COORDINADORES

CHRISTIAN CAMILO URCUQUI LÓPEZ

ANDRÉS NAVARRO CADAVID

CIBERSEGURIDAD:
LOS DATOS TIENEN LA RESPUESTA

Editorial Universidad Icesi

UNIVERSIDAD ICESI

Ciberseguridad
Los datos tienen la respuesta

Ciberseguridad
Los datos tienen la respuesta

Christian Camilo Urcuqui (Coord.)
Andrés Navarro Cadavid (Coord.)
Brayan Andrés Henao
Juan Sebastián Prada
Andrés Felipe Pérez Belalcázar
Steven Bernal Tovar
Julio César Gaviria Jaramillo
Ánderson Ramírez
Jhoan Steven Delgado Villarreal
David Alejandro Huertas Trujillo
Brayan José Vargas Plaza
Bayron Daymiro Campaz Hurtado
Juan David Díaz Monsalve
Santiago Gutiérrez Bolaños
Cristhian Eduardo Castillo Meneses
Kevin Gianmarco Zarama Luna
Javier Díaz Cely

Editorial Universidad Icesi, 2022

Ciberseguridad: los datos tienen la respuesta

1 ed. Cali, Colombia. Universidad Icesi, 2022
270 p., 19x24 cm
Incluye referencias bibliográficas
ISBN: 978-628-7538-78-8
https://doi.org/10.18046/EUI/ee.4.2022

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

Facultad de Ingeniería

Rector: Esteban Piedrahita Uribe
Decana Facultad de Ingeniería: Norha Villegas
Coordinador editorial: Adolfo A. Abadía

Edición, producción y diseño: José Ignacio Claros V.
Diseño de portada: Jhoan Sebastián Pérez.
Impresión: Carvajal Soluciones de Comunicación.

Publicado en Colombia / *Published in Colombia.*

Editorial Universidad Icesi

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

Este libro presenta los resultados de una serie de proyectos de investigación realizados en el Grupo de Investigación en Informática y Telecomunicaciones (i2t) de la Facultad de Ingenieria de la Universidad Icesi, desarrollada conjuntamente entre estudiantes de su programa de Ingeniería de Sistemas y docentes investigadores de i2t. Los proyectos fueron concebidos en el laboratorio como parte de un plan dirigido a cubrir de manera ordenada la temática propuesta y sirvieron además como insumo para la elaboración de los trabajos de grado de los ahora ingenieros de sistemas (ver los créditos en la página 9).

Tabla de contenido

Presentación

Distintos sectores han venido trabajando en la implementación de modelos basados en aprendizaje de máquina (*machine learning*) para el apoyo de procesos, la toma de decisiones y el desarrollo de nuevas herramientas digitales. Durante los últimos años esta subárea de la inteligencia artificial ha logrado una gran acogida en los distintos enfoques de la ciberseguridad, entre ellos: el ofensivo, el defensivo y el ético.

Las soluciones de *machine learning* se basan en la aplicación de algoritmos matemáticos y del concepto de aprendizaje a través de conjuntos de registros y variables, enfoque que incluye una serie de prácticas que parten de la metodología científica denominada ciencia de datos, la cual busca abstraer un patrón en la información que le dé respuesta a una pregunta, necesidad o hipótesis.

La ciberseguridad requiere de un esfuerzo constante que garantice soluciones y espacios ciberseguros. Los analistas e investigadores en seguridad informática se enfrentan a la complejidad de estudiar, día a día, casos que pueden ser o no una amenaza cibernética, un reto que involucra múltiples variantes —entre ellas: el crecimiento acelerado de la tecnología, la falta de personas especializadas en seguridad digital, las ciberguerras y las vulnerabilidades no detectadas—, que pueden ser significativas durante el estudio. A su vez, este puede verse afectado por el tiempo y los recursos, lo que puede conllevar no solo a problemas a nivel de negocio, sino también a afectar vidas.

La ciencia de datos ha ganado relevancia en los procesos investigativos porque permite apoyarlos, mejorar la toma de decisiones y así: dar respuesta a preguntas mediante la información que las organizaciones recolectan —

por ejemplo, en los *Security Information and Event Management* (SIEM)—, la cual representa el contexto de un incidente informático: ¿cuándo?, ¿dónde?, ¿quién?, ¿por qué?; y crear mecanismos para la prevención, detección y respuesta a eventos delictivos.

