el desarrollo de soluciones de seguridad informática.

Finalmente, a partir de la revisión del estado del arte y de las lecciones aprendidas en los proyectos realizados, se propone un marco de trabajo de ciencia de datos para la detección de amenazas digitales en un contexto de investigación, el cual le permite al lector conocer cuáles son las actividades necesarias para el entrenamiento y la evaluación de modelos de *machine learning* para la detección de *malware* en dispositivos con sistema operativo Android y la identificación *a priori* de páginas web maliciosas.

Ciberseguridad y ciencia de datos

Introducción

La seguridad informática o ciberseguridad no es tema reciente, ha presentado retos desde el momento en que los primeros computadores se conectaron a redes de comunicaciones, sin embargo, el crecimiento de la Internet; la popularización de los dispositivos móviles inteligentes; y la aparición y el crecimiento de tecnologías como la Internet de las Cosas (IoT, *Internet of Things*), las ciudades inteligentes y sostenibles, y las redes eléctricas inteligentes, han aumentado su importancia y complejidad. Es ahí donde la ciencia de datos surge como una opción para mejorar los mecanismos de análisis que requieren los sistemas cibernéticos para hacerle frente a los diferentes tipos de riesgos de seguridad que existen en la actualidad. La ciencia de datos, si bien puede ayudar a mejorar la seguridad, también puede servir para erosionarla, por lo que es importante mantener la dinámica de análisis y desarrollo de nuevas estrategias que garanticen el mejoramiento continuo de la seguridad informática. A continuación se presenta la revisión de algunos conceptos relacionados con la ciberseguridad y algunas técnicas de la ciencia de datos y de los sistemas de aprendizaje de máquina (*machine learning*).

Ciberseguridad

La ciberseguridad es el área de las ciencias de la computación encargada del desarrollo y la implementación de los mecanismos de protección de la información y de la infraestructura tecnológica. Debido a la importancia de tener sistemas seguros, que garanticen la confidencialidad, integridad y

disponibilidad de los datos, se han propuesto: métodos para la evaluación de amenazas cibernéticas (técnicas de análisis y marcos de trabajo); prácticas y herramientas para el desarrollo de software y hardware seguros; y sistemas de seguridad, entre otros. Estas propuestas han permitido tener líneas de defensa contra los cibercriminales y el software malicioso, sin embargo, dado el desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC) –con sus nuevas propuestas, como la IoT y el *Big Data*–, la presencia de nuevas vulnerabilidades [1] y los ataques de día cero, la ciberseguridad requiere de un trabajo constante para poder mitigar los riesgos. Existen distintas herramientas y técnicas para el análisis de amenazas cibernéticas, algunas de las más representativas se presentan en la Tabla 1.

Tabla 1. Métodos para el análisis de amenazas cibernéticas

Método	Descripción
Análisis estático	Técnica que evalúa los comportamientos maliciosos en el código fuente, los datos o los archivos binarios, sin ejecutar directamente la aplicación [2]. Su complejidad ha aumentado debido a la experiencia que han adquirido los cibercriminales en el desarrollo de aplicaciones. Se ha demostrado que es posible evadir este análisis a partir de técnicas de ofuscación, como las descritas por Sharif et al. [3].
Análisis dinámico	Métodos automatizados que estudian el comportamiento del *malware* en ejecución mediante un análisis de la interactividad del atacante y permiten evaluar características que solo pueden ser obtenidas mientras el software está en funcionamiento, como por ejemplo: la inyección de código en ejecución [4], los procesos en ejecución, la interfaz de usuario, las conexiones de red y la apertura de sockets [5]. Existen técnicas que permiten evadir este análisis, como las descritas por Petsas et al., [6] donde el *malware* tiene la capacidad de detectar ambientes sandbox y detener su comportamiento malicioso.
Análisis híbrido	Método que combina las ventajas de la aplicación de los análisis dinámico y estático.
Inteligencia artificial	Área que provee de una serie de técnicas para dar soluciones aproximadas a problemas complejos. Una de ellas, el *machine learning* tiene como propósito proveer a los sistemas de la capacidad de aprender cómo identificar a un *malware* sin ser programado de forma explícita. Gran cantidad de propuestas [7-12] usan algoritmos de clasificación, tales como: *Support Vector Machines* (SVM), *Bagging*, *Neural Network* (NN), *Decision Tree* (DT) y *Naïve Bayes* (NB) . Existen otras aplicaciones de la inteligencia artificial, como por ejemplo, las técnicas de programación genética para la detección de anomalías en peticiones HTML.

Para la detección de ciberataques se han propuesto dos marcos de trabajo [13]: el método basado en firmas, que tiene como finalidad detectar amenazas a partir de una base de datos que contiene distintas características (firmas) de peticiones

o archivos maliciosos; y el método de detección por anomalías, que tiene dos actividades, una dedicada a la construcción de un perfil del sitio de análisis a partir de ciertas variables, y otra enfocada en el monitoreo y la detección de anomalías (cambios no registrados) en un perfil previamente creado.

¿Es posible desarrollar un sistema determinista que garantice el cien por ciento de seguridad? Probablemente no, debido a que las soluciones de seguridad informática deben enfrentar variables que hacen de sus retos tareas crecientes en complejidad, entre ellas: el continuo aumento de las habilidades y capacidades de desarrollo de los cibercriminales, los ataques de día cero, las malas prácticas de desarrollo de software y hardware, las nuevas tecnologías, los *insiders* y los requerimientos no funcionales de los sistemas informáticos.

El software convencional de seguridad para la identificación de amenazas cibernéticas requiere de un esfuerzo humano que implica tiempo y recursos, la aplicación de la ciencia de datos es actualmente uno de los caminos más prometedores, porque a través de sus técnicas permite conseguir soluciones aproximadas a problemas complejos [14].

Ciencia de datos

La ciencia de datos tiene como objetivo obtener elementos de valor de distintas fuentes de información –incluso datos que pueden ser incompatibles y estar en distintos formatos–, a través de técnicas y herramientas que incluyen métodos de estadística, minería de datos, *machine learning* y visualización. Esta ciencia busca encontrar y entender patrones en los datos con el fin de generar modelos que representan al contexto de la información.

Un modelo es una representación general de las relaciones entre los atributos de los datos y una función matemática, estadística o una serie de reglas que asocia la información. Existen dos tipos de modelos: descriptivos, que tienen como objetivo proveer mayor información acerca del contexto de los datos –relaciones causa efecto–; y predictivos, encargados de estimar un objetivo a partir de una serie de valores.

En la ciencia de datos existen dos enfoques de análisis que dependen del nivel de conocimiento del científico acerca de la información: el análisis exploratorio de los datos (EDA, *Exploratory Data Analysis*), que tiene como finalidad conocer las relaciones o patrones que existen en los datos, aplicable cuando de antemano

no se tiene una hipótesis o un entendimiento de ellos; y análisis de datos confirmatorio (CDA, *Confirmatory Data Analysis*), que asume que el científico tiene una hipótesis acerca de la información y tiene como objetivo probarla o descartarla a partir de los modelos creados.

El EDA busca entender las relaciones y la calidad de los datos a partir de la extracción de atributos cuantitativos y de la producción de indicadores y resúmenes visuales basados en la aplicación de métodos estadísticos.

Los análisis numéricos hacen parte de la estadística descriptiva, la cual se subdivide en tres categorías: medidas de tendencia central, medidas de dispersión y medidas de asociación: las medidas de tendencia central permiten entender cómo los datos están organizados alrededor del centro de la distribución y cuáles son los valores de mayor ocurrencia (e.g., la media, la mediana y la moda); las medidas de variación o dispersión calculan las distancias entre los datos, es decir, proveen los mecanismos para determinar qué tan esparcidos del centro están (e.g., el rango, el rango intercuartil, la varianza y la desviación estándar); las medidas de asociación permiten conocer la existencia y la fuerza de la relación entre las variables (e.g., la correlación y la covarianza).

Los modelos en la ciencia de datos son de dos tipos: modelos basados en la estadística, desarrollados a partir del análisis de la distribución de los datos y de la probabilidad de la predicción de posibles resultados a partir de una ecuación matemática –que debe ser descubierta– y de los parámetros que mejor estén relacionados a partir de los datos analizados; y modelos basados en algoritmos de *machine learning*, cuyo objetivo es encontrar los datos mejor relacionados (patrones y reglas) y evaluar los resultados de los algoritmos que mejor se ajusten a la solución del problema.

Actualmente se pueden encontrar varias propuestas metodológicas para la aplicación de la ciencia de datos, entre ellas: CRISP-DM (*Cross-Industry Standard Process for Data Mining*), ASUM-DM (*Analytics Solutions Unified Method for Data Mining*) y SEMMA (*Sample, Explore, Modify, Model, and Assess*). Aunque cada una tiene un punto de vista distinto, su objetivo es el mismo: el aprovechamiento de la información.

Para un contexto de grandes volúmenes de información (*Big Data*), se plantea el ciclo de vida de la analítica de datos [14], que se presenta en la Tabla 2. Las etapas son secuenciales y se describen en su orden, salvo la etapa de análisis de datos, la que por su naturaleza es iterativa a sí misma.

Tabla 2. Ciclo de vida de la analítica de datos en *Big Data* [14]

Etapa	Descripción
Evaluación del *Business Case*.	Se enuncian y definen las necesidades, justificaciones y motivaciones del negocio para la definición de los objetivos del análisis. Esta iteración es importante ya que permite obtener una primera idea acerca de los distintos recursos y retos del proyecto, por ejemplo: la identificación de indicadores clave de rendimiento (KPI, *Key Performance Index*), para la futura evaluación de los modelos de datos; las bases de datos internas y externas; los procesos, las tecnologías y la inversión.
Identificación de datos.	Una vez evaluadas las distintas variables del negocio, se procede a la identificación de las fuentes de datos externas e internas. Para datos externos (especialmente de texto, e.g., blogs) puede ser necesario contar con herramientas automatizadas que permitan recolectar la información.
Adquisición y filtrado de datos.	Se recogen los datos desde todas las fuentes que se identificaron durante la etapa anterior y se someten a un filtrado automático para eliminar los datos corruptos y los que no tienen valor para los objetivos de análisis.
Extracción de datos.	Se extraen los datos que estén en un formato no compatible con la solución *Big Data* y se llevan a un formato que ella pueda usar para su análisis.
Validación y limpieza de datos.	Se limpian los datos erróneos a través de reglas o mecanismos preestablecidos y se valida que los datos estén completos y no presenten incoherencias que vayan a dificultar los futuros análisis.
Adición y representación de los datos.	Se llevan a un mismo contexto las distintas fuentes de información, de tal manera que los registros obtenidos, internos y externos, tengan el mismo significado y estén en sus respectivas representaciones (e.g., las tablas en las bases de datos).
Análisis de los datos.	Esta es una actividad iterativa a sí misma, busca aplicar análisis exploratorio o confirmatorio (EDA o CDA) a través del análisis, el entrenamiento y la validación de los resultados de distintos modelos.
Visualización de los datos.	Para que la capacidad de analizar grandes cantidades de datos y encontrar información útil no esté limitada a los analistas, se usan técnicas y herramientas para comunicar gráficamente los resultados del análisis y facilitar su interpretación efectiva por parte de usuarios no expertos.
Utilización de los resultados.	Para obtener el mayor provecho de los datos en términos de su aporte a la toma de decisiones, se define cómo y dónde lo hallado puede ser utillizado.

El análisis con visualización incluye algunas herramientas que facilitan la abstracción de la información, las más usuales son: los mapas de calor, el análisis de series de tiempo y el análisis de redes. El mapa de calor permite apreciar la cantidad de datos cualitativos presentes en unas áreas específicas representadas con colores en matrices o mapas geográficos; el análisis de series de tiempo facilita la comprensión y abstracción de patrones de datos que han sido registrados en intervalos de tiempo; y el análisis de redes permite comprender las relaciones que existen entre los nodos de una red.

Los algoritmos de *machine learning* se utilizan para encontrar soluciones aproximadas a distintos problemas y analizar la información. Existen algoritmos de clasificación, algoritmos de agrupamiento, algoritmos de detección de datos atípicos y algoritmos de filtrado.

Los algoritmos de clasificación a través del aprendizaje supervisado permiten predecir las categorías de una variable nominal u ordinal de un conjunto de datos; los algoritmos de agrupamiento están basados en el aprendizaje no supervisado y buscan encontrar patrones en los datos a través de su segmentación en distintos grupos, es decir, buscan crear agrupaciones de datos que se encuentran relacionados a través de sus propiedades; los algoritmos de detección de datos atípicos tienen como finalidad proveer los mecanismos para la identificación de anomalías o de irregularidades en la información, sean ellas favorables o no al contexto en que se aplica el análisis; y los algoritmos de filtrado buscan y ofrecen pequeñas cantidades de registros provenientes de un gran conjunto de datos por medio de la identificación del comportamiento (patrón) de un usuario (filtrado basado en contenido) o de varios (enfoque colaborativo).

El análisis semántico busca obtener elementos de valor por medio de la extracción de los datos que se encuentran en formato de texto y del reconocimiento de voz, incluye el procesamiento de lenguaje natural, la analítica de texto y el análisis de sentimiento. En la aplicación del procesamiento de lenguaje natural se automatiza el reconocimiento de voz y se ejecutan acciones computacionales dependiendo de los resultados obtenidos; en la analítica de texto se buscan elementos de valor en archivos textuales (datos no estructurados), como correos electrónicos, registros y mensajes en redes sociales; y en el análisis de sentimiento, se busca información acerca de los sentimientos de las personas y su intensidad a través del análisis de texto, sus resultados usualmente son utilizados para la toma de decisiones de una empresa o para la sistematización respuestas dado un texto de entrada.

Machine Learning

La inteligencia artificial presenta distintas definiciones,las cuales varían en dos dimensiones [15]: la primera, orientada a los procesos de pensamiento y razonamiento; y la segunda, encaminada al comportamiento. Estas dimensiones se enfocan al estudio de la fidelidad del rendimiento humano o al concepto de racionalidad.

Machine learning es la parte de la inteligencia artificial que busca que un sistema tenga la capacidad de aprender en entornos variables, sin que sea programado de forma explícita. Su uso ha venido creciendo debido a los volúmenes de información que se presentan en los medios electrónicos (*Big Data*) y a las mayores capacidades computacionales.

Dependiendo del problema a abarcar, se pueden emplear tres técnicas para el entrenamiento de los modelos: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo [16]. En el aprendizaje supervisado se conoce el conjunto de datos, principalmente se comprende la salida correcta del algoritmo y su relación con la entrada, abarca problemas de regresión y clasificación; en el aprendizaje no supervisado no se conoce el conjunto de datos, por lo que su objetivo es encontrar la estructura y los patrones presentes en la información; en el aprendizaje por refuerzo se busca que el algoritmo esté en capacidad de encontrar el conjunto de operaciones más adecuadas para cumplir un objetivo, a través del aprendizaje de reglas y acciones.

Los algoritmos de *machine learning* pueden aplicarse en distintos contextos, dependiendo de su tipo de aprendizaje: clasificación, para predecir la categoría de un ítem; regresión, para predecir un valor continuo; detección de anomalías, para identificar datos atípicos; agrupamiento, para encontrar grupos de elementos similares; asociación, para encontrar reglas de concurrencia; secuencia, para encontrar sucesiones de eventos; resumen, para simplificar la representación de una información; y visualización, para facilitar la comprensión y el descubrimiento.

En los problemas de clasificación existen distintas medidas que permiten evaluar la eficacia de los algoritmos (Tabla 3), entre las que podemos encontrar cuatro muy usuales que hacen parte de la tabla de confusión (Tabla 4) y otros indicadores que facilitan la elección del modelo que mejor cumple con el objetivo del proyecto (Tabla 5).

Existen otros mecanismos de valoración de los algoritmos, como por ejemplo: las Características Operativas del Receptor (ROC, *Receiver Operating Characteristics*), que permite comparar el conjunto de medidas encontradas; el Área Bajo la Curva (AUC, *Area Under Curve*), que permite conocer visualmente la eficacia del clasificador [16]; y el coeficiente de kappa de Cohen, de gran ayuda para aislar el efecto del azar de los datos observados y conocer así el desempeño de los modelos que fueron entrenados a través de un desbalance en

Tabla 3. Medidas de evaluación de la eficacia de los algoritmos de *machine learning*

Tipo	Medida	Descripción
Confusión	Verdaderos Positivos (VP).	Tasa de instancias identificadas correctamente y que hacen parte de su respectiva clase.
	Falsos Negativos (FN).	Tasa de instancias que se identificaron incorrectamente, pero que no hacen parte de una clase especifica.
	Falsos Positivos (FP).	Tasa de datos que fueron identificadas incorrectamente y que pertenecen a una clase especifica.
	Verdaderos Negativos (VN).	Tasa de instancias que fueron identificadas correctamente, pero que no pertenecen a una clase especifica.
Desempeño	Sensitividad.	Probabilidad de obtener un verdadero positivo.
	Error.	Tasa de instancias incorrectamente identificadas de todos los datos estudiados.
	Exactitud (*Accuracy*).	Proporción de datos que fueron correctamente identificados a través de todas las instancias utilizadas.
	Especificidad.	Probabilidad de obtener un verdadero negativo.
	Recuperación (*recall*).	Proporción de datos correctamente clasificados contra el total de datos de su clase.
	Precisión.	Tasa de datos identificados que son realmente relevantes.