Este libro inicia presentando la importancia de la información y la manera de utilizarla para el proceso de ciencia de datos, para luego introducir los fundamentos teóricos de ambas disciplinas y presentar su aplicación en ocho proyectos de investigación, los cuales buscan orientar al lector sobre cómo utilizar la inteligencia artificial tanto desde la perspectiva defensiva como en la ofensiva.

En el aspecto defensivo, en el texto se aborda la realización de experimentos precisos para el desarrollo de modelos para la detección de *malware* en dispositivos Android, *cryptojacking*, *deepfakes* y *botnets* maliciosas; y en el enfoque ofensivo, se exploran modelos para la generación de contenido multimedia no legítimo y se introduce el concepto de aprendizaje seguro y la aplicación de *adversarial machine learning* como enfoque para encontrar los valores que permiten sesgar los resultados de un modelo de aprendizaje de máquina.

Finalmente, el lector podrá conocer los estudios más recientes de ambas áreas, las ventajas y desventajas de las técnicas de análisis para la detección de amenazas cibernéticas y los caminos aún por explorar en investigación sobre seguridad informática.

Capítulo 1
Los datos tienen la respuesta

Christian Camilo Urcuqui, Andrés Navarro

Vivimos en un mundo hiperconectado gracias a la Internet, una red compuesta por dispositivos y enlaces, un medio que ha permitido mejorar la comunicación y las actividades diarias. Es un "mundo digital", en él, no solo hay registros de un individuo desde antes de su nacimiento, sino que dichos registros se incrementan exponencialmente con el transcurrir de los días, un comportamiento que es difícil de modelar en una función, en un mecanismo que permita calcular los volúmenes de datos de cada persona. En este escenario, se genera más información a medida que el sujeto va progresivamente sumergiéndose en las tecnologías.

Es un trabajo de cálculo complejo que hipotéticamente podría suceder en un entorno donde la red de computadoras que se conoce hoy esté compuesta por solo un eje central (uno o más nodos interconectados), administrador de los recursos (estructura conocida como cliente-servidor); pero la realidad es que vivimos en una red de redes no regulada por un solo agente.

Sin embargo, existen sistemas cerrados donde es posible la recolección y análisis de los nodos que interactúan con los servicios y la infraestructura, una zona regulada o, en otras palabras, una pequeña red de nodos; en este tipo de escenarios se integran los dispositivos encargados de la recolección y el almacenamiento de la información, por ejemplo: los equipos de interconexión (*switches* y *routers*); las bodegas de datos, eso es las bases de datos relacionales y no relacionales, los sistemas de archivos *data lake* y las soluciones data *warehouse*; los dispositivos de entrada y salida (*mouse*, impresoras, monitores y cámaras web); y aquellos dedicados a la administración de los servicios (servidores).

Los flujos de información y la administración de los recursos de un sistema dependen de la topología de la red, es decir, para el primer caso la trasmisión de

la información entre dos agentes A y B puede ser en unidireccional (mensajes en una sola dirección) o bidireccional (flujo de mensajes en ambos sentidos); por otra parte, en el segundo enfoque es posible analizar quién es el responsable del control de los recursos, ahí cabe citar algunos ejemplos mencionados por Tanenbaum [1].

Una topología de bus (Figura 1.1) consiste en un enlace que conecta a todos los nodos, donde la información fluye de manera bidireccional. Si bien es un tipo de red que no se ve afectada por el mal funcionamiento de uno de los nodos, su desventaja radica en poder ubicar a un nodo con fallas cuando la red es muy grande.

Una topología de estrella (Figura 1.2) cuenta con un nodo central —*switch* o *hub*—, que se encarga de recibir los recursos/mensajes de los emisores y redireccionarlos a su respectivo receptor. En este esquema existe un gestor de los flujos que implementa mecanismos de control; puede presentar problemas de seguridad y disponibilidad de la red si llega a no funcionar correctamente.

En una topología en anillo (Figura 1.3), los recursos se transmiten en una misma dirección y pueden pasar por uno o más nodos antes de llegar a su respectivo receptor. Cada nodo actúa como un repetidor para reforzar la señal; puede presentar problemas de disponibilidad si alguno de los nodos llega a fallar, y de privacidad si la transmisión o los mensajes no incorporan mecanismos de encriptación.