Tabla 4. Medidas de confusión para problemas de dos clases [16]

Clase verdadera	Clase predicta		Total
	Positiva	Negativa	
Positiva	Verdadero Positivo (VP)	Falso Negativo (FN)	P
Negativa	Falso Positivo (FP)	Verdadero Negativo (VN)	N
	p’	n’	N

Tabla 5. Medidas de desempeño para problemas de dos clases [16]

Medida	Fórmula
Sensitividad	tn / n =1-fp rate
Error	(fp+fn) / N
Exactitud (*Accuracy*)	(tp+tn) / N=1-error
Especificidad	tp / p= tp rate
Recuperación (*recall*)	tp / p =p
Precisión	tp / p’

la variable objetivo. El tiempo de respuesta, el número de niveles de un árbol de decisión y el número de neuronas o de niveles en una red neuronal entre otras, son también técnicas útiles para evaluar modelos basados en *machine learning*.

Durante el proceso de entrenamiento y evaluación de *machine learning* se debe evitar incurrir en: el sesgo (bias), una medida que indica qué tan lejos está el modelo de la verdad o de la solución del problema, e ilustra la variación de las predicciones respecto de una instancia; y el sobreajuste (*overfitting*), un fenómeno que se presenta cuando los algoritmos son entrenados y ajustados a una proporción de los datos, es decir, cuando los modelos no cuentan con la capacidad de generalizar a otra información que pertenece al contexto del problema y que no fue parte de los procesos de entrenamiento y validación. En la Tabla 6 se describen algunos algoritmos de uso común como clasificadores.

Tabla 6. Algoritmos de clasificación

Algoritmo	Descripción
Naïve Bayes	Clasificador probabilístico para aprendizaje supervisado basado en la aplicación del teorema de Bayes, que asume la independencia entre cada par de variables utilizadas. Muy útil para conjuntos de datos grandes, simple, rápido y muy buen método clasificador. Tiene buen rendimiento con variables categóricas. Para las variables numéricas usa una distribución normal. Durante el estudio McCallum y Nigam [17] se utilizaron dos tipos de Naïve Bayes: Gaussian y de Bernoulli, cuya diferencia radica en el tipo de distribución de los datos (Gaussiana o de Bernoulli, respectivamente).
Bagging	Su finalidad es combinar el resultado de la predicción de múltiples clasificadores aplicados a un remuestreo del conjunto inicial. Hace parte de los métodos de ensamblado que permiten reducir la varianza y el sobre aprendizaje.
K Nearest Neighbours	Algoritmo supervisado que clasifica a cada dato del conjunto en la clase más frecuente de sus k vecinos más cercanos.
Support Vector Machines (SVM)	Construye un modelo con un plano de puntos dividido en dos subconjuntos a partir de un conjunto de puntos, una función (lineal, polinómica, de base radial o sigmoide) y de otros parámetros, para con base en él predecir si un nuevo punto desconocido pertenece a alguna de esas dos categorías [18].
Stochastic Gradient Descent	Algoritmo de optimización que implementa una rutina de aprendizaje de primer orden. El método reduce el costo de las funciones lineales por cada iteración que se produce sobre el conjunto de entrenamiento.
Decision Tree	Algoritmo para problemas de clasificación que crea un modelo con una estructura lógica (árbol) a partir de las características de los datos de entrenamiento.
Regresión logística	Algoritmo que intenta predecir una característica categórica a partir de otras variables independientes.
C.45	Algoritmo que genera reglas para la predicción de variables objetivas con la ayuda de un algoritmo de árbol de clasificación [19]. Su aplicación en el lenguaje R o la plataforma Weka es posible a través de una librería de J48.

Ciencia de datos y ciberseguridad

El software convencional de seguridad requiere de un esfuerzo humano para identificar vulnerabilidades a través de un proceso que permita encontrar sus características y desarrollar la solución sobre la herramienta. Esta es una labor que puede ser más eficiente si se aplica un proceso de análisis a través de la ciencia de datos y de los algoritmos de *machine learning* [20]. La importancia de la aplicación de la inteligencia artificial en la seguridad informática ha sido destacada en varios trabajos, algunos de ellos se relacionan a continuación.

Gandotra, Bansal y Sofat [21] proponen un marco de trabajo para clasificar los *malware* en MS-Windows a través de las características extraídas de los análisis estático y dinámico; en su estudio emplearon como algoritmos de clasificación: MultiLayer Perceptron (MLP), IB1, Decision Tree (DT) y Random Forest (RF) y concluyeron que es posible conseguir buenos resultados empleando en conjunto la información de los análisis estático y dinámico.

Rieck [22] evaluó la aplicación de *machine learning* en la seguridad informática y enunció los problemas, retos y perspectivas de trabajar con ciencia de datos y ciberseguridad en conjunto. Indica además que para aplicar ambas áreas en un sistema, este debe ser efectivo, eficiente, transparente, controlable y robusto. Como parte de las ventajas de contar con un sistema de seguridad con *machine learning* menciona: la detección proactiva de los ataques, el análisis automático de amenazas y el descubrimiento asistido de vulnerabilidades. Concluye que hay buenas esperanzas de este enfoque de trabajo con ambas áreas y propone seguir fomentando investigaciones que hagan uso de ellas.

Ciberseguridad en Android

Android es un sistema operativo de código abierto adquirido por Google en 2005. En 2008 se lanzó su primera versión comercial y en 2009 se liberó su versión 1.5, conocida como Cupcake. Desde esa fecha se han liberado: Donut, Eclair, Froyo, Gingerbread, HoneyComb, IceCream Sandwich, Jelly Bean, KitKat, Lollipop, Marshmallow, Nougat y Oreo (versión 8.0). Aunque fue originalmente desarrollado para teléfonos inteligentes, se puede encontrar en tabletas, televisores, "usables" (wearables) (como gafas y relojes de pulso) y vehículos. A fines de 2018 es el más popular de los sistemas operativos para dispositivos móviles. Su arquitectura incluye cinco componentes [23], [24] (Tabla 7 y Figura 1): una capa de aplicaciones Android, el Marco de trabajo Android, la máquina virtual Dalvik, una capa de código nativo en espacio de usuario y el kernel Linux.

Tabla 7. Arquitectura de Android

Capa	Descripción
Aplicaciones.	Capa de mayor nivel donde se encuentran todas las aplicaciones que interactúan con el usuario, tanto las del sistema (preinstaladas), como las instaladas por él.
Marco de trabajo Android.	Provee las herramientas (paquetes y clases) para que los desarrolladores puedan construir aplicaciones. Las clases permiten interactuar con servicios de alto nivel ofrecidos por el sistema (e.g., *activities*, *services*, *content providers*) y con elementos de UI, y el manejo de bases de datos.
Máquina virtual Dalvik.	Android está mayormente implementado en Java, por ello requiere de una *Java Virtual Machine* (JVM). Dalvik es la JVM utilizada por Android para ejecutar aplicaciones programadas en Java y trabajar con los archivos *.dex*, que se encuentran dentro los APK (*Android Application Package*) o las librerías JAR (*JAva Archiver*).
Código nativo en espacio de usuario.	Capa compuesta del primer proceso de espacio de usuario *init* (encargado de iniciar todos los otros procesos), las librerías y los *daemons* nativos.
Kernel Linux.	Capa que cuenta con los drivers del *hardware*, el *networking*, el acceso al *file-system* y la gestión de procesos.

Figura 1. Android Software Stack [18]

Android incorpora dos modelos de permisos: el primero se refiere a un entorno aislado (*sandbox*) a nivel del Kernel Linux que previene el acceso al *file-system* y a otros recursos de Android; el segundo es una API de permisos que se expone al usuario cuando una aplicación va a ser instalada, es la herramienta que les permite a las aplicaciones acceder a algunos recursos del dispositivo [5].

Cada aplicación de Android esta compilada en un archivo APK que incluye el código de la aplicación (*.dex*), los recursos y el archivo AndroidManifest.xml. Este último provee la información de las características y la configuración de seguridad de cada aplicación [25] e incluye información de la API de permisos, las *activities*, los *services*, los *content providers* y los *broadcast receivers*.

Existen otros mecanismos de seguridad a nivel del desarrollo de las aplicaciones Android (clases de encriptación, comunicaciones seguras en IPC, HTTPS, etc.) y también elementos externos, como la plataforma de distribución digital Google Play, que tiene como finalidad limitar la difusión de código malicioso.

El modelo de permisos está conformado por tres partes: la API de permisos, los permisos del *file system* y los permisos de IPC. Los permisos hacen una conexión desde un alto nivel hasta un bajo nivel, que corresponde a las capacidades del sistema operativo. Los permisos de alto nivel que accede la aplicación se encuentran especificados en el archivo AndroidManifest.xml [5]. Android actualmente cuenta con una aproximada de 295 permisos registrados en su página web de desarrolladores [26].

Kali Linux es una distribución de Linux para pruebas avanzadas de penetración y auditoria de seguridad. El sistema operativo contiene muchas herramientas

para tareas de seguridad, pruebas de penetración, análisis forense e ingeniería inversa [27]. Las herramientas de ingeniería inversa para aplicaciones Android son: Apktool [28], que permite obtener los archivos resources.arsc, classes.dex y XML de un APK; y Dex2jar [27], que provee los mecanismos para convertir archivos .dex a .class y desensamblar .dex a archivos smali.

Desde la primera aparición del *malware* en Android se ha demostrado que este sistema operativo y sus procesos de seguridad presentan vulnerabilidades. El desarrollo de software malicioso ha aumentado durante los últimos años, su finalidad es alterar el buen funcionamiento de un dispositivo u obtener información sensible de sus usuarios, proveniente de sus cuadernos de contactos, mensajes de texto, redes sociales y cuentas bancarias, etc. Entre los casos de *malware* más interesantes se encuentran: Dendroid [29], que provee un conjunto de herramientas que permite generar archivos maliciosos APK; Police Malware [30], un tipo *ransomware* que bloquea los dispositivos de sus víctimas hasta que pagan para adquirir una llave de desbloqueo; y Android. MisoSMS [31], que captura mensajes de texto y los encripta con XTEA para reducir su nivel de detección por los sistemas de seguridad.

Se han realizado distintos estudios para encontrar vulnerabilidades en Android, como: Android Fake ID [32], que describe cómo copiar la identidad única de una aplicación; y Sidewinder Targeted Attack [33], que explora cómo un atacante puede interceptar información privada desde librerías de anuncios.

Como se ha mencionado, existen desarrollos de *malware* y pruebas de vulnerabilidad que ratifican que aún existe trabajo por abordar en temas de seguridad para este sistema operativo, hecho que ha motivado a la comunidad académica a trabajar sobre alternativas que mejoren los niveles de seguridad. Para dar solución a los problemas generados por los *malware* se han propuesto: técnicas de evaluación, como los análisis estático y dinámico y la inteligencia artificial; *frameworks* para dispositivos móviles, como el *Android Security Evaluation Software* (ASEF), el *Static Android Analysis Framework* (SAAF) y Wall-Droid; sistemas de seguridad, como MobSafe, que permite realizar el análisis basado en una plataforma de computación en la nube y minería de datos en un tiempo significativo; y soluciones con cajas de aislamiento, como AASandbox, que implementa los análisis estático y dinámico sobre ambientes aislados.

El desarrollo de software malicioso en Android y la complejidad de su detección, han aumentado con el transcurrir de los años, por lo que es

necesario seguir trabajando en herramientas que permitan mejorar los niveles de seguridad [34]. Con ello en mente, intentando aportar a solución de la problemática indicada, entre 2014 y 2016, el grupo de investigación i2t de la Facultad de Ingeniería de la Universidad Icesi junto con Password Consulting Services desarrollaron un sistema antimalware para Android denominado Safe Candy [35], un sistema cliente-servidor (ver Figura 2) para el análisis de aplicaciones maliciosas Android.

El servidor se encarga de realizar los análisis estáticos y dinámicos de las aplicaciones que son suministradas por los clientes (usuarios) y por repositorios externos de libre acceso. Su método incorpora una modificación del ASEF, la aplicación de distintos AVD (*Android Virtual Devices*), el llamado a servicios de análisis para enlaces maliciosos (e.g., WOT, PhishTank y Google Safe Browsing) y la aplicación de herramientas de análisis, como: Androguard, APKTool, Dex2Jar, Tcpdump, Android Debug Safe y Drozer.

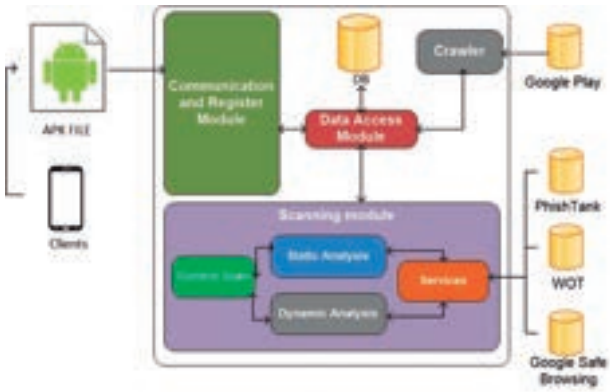


Figura 2. Arquitectura de Safe Candy

Durante el desarrollo del proyecto se realizaron pruebas de desempeño y usabilidad del sistema [36] a través de una muestra de software malicioso (integrado por once familias) y un conjunto de soluciones de seguridad, se demostró, con una detección del 100%, que la aplicación del análisis híbrido en una línea de defensa es prometedora para la detección de amenazas.

Los resultados del proyecto permitieron conocer distintas aproximaciones, tanto desde la comunidad académica, como desde la industria, para el desarrollo de soluciones de seguridad para Android. Estas experiencias permitieron evidenciar que un área de trabajo para la ciberseguridad era la aplicación de

la ciencia de datos, debido, a sus capacidades para el desarrollo de soluciones aproximadas a problemas complejos y a la identificación de patrones maliciosos a partir de la información.

La ciencia de datos y sus modelos de *machine learning* dependen de los datos de análisis y no pueden dar solución a problemas generales, sino a problemas específicos al contexto de la información, es decir, requieren de un conjunto de datos para aprendizaje y pruebas. Además, en el análisis de *malware* para dispositivos Android es necesario contar con una cantidad representativa de aplicaciones, maliciosas y benignas, para someterlas al análisis estático o dinámico para la extracción de los datos y su categorización.

Estado del arte

Navarro et al., [35] realizaron un estudio sobre algunas herramientas y marcos de trabajo para la automatización en la detección de aplicaciones Android, entre ellas: TaintDroid, Stowaway, Crowdroid y Airmid, como herramientas de análisis estático; Paranoid y DroidMOSS, como herramientas de análisis dinámico; y MobSafe, SAAF y ASEF, marcos de trabajo; e incluyeron algunos trabajos relacionados con *machine learning*. Encontraron que gran parte de las soluciones ha trabajado en análisis dinámicos con computación en la nube, ya que el procesamiento de máquinas las virtuales Android es más escalables en ese tipo de entornos, y vislumbran las primeras aproximaciones de estudios relacionados con ciencia de datos.

Tam et al., [37] recopilaron trabajos relacionados con *malware* para Android y las técnicas de análisis, estática, dinámica e hibrida, para detección de comportamientos maliciosos. Encontraron algunas desventajas y caminos por explorar, por ejemplo: que los métodos basados en firmas son ineficientes frente a los ataques de día cero y a los *malware* que cuentan con la capacidad de hacer cambios en su software; que los métodos estáticos son vulnerables a la ofuscación (e.g., a la encriptación) del código y a la inyección de software en tiempo de ejecución; y que los métodos dinámicos se ven enfrentados a distintas complejidades, uno de ellos, la necesidad de ejecutar herramientas que interactúen con la interfaz del dispositivo, también, el *malware* reconocedor de ambientes virtuales y la selección del ambiente de análisis (virtual o físico). Estos análisis, indican, están enfocados a obtener características y a identificar patrones en la información de los *malware*, entre algunas aproximaciones se

encuentran: tráfico de red, interfaces de programación de aplicaciones, grafos de dependencia, monitoreo de llamada de funciones, flujo de información, análisis comunicación entre procesos, análisis de hardware y metadatos.

Finalmente: proponen como trabajo a futuro desarrollar o mejorar los métodos de análisis, de tal manera que sean capaces de detectar comportamientos maliciosos en códigos ofuscados; mencionan, como un problema por resolver, la falta de conjuntos de software maliciosos representativos para investigaciones, en los que no solamente se debe tener en cuenta el desempeño sino la escalabilidad y la portabilidad en los métodos de análisis; identifican a la IoT como una oportunidad de trabajo para la seguridad en Android; y recomiendan la aplicación de métodos híbridos y el desarrollo de métodos de análisis dinámicos con acceso a ambientes de *hardware* real.

Para la aplicación de la ciencia de datos en el análisis de amenazas para Android, es entonces fundamental contar con muestras representativas conformadas por software de las dos clases, benigno y maligno; las muestras benignas deberían ser de distintas categorías (e.g., juegos, redes sociales y revistas) y de distintas versiones de sistema operativo; los conjuntos de *malware*, por su parte, deberían estar conformados por distintas familias, tales como: FakePlayer, DroidKungFu2 y DroidDeluxe, y distintas versiones.

Para el campo de la seguridad en Android, algunos investigadores han optado por el desarrollo de sus propios *malware* o han recolectado miles de aplicaciones a través de un script conocido como *crawler* [38].

En 2012, Zhou y Jian publicaron un conjunto de *malware* conocido como el Android Genome Project (MalGenome) [39], cuyo desarrollo cuenta con una muestra de 1.260 aplicaciones maliciosas provenientes de 49 familias distintas. Adicionalmente, estos dos investigadores habían publicado las características de su *dataset* [34], donde indicaron que el 86% de los *malware* (1.083) correspondía a versiones reempaquetadas de aplicaciones legítimas; el 36.7% tenía la capacidad de elevar sus privilegios; y el 45.3% tenía como finalidad la suscripción de servicios premium de mensajería. Además, evaluaron cuatro sistemas de seguridad, en el mejor de los casos se detectó el 79.6% de las aplicaciones maliciosas, y en el peor solo el 20.2%.

Krutz et al., [40] exploraron distintas aplicaciones y expusieron, como parte de sus resultados, un *dataset* de libre acceso. El conjunto de datos incluye información de 1.179 aplicaciones de libre descarga, 4.413 versiones de ellas y

435.680 de sus *commits* en repositorios. Adicionalmente, proveyeron resultados del uso del análisis estático sobre cada aplicación, empleando Androguard, Sonar y Stowaway.

Algunos trabajos exponen conjuntos de software malicioso para Android. Arp et al., se enfocan en Drebin [41] y ofrecen un total de 5.560 aplicaciones de 179 familias de *malware*; Allix et al., lo hacen en AndrooZoo [42] e incluyen una colección de 5.669.661 aplicaciones Android de distintas fuentes, incluyendo Google Play; y Cao et al., en DroidCollector [43], un conjunto que además de proveer 8.000 aplicaciones benignas y 5,560 muestras de *malware*, facilita sus capturas de tráfico web a través de su ejecución en ambientes virtuales.

Para la ciencia de datos es importante contar con buenas muestras para los procesos de análisis y entrenamiento de los modelos, además, debido a que la ciberseguridad puede verse enfrentada a nuevas amenazas digitales, es importante contar con un ciclo en el proceso de la analítica de la información que permita juzgar, tanto a las viejas, como a las nuevas familias de *malware*, es por esto que los modelos deberían ser probados con distintos conjuntos, como los que hemos mencionado. VirusShare [44] Contagio Mobile [45] y Malware-Traffic-Analysis.net [46]) son repositorios de libre acceso que podrían ser integrados a las pruebas generales de los modelos de predicción, constantemente registran nuevas amenazas digitales, por lo que son una fuente de muestras de *malware* para los investigadores en ciberseguridad

Hu y Jung realizaron su investigación sobre DroidDolphin [15], un *framework* de análisis dinámico que usa la interfaz gráfica de usuario (GUI, *Graphical User Interface*), el big data y el *machine learning* para la detección de aplicaciones maliciosas en Android. Su proceso de análisis consiste en extraer la información de las llamadas a la API y de trece actividades mientras la aplicación se encuentra en ejecución sobre ambientes virtuales. En su estudio se utilizó el algoritmo de *machine learning* SVM con la librería publica LIBSVM (*Library for Support Vector Machines*) publicada en [47], un conjunto de entrenamiento que consta de 32.000 aplicaciones benignas, 32.000 aplicaciones maliciosas y un conjunto de pruebas de 1.000 aplicaciones sanas y 1.000 aplicaciones maliciosas, y ha obtenido en sus resultados preliminares una precisión de 86.1% y un *F-score* de 0.875.

Hu y Jung concluyeron que: se debe tener en cuenta un equilibrio entre las aplicaciones sanas y las aplicaciones maliciosas; la calidad de la predicción depende la cantidad de aplicaciones en el *dataset*; el consumo de recursos es

alto al usar entornos virtuales; y es necesario tener en cuenta los *malware* con técnicas que detectan entornos virtuales.

Feizollah et al., evalúan cinco algoritmos de clasificación de *machine learning*: Naïve Bayes, K Nearest Neighbour, Decision Tree, Multilayer Perceptron y Support Vector Machines, con cien muestras de *malware* del Android Genome Project y doce muestras de aplicaciones sanas [48]. Su estudio parte del análisis dinámico para extraer información del tráfico de la red y así detectar las actividades maliciosas desde los dispositivos móviles. En su trabajo utilizaron el software de *machine learning* WEKA, presente en [49]. Sus resultados muestran que el mejor algoritmo fue K Nearest Neighbour, con un índice de verdaderos positivos del 99.94%, contra un 0.06% de falsos positivos.

Narudin et al., proponen un Intrusion Detection System (IDS) para detectar *malware* en Android a través de algoritmos de *machine learning*: RF, J48, MLP, NB y KNN y el uso de cuatro características del tráfico de la red: información básica, información de contenido, tiempo y conexión [38]. Trabajaron con dos *dataset* de software malicioso: uno formado por mil *malware* obtenidos del Android Genome Project y otro privado, con treinta ejemplos de códigos maliciosos de 2013. Asimismo, utilizaron veinte aplicaciones *top* gratuitas de Google Play para representar el tráfico benigno. Para el primer *dataset* se obtuvo una tasa de verdaderos positivos de 99,99% para RF, 99.98% para BN y J48, 99,65% para KNN y 88,25% para MLP; para el segundo *dataset* los mejores resultados fueron los de KNN (84.57%) y MLP (83.97%), los demás obtuvieron un resultado relativamente menor, BN (75.63%), RF (74.15%) y J48 (73.85%).

Sahs y Khan proponen un sistema de *machine learning* para la detección de *malware* en dispositivos Android con el algoritmo de clasificación One - Class SVM y el análisis estático como técnica para obtener la información de las aplicaciones [50]. En su desarrollo usaron Androguard [51] para extraer la información de los APK y el *framework* de *machine learning* scikit-learn [52]. A partir del archivo AndroidManifest.xml, desarrollaron un vector binario que contiene la información de cada permiso utilizado por cada aplicación y un diagrama de flujo de control (CFG, *Control Flow Graph*), que es una representación abstracta de un programa. El algoritmo fue implementado con un conjunto de test de 2.081 aplicaciones benignas y 91 aplicaciones maliciosas, con la información de seis kernels: uno sobre vectores binarios, otro sobre cadenas, uno sobre diagramas, uno más para los conjuntos, uno para permisos no comunes y otro para cada aplicación. Su tasa de falsos negativos fue baja, pero se presentó una alta tasa de falsos positivos.

Existen estudios que exploran otros algoritmos de *machine learning* como las redes neuronales; Ghorbanzadeh et al., proponen evaluar las vulnerabilidades de seguridad que se presentan en una estructura de permisos de una aplicación Android [53]. En su estudio utilizaron un *dataset* de 1.700 aplicaciones, entre benignas y maliciosas, y Apktool, herramienta que permite descompilar un archivo APK y obtener archivos .xml, como el AndroidManifest.xml. En la fase de entrenamiento, validación y testeo tomaron un 70%, 10% y 20%, respectivamente, de los datos del *dataset*; La red neuronal contó con una estructura de dos capas con diez neuronas y una capa de salida de 34 neuronas; al algoritmo se le suministró la información de un vector binario que representa los permisos solicitados por cada aplicación. Luego de diez experimentos consiguieron una precisión del 65%.

Yerima et al., proponen un modelo de *machine learning* que integra un clasificador Bayesiano [54], su diferencial se encuentra en la información que será suministrada al sistema de aprendizaje a partir de los detectores de: llamadas a la API, comandos, permisos, código encriptado y presencia de APK secundarias o archivos .jar. Cada detector genera variables binarias sobre las aplicaciones analizadas y, a partir de esos datos, se construye una clase que indica si el aplicativo es sospechoso o benigno, información que le será suministrada al clasificador Bayesiano. Su estudio contó con un total de dos mil aplicaciones, provenientes de tiendas oficiales y de terceros, mil *malware* y mil aplicaciones benignas. Adicionalmente, se empleó un conjunto de: 1.600 aplicaciones (benignas y *malware* en partes iguales) para la fase de entrenamiento y de 400 (200 benignas, 200 *malware*) para la etapa de testeo. Para un total de veinte características y 1.600 aplicaciones se obtuvo una tasa de verdaderos positivos de 90.6%, una tasa de falsos negativos de 0.094%; con una precisión de 93.5% y una probabilidad AUC de 97.22%.

Peiravian y Zhu proponen combinar los permisos y los llamados a la API en conjunto con los métodos de *machine learning* para la detección de aplicaciones maliciosas en Android [25]. Su estudio usó la técnica de análisis estático aplicando Apktool para la ingeniería inversa de los APK, para así obtener información de importancia de cada aplicación. Como producto de su trabajo proponen un *framework* para el análisis y la clasificación de aplicaciones Android basado en técnicas de *machine learning*, el cual combina los permisos solicitados y el comportamiento de la API de una aplicación para construir un clasificador que detecte aplicaciones maliciosas. El experimento contó con un total de

2.510 APK (1.260 *malware*, algunos tomados del Malware Genome Project; y 1.250 aplicaciones benignas), adicionalmente, se evaluaron los algoritmos de clasificación: SVM, DT y Bagging, contra tres *datasets*: uno para permisos, otro para llamadas de la API y otro de cadenas con permisos. El algoritmo con mejor desempeño fue Bagging, con él obtuvieron una precisión de 96.39% en la detección de *malware*.

Metodología

FRAMEWORK DE ANÁLISIS ESTÁTICO

El marco de trabajo desarrollado (Figura 3) incluye los siguientes siete pasos:

- Se recolectó un total de 644 APK.
- Se crearon dos colecciones: la primera compuesta por 279 aplicaciones de bajos privilegios del *dataset open source* mencionado en el estado del arte [40] y una selección aleatoria de 279 *malware* del MalGenome; la segunda, conformada por 43 aplicaciones descargadas de Google Play y una muestra aleatoria de 43 *malware* no seleccionados del MalGenome.
- Se utilizó la herramienta Apktool 2.0.3 para obtener los archivos AndroidManifest.xml de las dos colecciones.
- Se desarrollaron unas herramientas en Python con el fin de crear los vectores binarios que corresponden a los permisos accedidos por cada aplicación

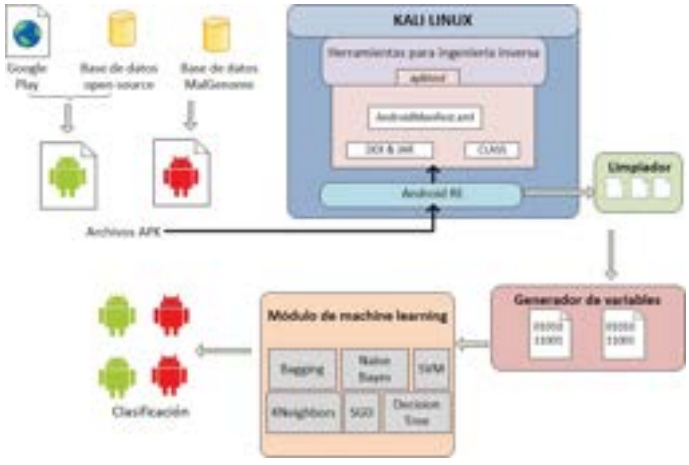


Figura 3. Marco de trabajo para el análisis estático

y a la etiqueta que identifica si el aplicativo es benigno o maligno, como se muestra en la sección "Generador de características"). Cada colección está conformada por dos *datasets* que contienen los permisos accedidos por cada aplicación en binario.

- Con el *dataset* binario que representa la primera colección se crearon dos particiones con el 71% y el 29% de los datos para el entrenamiento y testeo de los algoritmos de *machine learning*.
- Se entrenó y probó cada algoritmo de *machine learning*.
- Se utilizó el segundo *dataset* binario para evaluar la capacidad de generalización de los modelos ya entrenados.

GENERADOR DE CARACTERÍSTICAS

A través del desarrollo de un analizador de permisos se evaluaron los 664 AndroidManifest.xml contra una lista de 330 permisos del proyecto AndroGuard [51]. El analizador se encargó de crear una variable binaria (1) por cada permiso de la lista, el valor que le corresponde depende de si el permiso de la lista se encuentra en el archivo AndroidManifest.xml analizado.

$$R_i = \begin{cases} 1, \text{ Si el analizador detectó un permiso accedido} \\ 0, \text{ en otro caso} \end{cases} \quad (1)$$

Luego de analizar los permisos y antes de pasar a evaluar otro archivo AndroidManifest.xml, se creó una etiqueta que corresponde a la clasificación del aplicativo:

$$C_i = \begin{cases} 1, \text{ Si el aplicativo es malicioso} \\ 0, \text{ en otro caso} \end{cases} \quad (2)$$

A cada aplicación se le asignó un vector binario definido por $V = (R_1, R_2, ..., R_{330}, C)$, que contiene la información de los permisos y la etiqueta de clasificación.

Para evaluar el desempeño de los clasificadores en el contexto del problema se utilizaron las métricas: exactitud (*accuracy*), matriz de confusión, precisión y recuperación (*recall*), y la área bajo la curva (AUC). Se utilizó Python 3.4 como lenguaje de desarrollo y las herramientas de *machine learning* de la librería scikit-learn 0.17.1 [52]. Los algoritmos de clasificación seleccionados para el experimento fueron: Naïve Bayes, Bagging, K Nearest Neighbor (KNN), Support

Vector Machines (SVM), Stochastic Gradient Descent (SGD) y Decision Tree (DT). Se evaluaron algunos de los algoritmos con distintas configuraciones, así:

Naïve Bayes se utilizó para distribuciones Gaussianas y distribuciones de Bernoulli. Finalmente, con un desempeño en la clasificación del 90%, el mejor resultado fue para distribuciones de Bernoulli (Tabla 8).

Tabla 8. Desempeño de los clasificadores de Naïve Bayes

	Medida	Precisión		Recuperación		f1 score		Exactitud
Distribución	Ci =	0	1	0	1	0	1	
Gaussiana		0,90	0,76	0,79	0,88	0,84	0,82	0,84
De Bernoulli		0,93	0,88	0,88	0,92	0,90	0,90	0,90

Con Bagging se realizaron cuatro pruebas con distintas configuraciones y utilizando los clasificadores DT y KNN. Dos configuraciones tomaron muestras del 30% de los datos y 20% de las características, las otras del 90% de los datos y el 90% de las características. Finalmente, la configuración de Bagging con los mejores resultados fue Decision Tree para muestras con el 90% de los datos y 90% de las características (Tabla 9).

Tabla 9. Desempeño de los clasificadores de Bagging

	Medida	Precisión		Recuperación		f1 score		Exactitud
Clasificador	Datos/Car. Ci =	0	1	0	1	0	1	
KNN	30%/20%	0.94	0.88	0.88	0.93	0.91	0.90	0.91
	90%/90%	0.94	0.88	0.88	0.93	0.91	0.90	0.91
DT	30%/20%	0,93	0,80	0,82	0,91	0,87	0,85	0,87
	90%/90%	0,9	0,95	0,95	0,9	0,92	0,93	0,93

Para probar K Nearest Neighbour se emplearon valores 2, 3, 4 y 6 para k. Los resultados mostraron que el desempeño de la clasificación aumentó hasta un valor de k = 4 y empezó a disminuir con un valor de k > 6 (Tabla 10).

Tabla 10. Desempeño de los clasificadores de KNN

	Medida	Precisión		Recuperación		f1 score		Exactitud
K	Ci =	0	1	0	1	0	1	
2		0,93	0,94	0,94	0,93	0,93	0,93	0,93
3		0,90	0,95	0,95	0,90	0,92	0,93	0,93
4		0,94	0,95	0,95	0,94	0,94	0,94	0,94
6		0,95	0,89	0,89	0,95	0,92	0,92	0,92

Para SVM se realizaron dos pruebas, una con una función lineal, otra con una función de base radial. Los resultados mostraron un mejor desempeño cuando se utilizó la función lineal (Tabla 11).

Tabla 11. Desempeño de los clasificadores de SVM

	Medida	Precisión		Recuperación		f1 score		Exactitud
Tipo de fución	Ci =	0	1	0	1	0	1	
Lineal		0,93	0,95	0,95	0,93	0,94	0,94	0,94
RBF		0,93	0,88	0,88	0,92	0,90	0,90	0,90

Stochastic Gradient Descent fue entrenado con una función de perdida *hinge loss* con una penalidad de 12 y Decision Tree con la configuración por defecto de sckit-learn. DT obtuvo mejores resultados (Tabla 12).

Tabla 12. Desempeño de los clasificadores de SGD y DT

	Medida	Precisión		Recuperación		f1 score		Exactitud
Algoritmo	Ci =	0	1	0	1	0	1	
SGD		0,91	0,93	0,92	0,91	0,92	0,92	0,92
DT		0,93	0,95	0,95	0,91	0,94	0,94	0,94

Revisando los mejores resultados (Tabla 13), es claro que los algoritmos de NB y SGD presentan el menor desempeño de la clasificación, con un 90% y 92%, respectivamente; Bagging continúa en el listado con una buena detección de software malicioso del 95% y un desempeño de clasificación del 93%; finalmente, los algoritmos de KNN, SVM y Decision Tree cuentan con un desempeño de clasificación del 94%, un comportamiento similar en los resultados de los verdaderos positivos, contra los falsos negativos en el área bajo la curva (Figura 4).