Si bien la topología define el diseño, una red se puede clasificar dependiendo de su arquitectura, específicamente de quien administra los recursos del sistema para los nodos consumidores. Como se mencionó, la arquitectura que integra a un agente encargado de la gestión de los recursos se conoce como cliente-servidor. Por otra parte, una red en donde existen varios nodos que desempeñan un rol tanto de consumidores como de proveedores tiene una arquitectura *peer-to-peer* (P2P), una variante que busca descentralizar los recursos y dejar el control a una serie de nodos. Esta arquitectura suele encontrarse en solucionadas basadas de *blockchain* y en aplicaciones para compartir archivos, tales como Ares.

Conocer los nodos de la red y el flujo de los recursos y sus permisos permite incorporar mecanismos para ejecutar tareas de monitoreo y análisis útiles para detectar eventos atípicos al comportamiento habitual del perímetro de una zona tecnológica. Algunos controles basados en firmas y reglas destacados

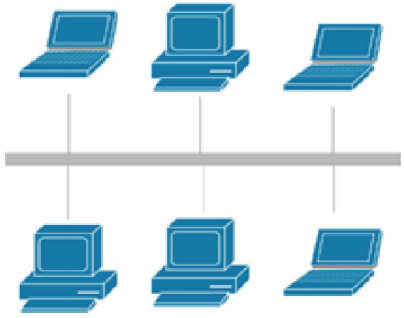

Figura 1.1 Topología de bus

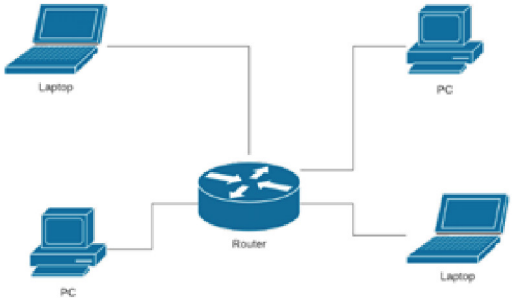

Figura 1.2. Topología de estrella

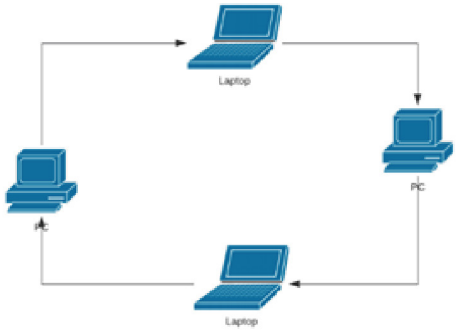

Figura 1.3. Topología de anillo

son: *firewalls*, sistemas de detección de intrusos (*Intrusion Detection System*, IDS) y sistemas de prevención de intrusos (*Intrusion Prevention Systems*, IPS).

Aunque un ciberdelincuente puede tener múltiples propósitos [2], si el enfoque es solamente en las actividades digitales dirigidas a ganar acceso a la infraestructura de una organización, el individuo puede ejecutar sus actividades maliciosas desde afuera del perímetro digital o dentro del nivel físico —sea ejecutándolo sí mismo o a través de un tercero—.

Si se analiza el escenario desde una caja negra por parte del cibercriminal, el sujeto ejecutará tareas de escaneo y enumeración sino ha conseguido éxito en la fase de búsqueda de fuentes abiertas de información; si el enfoque es solamente en las dinámicas de un escaneo de servicios contra los nodos del perímetro objetivo, esta actividad debería generar al menos un registro que podría abstraerse de la comunicación de la red o de los archivos de *logs*.

Si el punto de partida es que la red de computadoras genera bastas cantidades de datos, a ritmos constantes, ajustada a un comportamiento habitual de usuarios-nodos, estos insumos podrían ser elementos de valor para la identificación de un patrón que abstraiga el contexto de una zona perimetral, un mecanismo que podría permitir la identificación de actos inusuales de la red a partir de insumos que pueden estar en distintos tipos de formatos, por ejemplo en los *netflows*, las consultas DNS y los *logs*; en otras palabras, el patrón seria mapeado a un conjunto de reglas o a un modelo matemático que permitiría análisis continuos para la detección de amenazas dentro o hacia el perímetro de un objetivo.