Tabla 13. Desempeño individual de los seis clasificadores

	Medida	Precisión		Recuperación		f1 score		Exactitud
Algoritmo	Ci =	0	1	0	1	0	1	
Naïve Bayes		0,93	0,88	0,88	0,92	0,90	0,90	0,90
Bagging		0,90	0,95	0,95	0,90	0,92	0,93	0,93
K Nearest Neighbor		0,94	0,95	0,95	0,94	0,94	0,94	0,94
Support Vector Machines		0,93	0,95	0,95	0,93	0,94	0,94	0,94
Stochastic Gradient Descent		0,91	0,93	0,92	0,91	0,92	0,92	0,92
Decision Tree		0,93	0,95	0,95	0,91	0,94	0,94	0,94

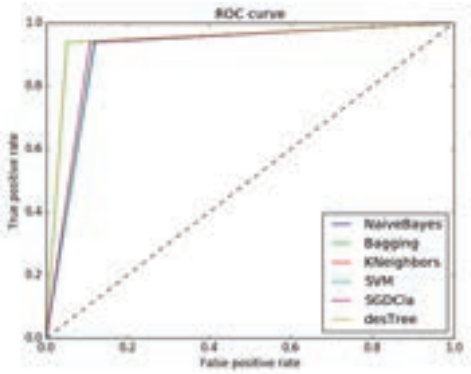


Figura 4. Resultados: área bajo la curva

ANÁLISIS DE PERMISOS

El análisis de los permisos de las aplicaciones sanas y de los *malware* del primer *dataset* (Tabla 14) muestra que los códigos maliciosos acceden a una mayor cantidad de ellos.

Tabla 14. Permisos accedidos por las aplicaciones - *Dataset* I

Aplicaciones Sanas >= 10	Malware >= 100
ACCESS_COARSE_LOCATION	WRITE_SMS
ACCESS_FINE_LOCATION	WRITE_EXTERNAL_STORAGE
ACCESS_NETWORK_STATE	RECEIVE_BOOT_COMPLETED
ACCESS_WIFI_STATE	READ_SMS
BLUETOOTH	READ_PHONE_STATE
CAMERA	INTERNET
CHANGE_WIFI_STATE	ACCESS_WIFI_STATE
GET_ACCOUNTS	ACCESS_NETWORK_STATE
INTERNET	
READ_CONTACTS	
READ_EXTERNAL_STORAGE	
READ_PHONE_STATE	
RECEIVE_BOOT_COMPLETED	
VIBRATE	
WAKE_LOCK	
WRITE_EXTERNAL_STORAGE	

Para evaluar la capacidad de generalización del mejor modelo, se descargó de Google Play un total de 43 aplicaciones de distintas categorías (el segundo *dataset* mencionado) y se seleccionó de forma aleatoria una muestra de 43 *malware* no utilizados del MalGenome. Analizando los permisos de las aplicaciones descargadas, más de 10 acceden a 24 permisos y entre estos se encuentran los mismos de las aplicaciones benignas del primer *dataset* (Tabla 15).

Tabla 15. Permisos accedidos por las aplicaciones descargadas de Google Play

Aplicaciones descargadas >=10
ACCESS_COARSE_LOCATION
ACCESS_FINE_LOCATION
ACCESS_NETWORK_STATE
ACCESS_WIFI_STATE
BLUETOOTH
CAMERA
CHANGE_WIFI_STATE
GET_ACCOUNTS
GET_TASKS
INTERNET
MANAGE_ACCOUNTS
MODIFY_AUDIO_SETTINGS
READ_CONTACTS
READ_EXTERNAL_STORAGE
RECEIVE_BOOT_COMPLETED
READ_PHONE_STATE
SYSTEM_ALERT_WINDOW
USE_CREDENTIALS
VIBRATE
READ_EXTERNAL_STORAGE
WAKE_LOCK
WRITE_EXTERNAL_STORAGE
WRITE_SETTINGS
INSTALL_SHORTCUT

Para evaluar la capacidad de generalización se utilizó el segundo *dataset* y con él se evaluó cada modelo entrenado. Los resultados obtenidos (Tabla 16) muestran que, a diferencia del primer experimento, los modelos de Bagging y

Naïve Bayes obtienen los menores resultados, tanto en el desempeño, como en la clasificación de aplicaciones benignas y malignas, mientras que: el modelo Decision Tree obtiene un 85% en el desempeño de la clasificación; SVM y KNeighbors, un desempeño del 87%; y SGD 92%. Es decir que el modelo que mejores resultados presentó, con un buen grado en la clasificación y buen desempeño es el que tiene como clasificador a SGD (Figura 5).

Tabla 16. Desempeño en la prueba de generalización

	Medida	Precisión		Recuperación		f1 score		Exactitud
Algoritmo	Ci =	0	1	0	1	0	1	
Naïve Bayes		0,56	0,91	0,86	0,67	0,68	0,77	0,79
Bagging		0,65	0,88	0,85	0,72	0,74	0,79	0,79
K Nearest Neighbor		0,74	0,95	0,94	0,79	0,83	0,86	0,87
Support Vector Machines		0,79	0,93	0,92	0,82	0,85	0,87	0,87
Stochastic Gradient Descent		0,88	0,95	0,95	0,89	0,92	0,92	0,92
Decision Tree		0,81	0,88	0,88	0,83	0,84	0,85	0,85

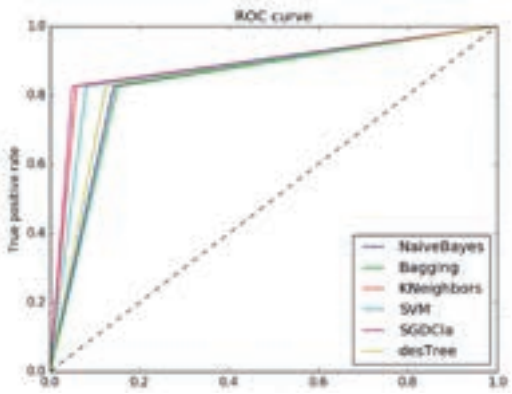


Figura 5. Generalización: área bajo la curva

Trabajo futuro

Los resultados obtenidos de la evaluación de los seis clasificadores de *machine learning* para la detección de *malware* en Android a partir de permisos accedidos, permiten afirmar que es posible preparar algoritmos de clasificación que tengan la capacidad de generalizar a otro conjunto de datos, a partir del *dataset* de entrenamiento utilizado. Para dar continuidad a los resultados obtenidos

en esta investigación, se recomienda: estudiar la aplicación de otros métodos de inteligencia artificial para detección de *malware* en Android (e.g., sistemas inmunes); evaluar otros algoritmos de *machine learning* de scikit-learn y explorar su utilidad; explorar otras características de Android para el entrenamiento y prueba de los clasificadores; explorar el uso de otros *datasets*, tanto de aplicaciones sanas, como de aplicaciones malignas; realizar un estudio de los permisos accedidos por los dos *datasets*; incluir en el procedimiento un ciclo de retroalimentación que le permita al sistema aprender de las malas predicciones; desarrollar mecanismos de defensa contra modelos de *machine learning* adversarios; y evaluar el desempeño del aplicativo SafeCandy con la implementación del módulo inteligente.

Ciberseguridad en aplicaciones web

El incremento casi exponencial de la tecnología en los últimos años ha hecho que los ataques y las vulnerabilidades en las páginas web aumenten con el tiempo [1-2]. En consecuencia, los desarrolladores de los aplicaciones web –y sus aplicaciones– deberían actualizarse casi al ritmo de la aparición de una nueva tecnología. Pero esto en la práctica es casi imposible.

Para 2018, Internet conecta alrededor de tres mil millones de usuarios en todo el mundo [Cerf14] y es un medio que permite, tanto a las personas, como a las entidades, intercambiar información, incluso información sensible, y acceder a páginas web. El uso de la tecnología se ha incrementado significativamente y así mismo ha aumentado la cantidad de vulnerabilidades y de ataques cibernéticos [55], [56], lo que ha hecho de la seguridad una preocupación importante, tanto para los usuarios, como para los investigadores [20].

Las redes, ese conjunto de nodos conectados entre sí que proporcionan datos que son transportados de un lugar a otro con el fin de proporcionar comunicación entre uno o más dispositivos, acceso e información al medio (personas, entidades, sociedades y servidores) [57], han sido un elemento clave para la conectividad entre las tecnologías y los humanos.

Open Web Application Security Project (OWASP), entidad internacional cuya misión es que las organizaciones desarrollen, adquieran y mantengan aplicaciones confiables, identifica anualmente las diez mayores riesgos para la segudad en aplicaciones, para 2017 [58], su listado incluye: inyección, pérdida de autenticación, exposición de datos sensibles, Entidades Externas XML (XXE), pérdida del control de acceso, configuración de seguridad incorrecta, Cross-Site Scripting (XSS), deserialización insegura, uso de componentes con vulnerabilidades conocidas y registro y monitoreo insuficientes (ver Tabla 17).

Tabla 17. OWASP Top Ten de los riesgos para la seguridad 2017 [58]

Vulnerabilidad	Descripción
Inyección	Envío de datos no confiables como parte de un comando o consulta que pueden ejecutar comandos involuntarios o acceder a datos sin la debida autorización.
Pérdida de autenticación	Una incorrecta implementación de las funciones de autenticación y gestión de sesiones permite al atacante asumir la identidad de los usuarios.
Exposición de datos sensibles	Datos sensibles (información financiera, registros de salud) protegidos de manera inadecuada permiten acceder y hacer mal de ellos o modificarlos.
Entidades externas XML (XXE)	Procesadores antiguos o mal configurados evalúan referencias a entidades externas en documentos XML, esto puede servir para revelar archivos, escanear puertos de la LAN, ejecutar código remotamente o realizar ataques de DoS.
Pérdida de control de acceso	Cuando las restricciones a los usuarios autenticados no son bien aplicadas, los atacantes pueden acceder a funcionalidades, datos y cuentas de terceros, ver archivos sensibles, modificar datos, cambiar derechos de acceso y permisos, etc.
Configuración de seguridad incorrecta	Este problema se puede generar al establecer la configuración de forma manual, ad hoc o por omisión, y por la falta de configuración (e.g., S3 buckets abiertos).
Cross-Site Scripting (XSS)	Una aplicación toma datos no confiables y los envía a un navegador web sin validarlos y codificarlos apropiadamente o actualiza una página web con los datos suministrados por el usuario utilizando una API que ejecuta JavaScript en el navegador, con lo que es posible secuestrar una sesión, modificar el sitio o redireccionar al usuario hacia un sitio malicioso.
Deserialización insegura	Ocurre cuando una aplicación recibe objetos serializados dañinos que pueden ser manipulados o borrados por el atacante para: realizar ataques de repetición, inyecciones, elevar sus privilegios y ejecutar código en el servidor.
Componentes con vulnerabilidades conocidas	Algunos módulos (e.g., bibliotecas, *frameworks*) se ejecutan con los mismos privilegios que la aplicación, si se explota un componente vulnerable, el atacante puede provocar la pérdida de datos o tomar el control del servidor.
Registro y monitoreo insuficientes	Junto con la falta de respuesta rápida a incidentes permiten mantener el ataque, pivotear a otros sistemas y manipular, extraer o destruir datos.

En su reporte de riesgos para la seguridad de 2018, White Hat Security [56] destaca cómo, aunque hay mejoras en los sectores de finanzas, salud y venta al detal, tradicionalmente muy vulnerables a los ciberataques, a nivel global la vulnerabilidad de las grandes empresas continúa en aumento: dos de sus tres indicadores: grandes ventanas de exposición y altos tiempos de solución de los incidentes crecen 33% y 2% respecto de 2017, mientras que la tasa de casos solucionados permanece constante.

Cada vez son más frecuentes las noticias sobre ataques de denegación de servicios (DoS, *Denegation of Services*). Uno de los más notables fue el que sufrió DynDNS, proveedora de importantes empresas (e.g., Twitter, Spotify y Reddit), cuyos URL fueron redireccionados, de tal manera que cuando un usuario intentaba cargar la página, esta se reportaba "caída" [59].

En Colombia, se destaca el ataque a la página de la Registraduría Nacional del Estado Civil [60] en 2016, justo unos días antes de la votación de un plebiscito crítico para el país (la aceptación o no de los acuerdos de paz con un grupo guerrillero). El ataque no implicó robo de información sino que se manifestó con cambios en la visualización de la página (defacement) que la llevó a su inactividad y genero desconfianza en los votantes.

La detección de páginas maliciosas no solo es un aspecto muy importante para la protección de las entidades y las personas frente a estas violaciones de seguridad, sino que además ayuda a mitigar problemas futuros relacionados con agentes externos que causan serios daños dentro de la lógica del sistema. Por este motivo, durante 2016 el grupo de investigación i2t de la Facultad de Ingeniería de la Universidad Icesi, junto con Password Consulting Services, trabajaron en el desarrollo de una solución, denominada snif, dirigida a la protección de sitios web contra ciberataques que tuvieran como finalidad específica realizar cambios no autorizados (defacement).

Snif es un sistema basado en la detección por anomalías que permite supervisar y corregir el defacement en páginas web [61]. Una parte de los procesos del sistema es la generación de hash a partir de atributos HTML de la página web (e.g., href, src y número de enlaces). Cada hash generado es único y dependiente de los elementos que usualmente no tienen una frecuencia de cambio muy alta, es decir de aquellos que el administrador del sitio web no altera constantemente. Otra actividad es el monitoreo de la llave, con la cual se revisa si el hash corresponde a los elementos del sitio web y genera las alarmas necesarias, si se presenta alguna inconsistencia. La eficiencia en la detección y corrección del defacement de sniff es de alrededor de 35 segundos para 415 atributos y 11 segundos para una página web con 149 elementos HTML.

Durante el desarrollo del proyecto se evidenció que en Colombia existen medidas legales, como el CONPES 3701 [62], e instituciones, como el Grupo de Respuesta a Emergencias Cibernéticas de Colombia (colCERT) [63], que velan por la ciberseguridad, cuya existencia ha motivado a las organizaciones públicas y privadas a implementar mecanismos de seguridad. Sin embargo, detectó también que algunas entidades públicas contratan a agentes externos para la administración de su infraestructura, delegándo en ellos gran parte de los procesos y con ello su control. Esto podría generar problemas, ya que se comparten datos sensibles y responsabilidades frente a las vulnerabilidades tecnológicas y humanas [64].

El proyecto también encontró una relación entre la prevención y mitigación de ataques de este tipo con la aplicación de la ciencia de datos. El problema puede ser enfrentado desde distintos frentes a través de la aplicación de los análisis estático y dinámico para la abstracción de la información. En algunos casos se optó por recolectar información de los ciberataques en tiempo de ejecución a través de entornos controlados con vulnerabilidades, por ejemplo, el acceso a sitios web con contenido malicioso a través de *honeypots*.

Actualmente, gran parte de los dispositivos –por ejemplo, el IoT y los dispositivos móviles–, requiere para su operación de acceso a la Internet y de intercambio de información con agentes externos. Este incremento de conectividad a su vez requiere de mejores mecanismos de defensa contra aplicaciones web con contenido malicioso. El proyecto de investigación buscaba, en una primera instancia, la detección de sitios web maliciosos a partir de patrones en su información, para que a futuro se puedan abordar, en mayor detalle, los ciberataques web en las tecnologías que se comunican a través de la red.

Estado del arte

Roesh [65] explica la utilidad de Snort. un *sniffer* y sistema de detección de intrusos que utiliza una serie de reglas configuradas en su base datos para la detección de contenido malicioso en la red de un sistema. Durante la captura del tráfico de red, Snort extrae algunos elementos que luego compara contra reglas configuradas (ilustradas en [66]), como por ejemplo, P2P, puertas traseras, ataques de denegación de servicio y virus informáticos.

Howard et al., trabajan con PSigene [67], un sistema de detección de intrusos que permite generar y almacenar automáticamente firmas de peticiones, tanto maliciosas, como benignas. Sus resultados muestran que el desempeño de PSigene es más eficiente que Bro y Snort (86.23% y 90.52%, respectivamente, para nueve firmas), pero no supera a ModSecurity.

Bartoli, Davanzo y Medvet proponen un marco de trabajo denominado Goldrake para el análisis de cambios no autorizados sobre páginas web con alto contenido dinámico [68]. Goldrake utiliza la técnica de detección por anomalías con un valor agregado, un servicio de monitoreo que no requiere de instalación sobre la infraestructura del sitio web a analizar. Los resultados de la evaluación muestran buen desempeño del prototipo en cuanto a las tasas de falsos positivos y falsos negativos.

Zhong et al., utilizan una estructura abstracta de parámetros para el mapeo de las peticiones HTTP [13]. Su propuesta tiene una fase de entrenamiento que se compone por tres actividades: transformación, filtro y determinación del perfil. El desempeño del método obtuvo una buena tasa de falsos positivos.

Xu et al., comparan una herramienta para la detección de enlaces maliciosos basada en operaciones AND, OR, XOR y agregación de datos, contra cuatro algoritmos clasificadores de *machine learning*: regresión logística, Naïve Bayes, SVM y C4.5 Decision Tree (implementación J48) [69]. Para la extracción de características en los URL se usó un crawler (rastreador) y TCPDUMP, con ellos se identificaron los URL activos y se extrajo su tráfico. Con esta información, la actividad se centró en obtener las características de la capa de aplicaciones (información del URL) y de la capa de red (datos del tráfico web). Las características obtenidas de cada una de ellas se presentan en las Tablas 18 y 19.

La validación de las herramientas se realizó en un entorno controlado en un lapso de 37 días, durante este proceso obtuvieron aproximadamente 124 variables que se analizaron midiendo su correlación y precisión por medio de tres algoritmos disponibles en Weka Toolbox: Principal Components Analysis (PCA), un algoritmo de aprendizaje no supervisado que no muestra fuertes indicios para la selección de las características de sitios maliciosos; CfsSubsetEval, útil para encontrar una baja correlación y una alta precisión en las características; e InfoGainAttributeEval, útil para determinar las características más importantes dependiendo de su ratio de adquisición y de la información que provee.

Con base en estos algoritmos, realizaron un análisis comparativo usando los cuatro clasificadores citados (regresión logística, NB, SVM y C4.5 DT) y los métodos de agregación de datos AND, OR y XOR, los mejores resultados los obtuvo XOR con los algoritmos subset e Infogain para el clasificador C4.5 Decision Tree. Encontraron además que variables longitud del URL, nombre del servidor y duración eran las más significativas y la base de su herramienta (Cross-Layer Detection).

Zhong et al., [13] proponen un método de tres pasos: transformación, filtrado y determinación del tipo de perfil. Para el entrenamiento utilizaron datos obtenidos de un *dataset* real, es decir que los URL no se encontraban en un entorno controlado. En el primer paso, transformación, definieron varias clases de caracteres básicos como alfabeto, número y símbolos –este método de

Tabla 18. Características de la capa de aplicaciones

Localización	Característica	Observación
Contenido del URL	Longitud	Las aplicaciones malignas tienen un promedio mayor que las benignas.
	Cantidad de caracteres especiales (e.g., ¿, -, _, =, %)	Las aplicaciones malignas tienen un promedio mayor que las benignas.
	Presencia de direcciones IP.	
Información delacabecera de HTTP	Charset	Tipo de codificación (e.g, ANSI, ISO-8859-1 y UTF8).
	Servidor	Los más utilizados por las páginas web maliciosas son Apache, Microsoft IIS y nginx.
	Cache control	El uso de no-cache, *private*, *public* y *max-age* es mayor en los sitios benignos.
	Longitud del contenido	
Información de la Whois y el DNS.	Fecha de actualización del *webserver.*	
	Fecha de registro del *webserver*.	
	País de registro del *webserver*.	Los más representativos fueron US, NL y AU, el número de registros en ellos fue mayor para los enlaces benignos.
	Estado de registró del *webserver*.	
	Dominio	El porcentaje de redireccionamiento de los dominios es mayor en los sitios web maliciosos.
Contenido de la web	Número de redirecciones	Total de redirecciones embebidas dentro de una URL.
	URL externas embebidas	Número de enlaces que se encuentran dentro de otros y que hacen un llamado a recursos externos.
	Longitud de contenido valida	Indicador de la consistencia entre el tamaño del contenido de la cabecera HTTP y el de la página web.
	Número de cadenas largas	Número de cadenas de más de 50 caracteres que se utilizan en el código *JavaScript* embebido en una entrada URL.
	Nímero de Iframe	Cantidad de Iframe encontrados en la página.
	Longitud de los Iframes	Longitud de los Iframes encontrados en la página.
J48	Funciones JavasScript que contiene la página (#)	Se revisaron, entre otras: eval(), escape() y unescape().

Tabla 19. Características de la capa de red

Localización	Característica	Observaciones
Características remotas basadas en los atributos del servidor.	Intercambio de conversaciones TCP (#)	Total de paquetes TCP enviados a un servidor remoto por el crawler.
	Número de puertos TCP remotos	Cantidad de distintos puertos TCP que el webserver usa durante la conversación con el *crawler*.
	Número de IP remotos	Cantidad de direcciones IP remotas que se conectan por el *crawler*.
Comunicación de *crawler- server*	Cantidad de Bytes app	Numero de bytes de la aplicación de datos enviados por el *crawler* al servidor web remoto.
	Número de paquetes UDP	Cantidad de paquetes UDP generados cuando el *crawler* visita una URL.
	Numero de paquetes urgentes TCP	Cantidad de paquetes con indicador URG.
	Source app packets (#)	Paquetes enviados por el *crawler* al servidor remoto (mayor, en promedio, en sitios web maliciosos).
	Número de paquetes de app remoto	Cantidad de paquetes enviados por el *webserver* remoto al *crawler* (mayor, en promedio, en sitios web maliciosos).
	Source app bytes	Cantidad de bytes que se dan en la comunicación del *crawler* al sitio remoto.
	App bytes remotos	Cantidad de bytes que se dan en la comunicación de sitio remoto al *crawler*.
	Duración	De la respuesta de conexión de la página web (es la característica más fuerte de todo el proceso.
	Promedio de ratio de paquetes locales	Cantidad de paquetes locales sobre la duración.
	Promedio de ratio de paquetes remotos	Cantidad de paquetes remotos sobre la duración.
	Paquetes de aplicación	Número de paquetes IP enviados en la comunicación del *crawler* al sitio web.
Flujo de *crawler*-DNS	Consultas al DNS (#)	
	Tiempo de respuesta del DNS	
	Flujo Iat	Tiempo acumulado entre la llegada de flujos consecutivos.
	Número de flujos	Flujos UDP o TCP generados en un ciclo de vida para que el crawler descargue el contenido web de un URL.
	Duración del flujo.	

extracción de caracteres es muy usado para la detección y el agrupamiento de enlaces maliciosos– y generaron una secuencia de clases a partir de un valor usando un algoritmo de coincidencia de prefijos incrementales. En la etapa de filtrado calcularon la frecuencia de cada clase de los tipos de caracteres identificados utilizando un alfa entre 0 y 1. Finalmente, en la tercera etapa encontraron que se pueden producir falsas alarmas de URL malignos cuando aplican una estricta restricción estructural a estos parámetros, como mensajes de chat de una aplicación web de red social, por lo que introdujeron un tipo de perfil llamado "conjunto de clases de caracteres", el cual indica un conjunto de clases de caracteres sin un orden específico.

Para la detección, los investigadores extrajeron los parámetros de las solicitudes HTTP que deben ser revisados. El valor de cada parámetro se transformó de la misma manera que en el primer paso y se realizó el cálculo de la similitud de un valor de parámetro, para cada uno de los ellos. Dependiendo de su perfil se clasificaron como malignos o benignos. Sus experimentos y evaluaciones se realizaron de la siguiente manera: toman un *dataset* y lo dividen en dos partes, la primera para entrenamiento, la segunda para detección; de la primera se eliminan las peticiones cuyos URL lanzan respuestas del tipo 404 o 500, debido a que posiblemente corresponden a las exploraciones para encontrar servidores web vulnerables; en la segunda se coloca una etiqueta que indica si la solicitud es o no un ataque, con base en los resultados de detección de múltiples aplicaciones comerciales de la aplicación web y el análisis manuales realizados; finalmente, realizan la detección de ataques con los datos de prueba utilizando su propuesta.

Los investigadores encontraron cuatro tipos de ataques: *SQL injection*, inyección de comandos, inclusión remota de archivos (RFI, *Remote File Inclussion*) y XSS (*Cross-Site Scripting*) y usaron las tasas de detección y de falsos positivos como indicadores de evaluación de precisión de detección. Es importante resaltar su recomendación de no usar parámetros con valores exactos para hacer la detección de entradas maliciosas porque puede causar falsas alarmas en el análisis del URL.

Mohaisen presenta un sistema que identifica cualquier contenido malicioso o benigno en un sitio web utilizando análisis estático y dinámico [70] y utilizando algoritmos de *machine learning*, si el algoritmo identifica una página como maliciosa, el sistema es tiene la capacidad de predecir su categoría. Las categorías evaluadas por este investigador fueron: *injection*, *exploit type*, *exploit kit*

type, *redirection type*, *defacement*, *obfuscation type*, *malicious executable or archive type* y *server side backdoor type*.

Mohaisen utilizó un sistema alfa que hace el papel de análisis dinámico para analizar el contenido web ingresado. Con él analizó un sistema llamado "archivos transferidos" para determinar si una página era maliciosa o no. Para la clasificación de cada aplicación web acudió a: metadatos (características HTTP); características derivadas de la URI que se asocian con los archivos transferidos (e.g., hostname, paquete, consulta, tamaño del URL, cantidad de características atípicas); archivos de transferencia (características como la dirección IP y el numero de redirecciones); características que se derivan del nombre del dominio (e.g., tiempo de la información, fecha de creación, fechas de actualización o de expiración); y características derivadas de la dirección IP (e.g., país, ciudad, región, organización).

Mohaisen describe su diseño así: el sistema recibe datos generados por el sistema alfa y determina, por cada archivo transferido, una etiqueta que indica si es benigno o maligno y después, trabaja en el entrenamiento y en los modos de operaciones. Para evaluar las características mencionadas, realizó una matriz para entrenamiento y otra para prueba, las que normalizó asignando valores entre 0 y 1 para los datos numéricos, y binarizando las características categóricas. Para la clasificación de los URL uso el algortimo SVM basado en SGD y un clasificador binario para cada tipo de vulnerabilidad. Es importante mencionar que trabajó con veinte mil páginas web, divididas a partes iguales entre malignas y benignas. Para la implementación de los algoritmos usó 10-fold cross validation, para que los *datasets* fueran partidos aleatoriamente, y usó una parte para el entrenamiento y la otra para las pruebas. Concluyó que las páginas web maliciosas tenían una concurrencia del 93% de vulnerabilidades *injection* y *server-side backdoor*.

Metodología

La investigación comenzó con un estudio del estado del arte y del marco teórico en el cual se analizaron los trabajos que incluían los algoritmos de *machine learning* para identificación de páginas web maliciosas, cuyos resultados se presentan en la sección anterior. Del análisis realizado, se encontró que los trabajos evaluados no exponen un conjunto de datos que permita realizar la evaluación de los resultados, es decir, no existe el *dataset* de características

expuesto para el entrenamiento de los algoritmos de *machine learning*, por lo que se procedió a aplicar un marco de trabajo (Figura 6) que tiene como objetivo generar los datos para el desarrollo del experimento, el cual incluye el entrenamiento y la evaluación de los clasificadores de *machine learning*.

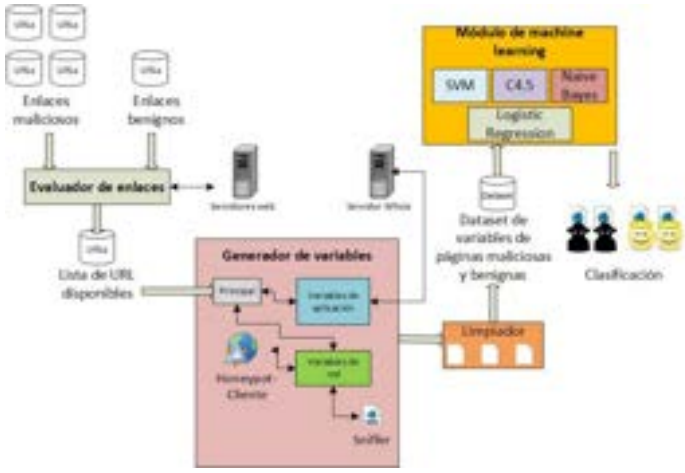


Figura 6. Marco de trabajo para detección de páginas maliciosas

El marco de trabajo se dividió en cuatro fases: la primera se encargó de la identificación y la adquisición de las fuentes e incluyó los enlaces de páginas benignas y malignas; la segunda consistió en el desarrolló de una herramienta para el preprocesamiento de la información encargada de verificar si las páginas estaban activas o no y, en caso de estarlo, de obtener los datos de sus capas de aplicación y de red; la tercera, denominada fase de procesamiento, tenía como objetivo desarrollar un *dataset* de características, para el cual se consiguieron 400; y la cuarta, dedicada a la selección de los algoritmos de clasificación y de las medidas de evaluación para el desarrollo del experimento.

FASE 1. ADQUISICIÓN DE LOS DATASETS

Se identificó como algo primordial obtener los *datasets* referenciados en los artículos, sin embargo, al buscarlos fue evidente que el acceso a ellos no es fácil, pues no suelen estar disponibles *open source*. En muchos artículos, aunque se indica la fuente de los URL malignos y benignos extraídos, no se incluye información acerca de la conformación del *dataset* utilizado. En vista de esto se decidió realizar la extracción de los *datasets* de páginas que brindaran *blacklists* pensando en su actualización y suponiendo que probablemente los URL no habían sido estudiados. Se extrajeron 185.181 enlaces malignos

desde machinelearning.inginf.units.it/data-and-tools/hidden-fraudulent-urls-dataset, malwaredomainlist.com y zeuztacker.abuse.ch; y 345.000 enlaces benignos desde el repositorio de Github (https://github.com/faizann24/Using-machine-learning-to-detect-malicious-URLs.git). Para la mejor identificación de sus URL, los *datasets* se organizaron así: Los *dataset* 0 a 5 con: 2.569, 11, 17, 75 y 185.181 URL malignos; y un *dataset* con 345.000 URL benignos.

FASE 2. PREPROCESAMIENTO DE LOS DATOS

El preprocesamiento de la información adquirida tuvo en cuenta tres etapas: verificación, modificación y extracción de la información.

En la primera etapa se corroboró el estado de las páginas almacenadas en cada *dataset* para determinar si el URL estaba activo o inactivo. Esta tarea se llevó a cabo mediante un *script* de Python y la librería urllib2, que permite la conexión a la página web. Con este proceso fue posible verificar el contenido y "limpiar" cada *dataset*. Durante la verificación que se realizó, fue muy evidente el cambio de actividad de estas páginas, lo que indica que las paginas malignas tienen una vida útil más baja que las páginas de comportamiento normal o benignas. Esto hace que los sitios de los *dataset* 0 a 4 sean muy variables y muestren cambios de estado activo a inactivo en tan solo días u horas.

En la segunda etapa se obtuvo un *dataset* con URL activos. Para realizar la separación y clasificación de cada una de ellos se les asignó un identificador único por página, cuyo principal uso es la ubicación de cada URL. Por ejemplo, si el URL pertenece al *dataset* 0 y es la primera en la lista, su identificador es D0_1. Este proceso se llevó a cabo mediante un *script* hecho en Python.

En la tercera etapa se obtuvo un *dataset* que incluye a cada uno de los URL clasificados con un identificador único. Para llevar a cabo este proceso se usaron de dos herramientas que permiten tener los datos necesarios de cada página web: un *honeypot* y un *sniffer*.

Un *honeypot* es un sistema que emula ciertos servicios de red, una de sus funciones es estudiar los posibles ataques que se hacen al sistema o a los servicio de red que éste emule. Los *honeypots* están clasificados de acuerdo con su permisividad de interacción con el atacante en baja, media y alta. En este proyecto se utilizó un *honeypo*t de baja interacción llamado Thug, el cual permite simular un entorno y distintos navegadores de red con vulnerabilidades para estudio de múltiples ataques provenientes de los sitios web que él visita. Un *sniffer* es un sistema

que permite la captura de tráfico de red de una determinada interfaz de red. En el mercado existen varios sistemas de este tipo muy populares, entre ellos: Wireshark y TCPDUMP. En este proyecto se trabajó con TShark, una versión de Wireshark que cuenta con una librería en Python, lo que lo hace adecuado.

Una vez las herramientas han sido configuradas y el *dataset* con los URL creados, se procede a examinar cada URL con el *honeypot* y el *sniffer* para capturar sus datos. Cada URL maneja un tráfico de red y su archivo resultante se guarda en una carpeta independiente para su posterior análisis. Para los *dataset* malignos, el preprocesamiento arrojó 599 URL activos para el *dataset* 0; 3 activos, para el 1; 17 activos para el 2; 75 activos, pare el 3; y 35.279 activos, para el 4, en total 35.973 URL activos. Dado que existía un mayor número de URL benignos, se procesaron primero los URL malignos con el fin de saber cuántos estaban activos, y una vez determinado ese número, se seleccionó aleatoriamente un número similar para verificar los URL benignos y obtener su tráfico. Finalmente, se adquirieron 27.912 URL activos.

FASE 3. PROCESAMIENTO DE LOS DATOS

La selección de las características con la que se trabajó durante el proyecto se realizó con base en el artículo de Cross-LayerDetection [69], las Tablas 20 y 21 presentan el detalle para las capas de aplicación y red, respectivamente.

Tabla 20. Características - Capa de aplicación

Localización	Característica	Observaciones
Generales	(A1) URL_LENGTH	Tamaño total del URL.
	(A2) NUMBER_SPECIAL_ CHARACTERES	Cantidad de caracteres extraños que contiene el URL (e.g., /, %, #, &, =).
Información de HTTP	(A3) HTTPHEADER_ CHARSET	Conjunto de caracteres con los que se identifica el sitio web (e.g., ANSI, ISO-8859-1, UTF-8).
	(A4) HTTPHEADER_SERVER	Servidor en el que fue montado el URL.
	A6) HTTPHEADER_ CONTENT_LENGTH	Tamaño del contenido de la cabecera HTTP.
Propiedades Whois	(A7) WHOIS_COUNTRY	País donde estás el servidor del sitio web.
	(A8) WHOIS_STATEPRO	Indica el estado del sitio web.
	(A9) WHOIS_REGDATE	Fecha de registro del servidor del sitio web.
	(A10) WHOIS_UPDATED_DATE	Fecha de actualización del servidor.
	(A11) WHOIS_DOMAIN	Dominio del sitio web.