En conclusión, el reto es conocer ese conjunto de datos y el patrón que permita deducir la diferencia entre actividades legítimas y maliciosas. Es aquí donde se entrelazan la ciberseguridad y la ciencia de datos para proponer soluciones y dar respuestas aproximadas a retos no resolubles (no actualmente) por los sistemas determinísticos convencionales.

Finalmente, si se parte de la hipótesis de que al menos un registro es generado durante un ataque informático, se podría afirmar que los datos son la respuesta para la detección proactiva de compromisos [3].

Gran parte de las decisiones que se toman actualmente están basadas en información. Un caso de ejemplo son los eventos desarrollados durante la pandemia por el Sars-Cov-2 (Covid-19), cuando gran parte de los organizamos de control sanitario utilizaron mecanismos para el monitoreo de personas

diagnosticadas con la enfermedad con el fin de rastrear su posición geoespacial y conocer la ubicación de las mayores concentraciones de individuos en zonas específicas. Este tipo de insumo le sirvió a las organizaciones sanitarias para proponer zonas de contención, buscando así reducir el grado de propagación de la enfermedad entre las personas. Ahora, pensando en un escenario de ciberseguridad, revisemos las soluciones que apliquen los análisis continuos para la detección de amenazas cibernéticas a partir de los flujos constantes de información de la red de computadoras.

Actualmente podemos encontrar soluciones comerciales, académicas y de acceso libre para la recolección, el preprocesamiento y la aplicación de modelos matemáticos para mecanismos de defensa cibernética. Como se mencionó, una red puede generar grandes volúmenes de información (*big data*) dependiendo del número de nodos que la compongan, información que trae consigo los insumos de valor para el análisis; estos inmensos datos necesitan de una infraestructura que permita escalar, de acuerdo con los requerimientos del sistema y el comportamiento de los datos.

Un contexto suele relacionarse con *big data* cuando se presenta alguna de las siguientes características en los datos: volumen, velocidad o variedad. Además, de las anteriores "v" de los datos, se pueden incluir: valor, visualización y veracidad como propiedades subyacentes al proceso de la analítica de datos. Una infraestructura tecnológica que soporte las anteriores variables debe por lo menos tener un mecanismo de almacenamiento, un motor de procesamiento y un gestor de los recursos.

Gran parte de las soluciones actuales de *big data* incluyen motores de transferencia de datos, motores de consulta, nodos de analítica y administradores de flujos de tareas y de coordinación. Entre las tecnologías de libre acceso para procesamiento distribuido de datos se destaca Apache Hadoop [4], una solución que ha ganado relevancia en la comunidad de desarrolladores, entre cuyos modelos se encuentran: MapReduce, para computación paralela de grandes colecciones de datos; y Spark, para flujos de información a grandes velocidades.

Una vez introducidos algunos elementos de computación para grandes colecciones de datos, cabe revisar el escenario de ciberseguridad. El cibercrimen es un fenómeno de estudio bastante interesante y de ardua labor debido a sus complejas variables, entre ellas: la astucia del atacante, las posibilidades que brinda actualmente la tecnología para que el acto malicioso sea difícil

de rastrear, las malas prácticas en el desarrollo de soluciones seguras debido al desconocimiento o a las exigencias del mercado, las amenazas persistentes avanzadas (*Advanced Persistent Threats*, APT) y los ataques de día cero. Para el análisis de amenazas cibernéticas es posible encontrar varias herramientas y técnicas que parten de dos enfoques o métodos para el análisis de amenazas cibernéticas: los análisis estático y dinámico.

El análisis estático se compone de las técnicas y herramientas para el análisis del software en el código fuente y tiene como objetivo encontrar los elementos maliciosos a partir de la ingeniería inversa, el llamado entre clases y métodos, la estructura, la memoria y el análisis del software en lenguaje de bajo nivel. Existen mecanismos de encriptación que permiten a un cibercriminal cifrar el *malware* o aplicar técnicas de ofuscación que dificultan la tarea del analista. El análisis dinámico, por su parte, es un enfoque orientado a encontrar insumos de valor del software a partir de su ejecución. En él se incluye un conjunto de métodos que tienen como objetivo crear un ambiente (físico o virtual) y aplicar herramientas para la captura de información del *malware*, por ejemplo, el tráfico de red. Un reto que se presenta al analista es el uso de recursos, si el investigador desarrolla o aplica ambientes artificiales y mecanismos para generar interacciones, debe afrontarse con los códigos maliciosos que cuentan con las capacidades de detectar entornos simulados.