Para obtener la información HTTP y las propiedades Whois, se examinó directamente el URL con librerías que Python proporciona (requests y whois). El inconveniente de trabajar con enlaces tan cambiantes es que ellos pueden pasar de estado "activo" a "no activo" en cuestión de segundos, lo que implica un cambio del retorno de la consulta hacia ellas. En la matriz de aplicación existe un valor "None", el cual indica que el URL no presentaba información al momento de ser consultada esa característica.

Para la capa de red se tuvo en cuanta a los archivos de tráfico de datos que se generaron con la herramienta TCPDUMP y por medio de la herramienta llamada PyShark, un *wrapper* de TShark que permite la manipular los datos de un tráfico de red en formato pcap [71].

Tabla 21. Características - Capa de red

Característica	Observaciones
(R1) TCP_CONVERSATION_ EXCHANGE	Cantidad de paquetes que hay entre el *honeypot* y el sitio web para el protocolo TCP.
(R2) DIST_REMOTE_TCP_PORT	Total (#) de puertos distintos a los puertos TCP.
(R3) REMOTE_IPS	Cantidad de distintas direcciones IP conectadas al *honeypot*.
(R4) PACKETS _WITHOUT_DNS	Almacena en un arreglo todos los paquetes que no son DNS.
(R5) APP_BYTES	Bytes (#) de la capa de aplicación que envía el *honeypot* hacia el sitio web, sin incluir los datos de los servidores DNS.
(R6) UDP_ PACKETS	Paquetes UDP (#), sin incluir los datos de los servidores DNS.
(R7) TCP_URG_ PACKETS	Paquetes TCP (#) con la bandera URG (urgente).
(R8) SOURCE_APP_ PACKETS	Paquetes enviados (#) por el *honeypot* hacia el servidor remoto.
(R9) REMOTE_APP_PACKETS	Volumen (*bytes*) de la comunicación del servidor web hacia el *honeypot*.
(R10) DURATION	Tiempo de duración de la página web.
(R11) AVG_LOCAL_PACKETS_ RATE	Promedio de paquetes IP por segundo (paquetes enviados / duración).
(R12) AVG_REMOTE_PACKETS_ RATE	Promedio de paquetes IP por segundo (paquetes enviados / duración).
(R13) APP_PACKETS	Paquetes IP (#) , incluidos los del servidor DNS.
(R14) DNS_QUERY_TIMES	Lista de capas de DNS queries.

Después de obtener todas las características de las dos capas se inició la creación de una matriz con valores entre 0 y 1 (incluyendo estos dos números) para posteriormente realizar un análisis con algunos algoritmos de *machine Learning*. En la definición de las características y las métricas utilizadas para que los datos de los atributos mencionados se representen con los dígitos 0 y

1 se tuvo en cuenta la investigación de Mohaisen [70], quien indica que los atributos que tienen valores numéricos se deben normalizar para que sean comparables. Por otra parte, para los valores que son características o alfabetos, se usaron variables indicadoras –más conocidas como variables *dummy*.

Esto último se explica con el siguiente ejemplo: una matriz de cuatro columnas, incluyendo el identificador, tiene una columna con valores alfabéticos y las otras dos numéricos, como se presenta en la Tabla 22. Como se puede observar, los valores de la matriz que se presenta en la Tabla 22 corresponde a datos normales. Dado que esta información debe ser binaria, se normalizan los datos numéricos (columnas A y B) y se convierten en columnas los valores de C, de tal manera que se puedan expresar como: 1, si el identificador estaba asociado al valor determinado, y 0, si no lo estaba, como se ilustra en la Tabla 23. De esta forma, se garantiza la binarización de la matriz de características que va a pasar por los algoritmos clasificadores de *Machine Learning*.

Además, se tuvo en cuenta la correlación, el promedio y la frecuencia de los datos así: primero, por cada uno de los *datasets* se halló la correlación propia para analizar cómo es el comportamiento individual de los datos; después, al unir los *datasets* por clasificación (benigno y maligno) se revisó la correlación, pues se deseaba conocer la dependencia entre las características de las capas de aplicación y de red, sin embargo, al tener una matriz con tantas columnas, por convertir las variables categóricas en variables indicadoras, no se pudo representar gráficamente. Finalmente, se halló la frecuencia de los datos para

Tabla 22. Ejemplo de matriz de datos

Id	A	B	C
D0	1	45	Europa
D1	23	67	América
D2	45	89	Asia
D3	5	9	Europa

Tabla 23. Ejemplo matriz con variables dummy

id	A	B	C. Europa	C. América	C. Asia
D0	0	0.450	1	0	0
D1	0.4	0.725	0	1	0
D2	1	1	0	0	1
D3	0.09	0	1	0	0

las variables categóricas y el promedio para las numéricas de cada *dataset*. Lo anterior se refiere a un estudio experimental realizado con datos propios extraídos de los *datasets* generados –el cual permite obtener características propias– y las variables propuestas por Xu et al., [69], quienes eligieron los datos de Subset e InfoGain, por considerar que arrojarían los mejores resultados: longitud del URL, duración y nombre del servidor.

FASE 4. SELECCIÓN DE ALGORITMOS Y DE MEDIDAS DE EVALUACIÓN

Se seleccionaron como algoritmos de clasificación: J48, Naïve Bayes, SVM y Regresión logística. Como se mencionó, los clasificadores pueden ser evaluados a través distintas medidas que dependen del contexto o de las necesidades del modelo a ser aplicado, por ello, de acuerdo con el propósito de esta investigación, se continuó la evaluación aplicando la matriz de confusión, el tiempo de respuesta de los algoritmos y el coeficiente Cohen's kappa.

Experimento

Se utilizaron tres computadores: un Toshiba con procesador Intel Core i5 de 1.8 GHz y 4G de RAM, y dos Asus, uno con procesador Intel Core i7 de 2.4GHz y 8GB de RAM, otro con procesador Intel Celeron de 1.6GHz y 2 GB de RAM. Se seleccionaron los lenguajes Python y R, el editor de códigos Visual Studio Code y el señuelo Thug.

Python fue seleccionado porque presenta características acordes con una de las expectativas de este proyecto: ser la base de futuros proyectos. De las características apreciadas en él se destacan dos: es *open source*, lo que hace que funcione en diversas plataformas (e.g., Linux, Windows, Macintosh, Solaris, OS/2); y cuenta con múltiples librerías útiles, tipos de datos y funciones en lenguaje propio, lo que facilita la programación al no tener que implementar todo desde cero; R permite la optimización para uso estadístico en los datos obtenidos, facilita la limpieza y el manejo de un conjunto de ellos y proporciona herramientas y librerías que permiten el uso de *machine learning*; R Studio, por su parte, permite un mejor manejo de los recursos de R para la visualización y usabilidad del programador que desee desarrollar y solucionar problemas a nivel estadístico; Visual Studio Code es un editor de texto que permite el manejo de muchos archivos y la realización y ejecución de scripts; y Thug es un *honeypot* de baja interacción que permite ejecutar los URL en un ambiente controlado.

Resultados

Las Tablas 24, 25 y 26 presentan los resultados de los análisis exploratorios de los *datasets* de las capas de aplicación y de red. En cada una de ellas tablas se presenta la comparación entre los enlaces maliciosos y los enlaces benignos. La mayoría de los resultados son similares a los de la investigación realizada por Xu et al., [69]: se mantiene el patrón encontrado en 2013, la cantidad de caracteres extraños que hay en los URL malignos es mayor que la cantidad presente en los benignos; y esto, probablemente no va a cambiar en un periodo corto de tiempo.

Sin embargo, en la capa de red si existen algunas diferencias, estos son:

Tabla 24.Frecuencia de los datos no numéricos de la capa de aplicación

Característica evaluada	Datos benignos		Datos malignos	
	Característica frecuente	Frecuencia	Característica frecuente	Frecuencia
A3	UTF – 8	384	UTF – 8	1.547
	ISO - 8859 -1	193	ISO - 8859 -1	417
	US – ASCII	54	US – ASCII	33
A4	APACHE	204	APACHE	1.192
	NGINX	102	NGINX	425
	SIN REGISTRO	68	MICROSOFT – IIS	220
	MICROSOFT-HTTPAPI	30	SIN REGISTRO	182
	MICROSOFT – IIS	40	MICROSOFT–HTTPAPI	32
	CLOUDFLARE–NGINX	6	CLOUDFLARE–NGINX	30
A8	US	383	SIN REGISTRO	1.150
	SIN REGISTRO	120	US	272
	CN	4	CN	181
	CA	2	CA	27
	FR	2	CZ	8
	CZ	1	FR	4
A9	CA	158	AZ	94
	FL	27	WA	89
	WA	24	PA	66
	PA	15	FL	32
	AZ	13	AL	31
	AL	1	CA	16

- en los paquetes enviados al servidor remoto (R7) se observa que el promedio de los URL benignos es mayor que el de los maliciosos;
- en los paquetes enviados al *honeypot* (R8) se observa que el promedio de los enlaces benignos es mayor que las maliciosas;
- el promedio de *bytes* de aplicación enviados desde el *honeypot* es mayor en las benignas que en las malignas (R9);
- la tasa promedio de paquetes enviados al servidor remoto es mayor para las páginas web benignas que para las maliciosas (R12);
- la tasa promedio de paquetes enviados al *honeypot* es mayor para las páginas web benignas que para las maliciosas (R13); y
- el promedio de la cantidad de paquetes de aplicación enviados es mayor para las páginas web benignas que para las malignas (R14).

Tabla 25. Promedio de los datos numéricos de la capa de aplicación

Características evaluadas	Datos benignos	Datos malignos
A1	56,3155	85,455
A2	10,8163	17,203
A5	12.550,4000	9.683,262

Tabla 26. Promedio de los datos numéricos de la capa de red

Característica evaluada	Datos benignos	Datos malignos
R1	32,79	22,47
R2	10,63	2,47
R3	5,56	4,09
R4	3.562,86	2.691,37
R5	0,00	0,00
R6	0,00	0,00
R7	37,99	27,66
R8	37,89	25,42
R9	35.038,54	11.117,65
R10	3.961,74	3.088,23
R11	0,87	1,90
R12	44,62	14,58
R13	42,35	14,17
R14	37,99	27,66
R15	5,17	5,19

En cuanto a la correlación de los datos de la capa de red y la de aplicaciones, la Figura 7 muestra que para la capa de red con URL benignos las características con menos interdependencias en relaciones asociadas son REMOTE_IPS, UDP_PACKETS y TCP_URG_PACKETS. Ésta última, no tiene datos. La ilustración indica entonces, que no se deben tener en cuenta estas características cuando se vayan a entrenar los algoritmos.

La Figura 8 muestra que para la capa de red con enlaces malignos las características con menos interdependencia en relaciones asociadas son UDP_PACKETS y TCP_URG_PACKETS. En comparación con las benignas, los URL malignos tienen una mayor dependencia del valor REMOTE_IPS, lo que se debe a que hay mucha más cantidad de redirecciones realizadas al *honeypot*. A partir del nivel de correlación, en las Figuras 9 y 10 se pueden identificar las características que no se deben tener en cuenta en el entrenamiento de los algoritmos.

En las Figuras 9 y 10 se muestra la correlación entre los datos numéricos, debido a que esta capa tiene muchos valores no numéricos, al aplicarle el proceso de binarización, el resultado es una matriz bastante grande que genera problemas para observar la dependencia entre los datos. Sin embargo, cabe recalcar que las fechas (de registro y modificación) y el dominio no mostraron

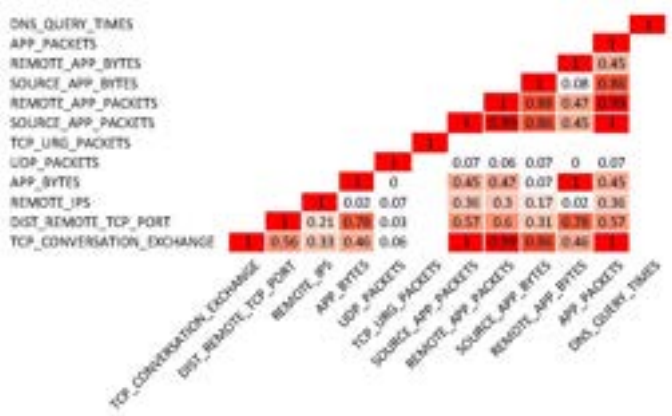


Figura 7. Correlación de los datos benignos de la capa de red

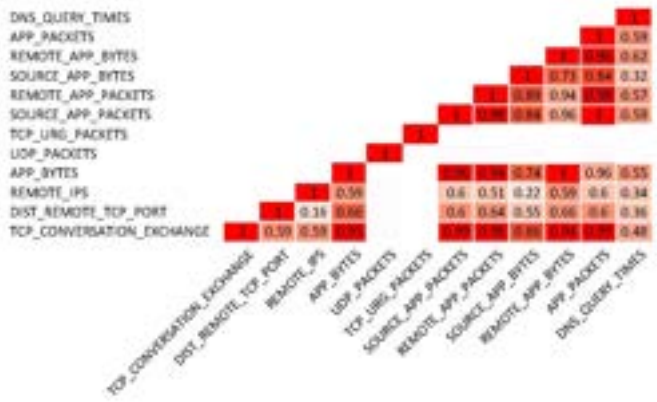


Figura 8. Correlación de los datos maliciosos de la capa de red

mayor relación con las otras variables. Las fechas, porque no se pueden repetir, es decir, no se puede devolver el tiempo para llegar a la fecha de creación o modificación y montar o actualizar el servidor; y el dominio, probablemente porque se ha dejado de utilizar de manera temporal o permanente, en el caso de un URL benigno.

En la Figura 9 se muestra que la correlación entre el número de caracteres especiales y la longitud del URL es muy alto, y que el CONTENT_LENGTH, no tiene datos para capturar la correlación con las otras dos variables.

En la Figura 10 se presenta una correlación alta entre el número de caracteres especiales y la longitud del URL, aunque no tan alta como la dependencia de la capa benigna, pero es un buen valor de dependencia. Tal como ocurre en la Figura 9, se observa que A5 no tiene datos que permitan capturar la correlación con las otras dos variables.

Teniendo en cuenta los análisis de las variables realizados y mostrados, se exponen a continuación los resultados de los algoritmos de entrenamiento. Se debe tener en cuenta que el *dataset* ésta compuesto por 967 observaciones y 400

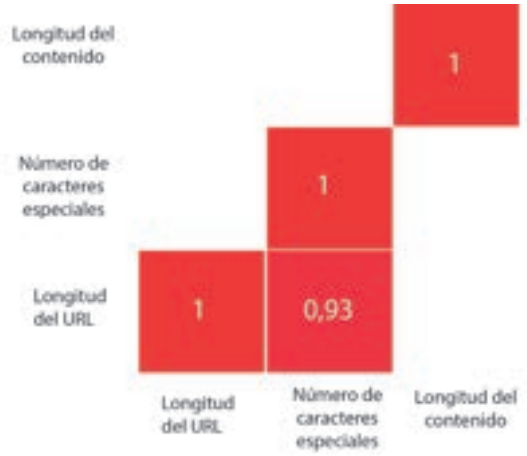


Figura 9. Correlación de los datos benignos de la capa de aplicación

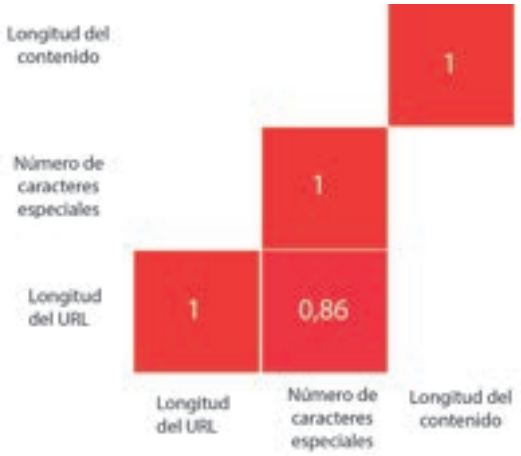


Figura 10. Correlación de los datos maliciosos de la capa de aplicación

variables (por la binarización de las características, la separación de los valores del cache control y del servidor). Para esta parte, primero se evaluó, por separado, la matriz de la capa de aplicación (Tabla 27) y luego la matriz de la capa de red y la matriz completa. Además, se aplicó la técnica de validación cruzada, con un k = 10 y un k = 15, para la evaluación de resultados, garantizando así la independencia de la partición de los datos correspondiente al entrenamiento y las pruebas. Las características evaluadas de la capa de aplicación son A1, A2, A3, A4, A5, A7 y A8; las evaluadas de la capa de red son R1, R2, R3, R4, R5, R6, R7, R8, R9, R13 y R14.

En la Tabla 28 se muestra el resultado del experimento para las características más importantes mencionadas en la investigación de Xu et al. [69]. Se encuentra una precisión adecuada para el algoritmo J48, con un porcentaje de 96.05%; se puede distinguir entonces que estas características resultan ser muy importantes para la detección de páginas maliciosas tal como lo explican Xu et al.

En cuanto a los resultados de los algoritmos ejecutados con la matriz completa (Tabla 29), es importante resaltar que aunque la matrices de correlación de las capas de red (Figuras 9 y 10) mostraron poca dependencia de las variables UDP_PACKETS y TCP_URG_PACKETS, se decidió tenerlas en cuenta para

Tabla 27. Resultados de los algoritmos por cada capa (k=10)

		Estimación de la exactitud de la clasificación (%)		Tiempo de respuesta (s)	
Algoritmo	Dataset capa de:	Aplicación	Red	Aplicación	Red
SVM		89,09	55,16	2, 01	1,90
Regresión Logística		88,43	54,11	3,56	0,87
Naïve Bayes		84,70	55,16	3,38	0,89
J48		90, 10	57, 01	4,00	4, 05

Tabla 28. Resultados de los algoritmos para las tres características obtenidas por los métodos subset e infogain (k=10)

Algoritmo	Estimación de la exactitud de la clasificación (%)	Tiempo de respuesta (s)
SVM	85.46	0.19
Regresión Logística	84.51	0.06
Naïve Bayes	85.46	0.02
J48	96.05	0.01

Tabla 29. Resultados de los algoritmos para la matriz de datos completa

		Estimación de la exactitud de la clasificación (%)		Tiempo de respuesta (s)	
Algoritmo	Dataset capa de:	k = 10	k = 15	k = 10	k = 15
SVM		97,41	96,74	3,38	3, 45
Regresión Logística		90,58	90,18	5, 31	5, 40
Naïve Bayes		98,76	99,07	53, 37	53, 43
J48		10,96	10,96	2, 28	2, 32

el entrenamiento porque, aunque no arrojaron buenos resultados, generan datos importantes y a futuro, si no se tienen en cuenta, se pueden realizar ataques por estos medios. UDP_Packets es el encargado de revisar streaming, videos, mp3, y TCP_URG_PACKETS, de los flasks que guardan información cuando ocurre algún problema en los paquetes. En la validación cruzada se aplicaron valores de k = 10 y k = 15.

Análisis

Según los resultados de los modelos, cuando se prueba con la matriz de aplicación, la probabilidad de que la clasificación sea correcta es superior a 80%, mientras que si se prueba con la matriz de red, las clasificaciones correctas no superan el 60%. Analizar cada uno de los modelos, con cada una de las capas, es importante ya que permite conocer el aporte de cada capa en la clasificación de un URL. En consecuencia, en este caso, las características de la capa de aplicación serán de mayor peso en la toma de la decisión de clasificación cuando se haga el entrenamiento con la matriz completa (integrada por ambas capas).

Los mejores algoritmos para entrenar la matriz completa fueron SVM y J48, ambos superaron el 95% de aciertos en la clasificación. De ellos, el mejor fue J48 y mejor aún con un valor de k=15. En contraposición, Naïve Bayes, no es una buena opción para el entrenamiento, ya que en todas los casos obtuvo los más bajos resultados.

Los hallazgos de esta investigación son muy similares a los obtenidos por Xu et al., [69] aunque en lugar de optar por el manejo de las tres variables usadas en dicho estudio, se incluyó el mayor número posible de ellas. Cabe destacar que al estudiarlas indicaron una buena correlación.

Trabajo futuro

Este proyecto deja: cinco *datasets*, dos para cada capa y uno que contiene ambas capas; los scripts de los algoritmos que se implementaron; las gráficas obtenidas; y la base para la ejecución del software de detección. De las actividades realizadas y los resultados obtenidos surgen algunas recomendaciones para darle continuidad a lo hecho durante esta investigación: generar un sistema de detección de páginas maliciosas por medio de aprendizaje no supervisado y un sistema para la detección de tipos de ataques web por medio de aprendizaje supervisado y no supervisado; investigar *honeypots* y herramientas que permitan un mejor avance; obtener *datasets* –que permitan el desarrollo experimental– y de algoritmos –que permitan identificar ataques y problemas en una página web–; y desarrollar dos herramientas, una para la detección de ataques en aplicaciones móviles, otra que permita la detección e identificación de ataques en tiempo real.

A partir de las lecciones aprendidas

La ciencia de datos puede llegar a ser (si es que aún no lo es) una herramienta importante para el desarrollo de la ciberseguridad. Puede ser útil, tanto para prevenir ataques, como para detectar aplicaciones maliciosas o analizar *a posteriori* (análisis forense) información relacionada con un evento. Sin embargo, para usar técnicas de aprendizaje de máquina, tales como el *deep learning*, es indispensable disponer de cierta información que no siempre está disponible, información que incluso, algunas veces, no se puede conseguir.

Por ejemplo, en muchos casos, para hacer análisis de *malware* se requiere, entrenar a los algoritmos, y para ello es necesario contar con *datasets* de *malware* que provengan de fuentes confiables, pues solo así es posible adelantar un proceso de entrenamiento adecuado.

Una de las tareas más complejas es encontrar estos *datasets*, pues no solo es dispendioso recolectar la información, sino que en muchos casos, la que se encuentra está desactualizada. Por lo tanto, es necesario incluir este factor en el entrenamiento de los algoritmos y considerar los posibles errores de detección asociados con el uso de datos no actualizados.

Si se utilizan algoritmos no supervisados, por ejemplo para analizar páginas web, se requiere contar con un período estable de operación "normal" del sistema para entrenar el algoritmo, de tal forma el sea capaz, más adelante, de identificar situaciones "anormales". Bajo este supuesto, lo recomendable es mantener algún tipo de supervisión humana experta, que permita identificar las falsas alarmas (falsos positivos) e ir mejorando el entrenamiento del algoritmo.

Uno de los riesgos que enfrentan los algoritmos de *machine learning* para este tipo de aplicaciones se deriva de los grandes cambios que puede sufrir un sitio

web corporativo cuando se hace una actualización masiva, pues la gran cantidad de cambios puede llevar al algoritmo a confundir esa situación normal, sin riesgos, de mantenimiento del sitio, con una situación anómala equivalente a un ataque. Este tipo de consideraciones se deben incorporar en el proceso.

APLICACIÓN DE LA CIENCIA DE DATOS AL ANÁLISIS DE CIBERAMENAZAS

Con base en lo dicho en este documento y teniendo en cuenta las metodologías de aplicación de la ciencia de datos presentadas, se propone un procedimiento estrechamente vinculado con el método científico de investigación, el cual puede constituirse en una base para la formulación futura de un framework para el entrenamiento de modelos de *machine learning* aplicados a la detección de amenazas cibernéticas en un contexto de investigación. Este procedimiento, como se ilustra en la Figura 11, desarrolla el método científico, es decir: identifica un problema, formula un objetivo o pregunta de investigación, establece una hipótesis y busca su aceptación o refutación.

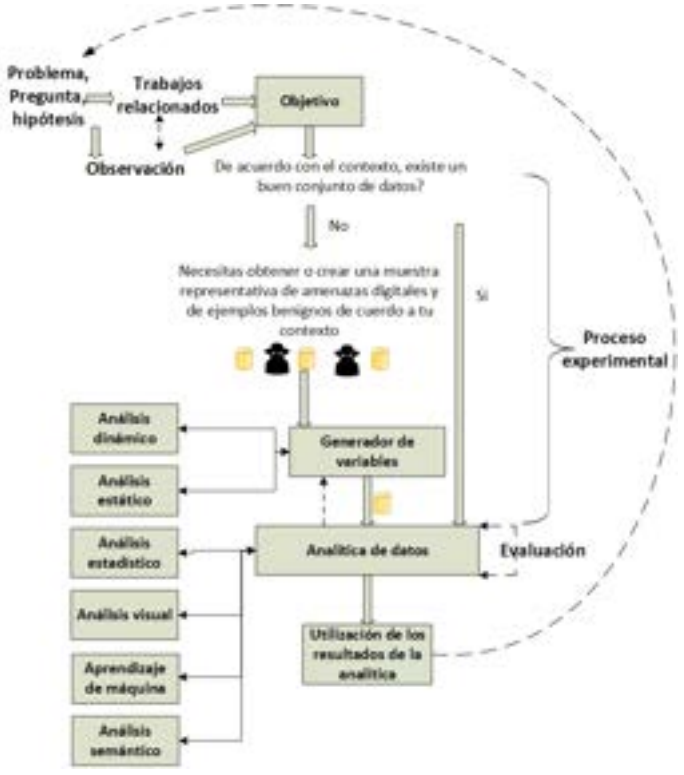


Figura 11. Proceso de aplicación de la ciencia de datos en ciberseguridad

Como se observa en la figura, identificar y analizar trabajos relacionados con el tipo de problema que se ha identificado –sean o no soluciones que aplican ciencia de datos–, es una actividad inicial central. De su implementación se espera conocer: las metodologías con las que abordaron los problemas que identificaron, las muestras de amenazas cibernéticas que utilizaron, los métodos de análisis que aplicaron, las características que permitieron la identificación de las amenazas, los resultados que obtuvieron y las conclusiones y recomendaciones de trabajos a futuro que de ellos se derivaron.

Una vez identificados los elementos diferenciadores de cada trabajo, se propone una hipótesis. La complejidad que se presenta en la ciberseguridad permite una variedad que va desde la adquisición de los datos, hasta la evaluación de los modelos en el mundo real.

Cuando se ha formulado la hipótesis, es necesario conocer si existen *datasets* para uso investigativo (preferiblemente con soporte académico o industrial) que relacionen a las amenazas digitales y a los entornos benignos. Si no existen o no son accesibles, se deben obtener muestras representativas de ciberamenazas, esto es conjuntos conformados por distintas familias de software malicioso, tipos de ciberataque y capturas de tráfico web, y entornos sanos, para que puedan ser procesadas a través de los análisis estático, dinámico e híbrido. Es posible también desarrollar *malware* propio o simular ciberataques, opciones válidas siempre y cuando se pueda obtener información que le permita al modelo la identificación de casos reales.

Para aplicar los análisis estático, dinámico e híbrido y generar así información acerca de las amenazas digitales y las acciones benignas, existen herramientas –de libre acceso y comerciales–, Es posible también desarrollar mecanismos propios de análisis, sin embargo, hacerlo dificulta la replicación y evaluación de los resultados, a menos que se ofrezca al público las fuentes de información utilizadas en la fase de entrenamiento. Resumiendo, se busca la abstracción de las características con la finalidad de generar uno o más *datasets* para la etapa de análisis.

En la etapa de análisis de datos, se define un enfoque y derivado de él un objetivo: exploratorio (EDA), si no se conocen los datos previamente y se quieren encontrar patrones o relaciones; confirmatorio, si se conocen los datos previamente, y se quiere realizar el entrenamiento y prueba de modelos predictivos.

Una vez entrenados y testados los modelos predictivos de ciberamenazas, el siguiente objetivo es medir su capacidad con una nueva muestra, totalmente distinta. La identificación de falsos positivos o positivos negativos es útil para futuros trabajos, por lo que se recomienda tenerlos en cuenta en la conformación de una muestra representativa que pueda usarse para el ajuste del proyecto o para el entrenamiento de otros modelos predictivos.

CONJUNTOS DE DATOS

Durante la ejecución del proyecto de ciberseguridad para dispositivos móviles con sistema operativo Android fue evidente la escasez de repositorios de *malware* de libre acceso para investigación; y en el proyecto de análisis de páginas maliciosas, fue evidente la rápida pérdida de vigencia de que gran parte de los enlaces a sitios web malignos.

Compartir información es fundamental para el desarrollo de mejores soluciones porque le permite a otros continuar o replicar los resultados de los trabajos de investigación, para hacerlo se pueden utilizar las plataformas IEEE DataPort [72] y Kaggle [73]. Los *datasets* utilizados en los proyectos citados en este libro están disponibles en ellas [74], [75].

UN CAMINO PROMETEDOR

Los métodos basados en *machine learning* para identificación de amenazas cibernéticas también pueden ser vulnerados, Adversarial Machine Learning es el campo que hace la intersección entre la ciberseguridad y la ciencia de datos, e investiga acerca de las falencias y los mecanismos que ofrecen un mayor soporte a los sistemas basados en este tipo de aprendizaje. Existen: trabajos que demuestran cómo un clasificador puede ser vulnerado a partir de sus resultados y un proceso de adaptación de un ataque de evasión [76]; investigaciones en *malware* para Android [77]; y soluciones con redes neuronales simples y profundas [78].

Debido a la relevancia que ha tomado la ciencia de datos y a las vulnerabilidades que se presentan, la inteligencia artificial es también un camino prometedor de trabajo para la ciberseguridad. Sin embargo, las soluciones basadas en ella tienen sus propios retos, y por tanto es necesario trabajar en mecanismos que las vuelvan más eficientes y seguras.

Referencias

[1] Symantec Corp. *Internet Security Threat Report* [vol. 22]. Mountain View, CA: Symantec Corp., 2017.

[2] L. Batyuk, M. Herpich, S. A. Camtepe, K. Raddatz, A. Schmidt, y S. Albayrak. "Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within Android applications", en *Malicious and Unwanted Software (MALWARE), 2011 6th International Conference on*. IEEE.

[3] M. I. Sharif, A. Lanzi, J. T. Giffin, y W. Lee, W. Impeding Malware Analysis Using Conditional Code Obfuscation. In *NDSS Symposium 2008*. Internet Society.

[4] L. F. Fuentes. (2008). "Malware, una amenaza de Internet". Revista Digital Universitaria, vol. 9, no. 4, art. 22. Disponible en: http://www.revista.unam.mx/vol.9/num4/art22/art22.pdf

[5] J. J. Drake, Z. Lanier, C. Mulliner, P. O. Fora, S. A. Ridley, y G. Wicherski. Android hacker's handbook. Indianapolis, IN: John Wiley & Sons, 2014.

[6] T. Petsas, G. Voyatzis, E. Athanasopoulos, M. Polychronakis, y S. Ioannidis, S. "Rage against the virtual machine: Hindering dynamic analysis of android malware, en *Eurosec'14 Proceedings of the Seventh European Workshop on System Security*, Amsterdam: The Netherlands (art. 5), ACM, abr. 2014. https://doi.org/10.1145/2592791.2592796

[7] C. Urcuqui, y A. Navarro, "Framework for malware analysis in Android", *Sistemas & Telemática*, vol. 14, no. 37, pp. 45-56, 2016. https://doi.org/10.18046/syt.v14i37.2241

[8] J. Vergara, M. Martinez, y O. Caicedo, "A benchmarking of the efficiency of supervised ML algorithms in the NFV traffic classification", *Sistemas & Telemática*, vol. 15, no. 42, pp. 47-67, 2017. https://doi.org/10.18046/syt.v15i42.2539

[9] R. Lara, y A. Estévez. "Towards an automatic detection system of sports talents: An approach to Tae Kwon Do", *Sistemas & Telemática*, vol. 16, no. 47, pp. 31-44, 2018. https://doi.org/10.18046/syt.v16i47.3213

[10] O. Trejos, y V. Miramá, "Machine learning algorithms for inter-cell interference coordination", *Sistemas & Telemática*, vol. 16, no. 46, pp. 37-57, 2018. https://doi.org/10.18046/syt.v16i46.3034

[11] D. Corrales, A. Ledezma, A. Peña, J. Hoyos, A. Figueroa, y J. C. Corrales. "A new dataset for coffee rust detection in Colombian crops base on classifiers", *Sistemas & Telemática*, vol. 12, no. 29, pp. 9-23, 2014. https://doi.org/10.18046/syt.v12i29.1802

[12] R. Rosales, y M. Correa. "Model of routing for raw milk collection using genetic algorithms", Sistemas & Telemática, vol. 12, no. 31, pp. 77-88, 2014. https://doi.org/10.18046/syt.v12i31.1916

[13] Y. Zhong, H. Asakura, H. Takakura, y Y. Oshima, "Detecting malicious inputs of web application parametersusing character class sequences, en *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual, jul.* 2015, vol. 2, pp. 525-532).

[14] T. Erl, W. Khattak, y P. Buhler. *Big data fundamentals: concepts, drivers & techniques*, Boston MA: Prentice Hall Press, 2016.

[15] W. C. Wu, y S. H. Hung, "DroidDolphin: A dynamic Android malware detection framework using big data and machine learning". en *Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems*, New York, NY: ACM, Oct. 2014, pp. 247-252.

[16] E. Alpaydin, *Introduction to machine learning*. Cambridge, MA: MIT Press, 2014.

[17] A. McCallum y K. Nigam, "A Comparison of Event Models for Naïve Bayes Text Classification," *AAAI/ICML- 98 Work. Learn. Text Categ.*, 1998, pp. 41- 48.

[18] Y. Gala-García, "Algoritmos SVM para problemas sobre big data", tesis de maestría, Madrid, España: Universidad Autónoma de Madrid, 2013.

[19] G. Kaur y A. Chhabra, "Improved J48 classification algorithm for the prediction of diabetes", *Int. J. Comput. Appl.*, vol. 98, no. 22, pp. 13-17, 2014.

[20] P. K. Chan, y R. Lippmann, "Machine learning for computer security", *The Journal of Machine Learning Research*, vol. 7, 2669-2672, 2006.

[21] E. Gandotra, D. Bansal, y S. Sofat, "Integrated framework for classification of malwares, en *Proceedings of the 7th International Conference on Security of Information and Networks*. ACM, Sep. 2014, p- 417.

[22] K. Rieck, Computer security and machine learning: Worst enemies or best friends? en *First SysSec Workshop (SysSec)*, IEEE, Jul. 2011, pp. 107-110.

[23] N. Elenkov, Android security internals: An in-depth guide to Android's security architecture. San Francisco, CA: No Starch Press, 2014.

[24] *Security* [en línea], disponible en: https://source.android.com/devices/tech/security/

[25] N. Peiravian, y X. Zhu, Machine learning for android malware detection using permission and api calls, en *Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on*, Nov. 2013, pp. 300-305.

[26] *Manifest.permission* [en línea], disponible en: http://developer.android.com/reference/android/Manifest.permission.html

[27] L. Black (2018, oct.), *Tumbleson, C. Dex2jar: Tools to work with android. dex and java. class files* [en línea]", disponible en: https://www.kitploit.com/2018/10/dex2jar-tools-to-work-with-android-dex.html

[28] C. Tumbleson y R. Winsniewski (2012), *Android–Apktool: A tool for reverse engineering android apk files* [en línea], disponible en: https://ibotpeaches.github.io/Apktool/

[29] S. Khandelwal (2014, mar.), *Android malware 'Dendroid' targeting Indian users* [en línea], disponible en: http://thehackernews.com/2014/03/android-malware-dendroid-targeting_26.html

[30] S. Khandelwal (2014, may.), *Police ransomware malware targeting Android smartphones* [en línea], disponible en: http://thehackernews.com/2014/05/police-ransomware-malware-targeting.html.

[31] H. Dharmdasani, y V. Pidathala (2014), *Android.MisoSMS : Its back! now with XTEA* [en línea], disponible en: http://www.fireeye.com/blog/technical/malware-research/2014/03/android-misosms-its-back-now-with-xtea.html

[32] J. Forristal. "Android Fake ID Vulnerability", en *Black Hat USA 2014*, Las Vegas: NV, recuperado de: https://www.blackhat.com/docs/us-14/materials/us-14-Forristal-Android-FakeID-Vulnerability-Walkthrough.pdf

[33] T. Wei, Y. Zhang, H. Xue, M. Zheng, C. Ren, y D. Song. "Sidewinder. Targeted attack against Android in the golden age of ad libraries", en *Black Hat USA 2014*, Las Vegas: NV, recuperado de:https://www.blackhat.com/docs/us-14/materials/us-14-Wei-Sidewinder-Targeted-Attack-Against-Android-In-The-Golden-Age-Of-Ad-Libs.pdf

[34] Y. Zhou, y X. Jiang. "Dissecting android malware: Characterization and evolution", en *2012 IEEE Symposium on Security and Privacy (SP)*, pp. 95-109), 2012, http://doi.org/10.1109/SP.2012.16

[35] A. Navarro, S. Londoño, C. Urcuqui, M. Fuentes, y J. Gomez, "Análisis y caracterización de frameworks para detección de aplicaciones maliciosas en Android", en *XIV Jornada Internacional de Seguridad Informática ACIS 2014,* jun. 2014, recuperado de: https://www.researchgate.net/publication/263236428_Analisis_y_caracterizacion_de_frameworks_para_deteccion_de_aplicaciones_maliciosas_en_android

[36] S. Londoño, C. Urcuqui, A. Navarro, M. Amaya, y J. Gómez, SafeCandy: System for security, analysis and validation in Android, *Sistemas & Telemática*, vol. 13, no. 35, pp. 89-102, 2015. https://doi.org/10.18046/syt.v13i35.2154

[37] K. Tam, A. Feizollah, N. Anuar, R. Salleh, y L. Cavallaro. "The evolution of android malware and android analysis techniques", *ACM Computing Surveys (CSUR)*, vol. 49 no.76, art. 76, 2017. http://doi.org/10.1145/3017427

[38] F. Narudin, A. Feizollah, N. Anuar, y A. Gani. "Evaluation of machine learning classifiers for mobile malware detection". *Soft Computing*, vol. 20, no. 20, pp. 343-357, 2016. https://doi.org/10.1007/s00500-014-1511-6

[39] Y. Zhou, y X. Jiang (2012). *Android malware genome project* [en línea], disponible en: http://www.malgenomeproject.org

[40] D. Krutz, M. Mirakhorli, S. Malachowsky, A.Ruiz, J.Peterson, A.Filipski, y J.Smith. "A dataset of open-source Android applications", en *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories (MSR)*, IEEE, may. 2015, pp. 522-525. https://doi.org/10.1109/MSR.2015.79

[41] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, y K. Rieck. "DREBIN: Effective and explainable detection of Android malware in your pocket", en *Proceedings of the 2014 Network and Distributed System Security Symposium (NDSS'14)*, San Diego, CA, feb. 2014, recuperado de: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/11_3_1.pdf

[42] K. Allix, T. Bissyandé, J. Klein, y Y, Le. "Androzoo: Collecting millions of Android apps for the research community", en *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, IEEE, mayo 2016, pp. 468-471. https://doi.org/ 10.1109/MSR.2016.056

[43] D. Cao, S. Wang, Q. Li, Z. Cheny, Q. Yan, L. Peng, y B. Yang. "DroidCollector: A high performance framework for high quality Android traffic collection", en *2016 IEEE Trustcom/BigDataSE/I SPA*, ago. 2016, pp. 1753-1758. http://doi.org/ 10.1109/TrustCom.2016.0269

[44] VirusShare (2017), *VirusShare.com - Because sharing is caring* [en línea], disponible en: https://virusshare.com/

[45] M. Parkour (2013), *Contagio Mobile - Mobile malware mini dump* [en línea], disponible en: http://contagiominidump.blogspot.com.co

[46] Malware-Traffic-Analysis.net (2013). *A source for pca. p files and malware samples* [en línea], disponible en: http://malware-traffic-analysis.net/

[47] C. Chang, y C. Lin. "LIBSVM: A library for support vector machines", *ACM Transactions on Intelligent Systems and Technology (TIST)*,vol. 2,no. 27, 2011. http://doi.org/ 10.1145/1961189.1961199

[48] A. Feizollah, N. Anuar, R. Salleh, F. Amalina, R. Ma'arof, y S. Shamshirband, "A study of machine learning classifiers for anomaly-based mobile botnet detection", *Malaysian Journal of Computer Science*, vol. 26, no. 4, pp. 251-265, 2013.

[49] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, e I. Witten. "The WEKA data mining software: an update". *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10-18, 2009. http://doi.org/10.1145/1656274.1656278

[50] J. Sahs, y L. Khan, "A machine learning approach to android malware detection", en *2012 European Intelligence and Security Informatics Conference (EISIC)*, ago. 2012, pp. 141-147. http://doi.org/10.1109/EISIC.2012.34

[51] A. Desnos, y G. Gueguen, "Android: From reversing to decompilation", en *Black Hat Abu Dhabi*, 2011, pp. 77-101. Disponible en: https://media.blackhat.com/bh-ad-11/Desnos/bh-ad-11-DesnosGueguen-Andriod-Reversing_to_Decompilation_WP.pdf

[52] scikit-learn: machine learning in Python (2018, nov.), *scikit-learndocumentation* [en línea], disponible en: http://scikit-learn.org/stable/

[53] M. Ghorbanzadeh, Y. Chen, Z. Ma, T. Clancy, y R. McGwier. "A neural network approach to category validation of android applications", en *2013 International Conference on Computing, Networking and Communications (ICNC)*, IEEE, ene., 2013, pp. 740-744. http://doi.org/ 10.1109/ICCNC.2013.6504180

[54] S. Yerima, S. Sezer, G. McWilliams, e I. Muttik, "A new android malware detection approach using bayesian classification", en *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, 2013, mar., pp. 121-128). http://doi.org/ 10.1109/AINA.2013.88

[55] A. K. Dalai, y S. K. Jena, "Evaluationof web application security risks and secure design patterns, en: *Proceedings of the 2011 International Conference on Communication, Computing & Security*, ACM, 2011, feb., pp. 565-568.

[56] WhiteHat Security, *Applications security statistics report: The evolution of the secure software life cycle*. San José, CA: WhiteHat Security, 2018.

[57] A. Halsall, "Introducción", en R. Moreno (Ed.), *Redes de computadoras e internet*, Madrid, España: Addison-Wesley, 2006, pp. 20-24.

[58] The Open Web Application Security Project (2017), *OWASP Top 10 - 2017 The Ten Most Critical Web Application Security Risks* [en línea], disponible en: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

[59] Hipertextual.com, (2016, Oct, 21), *Esta es la razón por la que muchas webs importantes están caídas* [en línea], disponible en: https://hipertextual.com/2016/10/ataque-dns-webs-caidas

[60] Página web de la Registraduría sufrió un ataque informático, (2016, sept. 28), disponible en: http://www.elpais.com.co/elpais/colombia/proceso-paz/noticias/pagina-web-registraduria-sufrio-ataque-informatico#.V-yIXlhusjg.email

[61] O. Mondragón, A. Mera, C. Urcuqui, y A. Navarro, "Security control for website defacement", *Sistemas & Telemática*, vol. 15 no. 41, pp. 45-55. https://doi.org/10.18046/syt.v15i41.2442

[62] Ministerio de las Tecnologías de la Información y las Comunicaciones [MINTIC], Conpes 3701 de 2011: Lineamientos de política para la Ciberseguridad y Ciberdefensa, Bogotá, Colombia, MINTIC, 2011.

[63] colCERT, Grupo de Respuesta a Emergencias Cibernéticas de Colombia [portal], disponible en: http://www.colcert.gov.co/

[64] C. Urcuqui y A. Navarro, "Working together to control website defacement: Lessons from Colombia, *ITUNews*, 2017, marzo 23, disponible en: https://news.itu.int/working-together-to-control-website-defacement-lessons-from-colombia/

[65] M. Roesch, Snort: Lightweight intrusion detection for networks, *LISA*, vol. 99, no. 1, pp. 229-238). Nov., 1999.

[66] B. Caswell, J. Beale, y A. Baker, *Snort intrusion detection and prevention toolkit*. Burlington, MA: Syngress, 2007.

[67] G. Howard, C. Gutierrez, F. Arshad, S. Bagchi, y Y. Qi. "pSigene: webcrawling to generalize SQL injection signatures",en *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, jun., 2014, junio, pp. 45-56. http://doi.org/10.1109/DSN.2014.21

[68] A. Bartoli, G. Davanzo, y E. Medvet, "A framework for large-scale detection of web site defacements", *ACM Transactions on Internet Technology (TOIT)*, vol. 10, no. 3, Oct., 2010.

[69] L. Xu, Z. Zhan, S. Xu, and K. Ye, "Cross-layer detection of malicious websites", en *Proc. ACM Conf. Data Appl. Secur. Priv.*, 2013, pp. 141–152.

[70] A. Mohaisen, "Towards automatic and lightweight detection and classification of malicious web contents", en *Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), Washington, DC,* 2015. https://doi.org/10.1109/HotWeb.2015.20

[71] *PyShark: Python packet parser using wireshark's tshark* [en ínea], disponible en: http://kiminewt.github.io/pyshark/

[72] *Kaggle / Datasets* [en línea], disponible en: https://www.kaggle.com/datasets

[73] *IEEE DataPort* [en línea], disponible en: https://ieee-dataport.org/documents/dataset-malwarebeningn-permissions-android

[74] C. Urcuqui, *Malicious and benign websites* [en línea], disponible en: https://www.kaggle.com/xwolf12/malicious-and-benign-websites

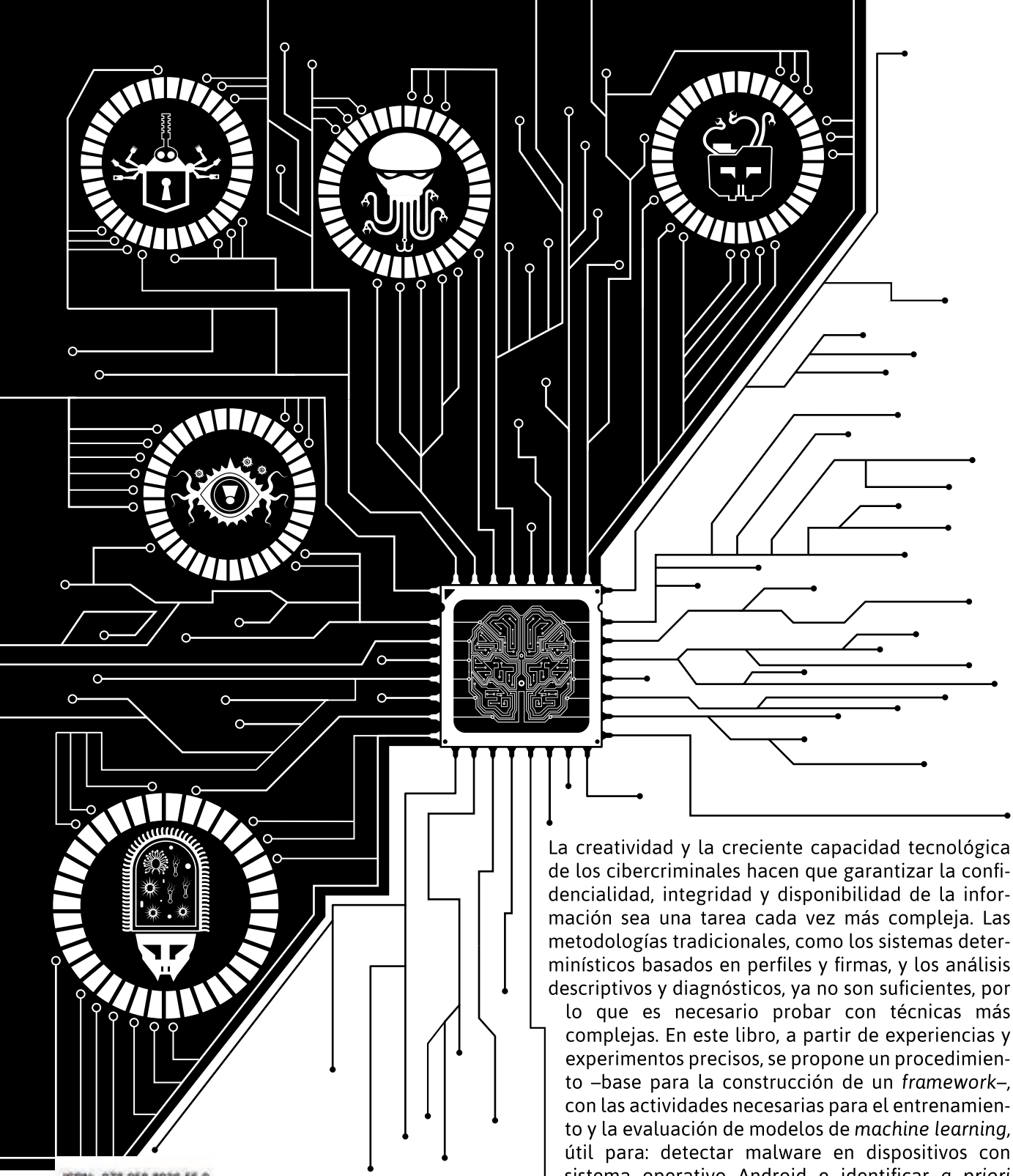
[75] C. Urcuqui., A. Navarro, J. Osorio, y M. García (2018), *Malicious and benign websites* [en línea], disponible en: https://ieee-dataport.org/documents/malicious-and-benign-websites

[76] H. Dang, Y. Huang, y E. C. Chang, "Evading classifiers by morphing in the dark, en *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security,* oct. 2017, pp. 119-133.

[77] W. Hu, y Y. Tan (2017, feb. 20, *Generating adversarial malware examples for black-box attacks based on GAN* [en línea], disponible en: https://arxiv.org/abs/1702.05983

[78] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, y P. Abbeel (2017, feb. 8), *Adversarial attacks on neural network policies* [en línea], disponible en: https://arxiv.org/abs/1702.02284

Este libro se terminó de imprimir y encuadernar en diciembre de 2018 en los talleres de Carvajal Soluciones de Comunicación, en la ciudad de Bogotá D.C. En su preparación, realizada desde la Editorial Universidad Icesi, se emplearon los tipos Gill Sans MT de 8, 10, 14 y 26 puntos, Baskerville MT Std de 9, 10, 12 y 30 puntos, Times New Roman de 10 puntos y Cambria Math de 12 puntos. La edición, que consta de 50 ejemplares, estuvo al cuidado de Claros Editores S.A.S.



La creatividad y la creciente capacidad tecnológica de los cibercriminales hacen que garantizar la confidencialidad, integridad y disponibilidad de la información sea una tarea cada vez más compleja. Las metodologías tradicionales, como los sistemas determinísticos basados en perfiles y firmas, y los análisis descriptivos y diagnósticos, ya no son suficientes, por lo que es necesario probar con técnicas más complejas. En este libro, a partir de experiencias y experimentos precisos, se propone un procedimiento –base para la construcción de un *framework*–, con las actividades necesarias para el entrenamiento y la evaluación de modelos de *machine learning*, útil para: detectar malware en dispositivos con sistema operativo Android e identificar *a priori* aplicaciones web maliciosas.

ISBN: 978-958-8936-55-0