Los análisis de amenazas cibernéticas mencionados han permitido el desarrollo de mecanismos de detección básicamente agrupados en dos conjuntos: el primero, orientado a la identificación a partir de firmas, específicamente una base del conocimiento de los códigos maliciosos reportados como firmas digitales (*hash*) que sirven como validación contra cada aplicación evaluada; el segundo, donde se cuenta con mecanismos que parten del monitoreo constante al sitio que se desea proteger, del cual se abstraen subelementos no muy variables en el tiempo para con el fin de crear un perfil, donde la representación del objetivo y de su monitoreo servirá para la identificación de anomalías, y el desarrollo de alertas y tareas de corrección.

Como expusieron Urcuqui et al. [5], la ciberseguridad y la ciencia de datos pueden ser utilizadas en conjunto con el fin de proponer mecanismos de *machine learning* para la detección de amenazas digitales. Como se introdujo en el transcurrir del presente capítulo, los datos son una fuente que puede ser aprovechada para encontrar actos maliciosos de manera predictiva y proactiva, para lo cual la ciencia de datos tiene como objetivo encontrar el conjunto

de patrones en los datos que permitan generar modelos que representan al contexto de la información, propósito que se lleva a cabo a partir de la aplicación de métodos de estadística, minería de datos, *machine learning* y visualización. El *machine learning* es una subárea de la inteligencia artificial encargada de desarrollar modelos de software que cuentan con la capacidad de aprender en entornos variables, es decir, sin ser programados de forma explícita. Ambos conceptos ser abordan con mayor profundidad y detalle en las siguientes secciones de este texto.

Dentro del campo de la ciberseguridad se encuentran los sistemas SIEM (*Security Information and Event Management*), un conjunto de soluciones que incorpora distintas tecnologías que tienen como objetivo la recopilación de información y la detección rápida de amenazas cibernéticas. Este tipo de mecanismos son un claro ejemplo del uso de los datos para análisis en tiempo casi real y de su importancia en la toma de decisiones. Los SIEM suelen proveer análisis descriptivos a través de: tableros de control; la incorporación de mecanismos de detección a partir de firmas y anomalías; y el uso de modelos de *machine learning* para la detección de amenazas a través de patrones en las variables.

Un proyecto de libre acceso que toma como base el análisis de datos y el uso de *machine learning* de la red de computadoras es Apache Spot [6]. Para la prevención de ataques, esta solución parte del análisis perimetral o de movimientos laterales a través de los datos de telemetría, es decir, de información de *netflows*, consultas DNS e información de PROXY. Como se menciona en su documentación, los grandes conjuntos de datos son procesados a través de un ecosistema tecnológico que incluye Apache Hadoop Spark.

La detección de los datos atípicos (anomalías) se lleva a cabo a través de un conjunto de modelos de *machine learning*, uno de los cuales aplica un enfoque de procesamiento de lenguaje natural [7] que busca construir un diccionario de palabras que representan a la telemetría de la red que se quiere defender, un insumo que permite identificar la frecuencia de los datos transmitidos (palabras) y así conocer los inusuales. En la Figura 1.4 se presentan las variables que representan a una palabra para los datos de una dirección IP de un *proxy* (cada dispositivo en la red se representa como un texto compuesto de palabras).

Cada dispositivo y sus palabras son luego agrupados a través del algoritmo generativo *Latent Dirichlet Allocation* (LDA), mecanismo que identifica a las palabras comunes en todos los textos con el fin crear tópicos (ver Figura 1.5).

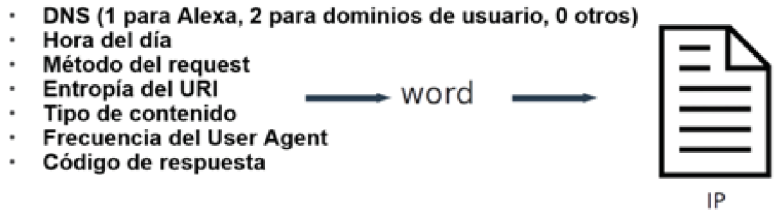

Figura 1.4. Variables que conforman una palabra de un proxy

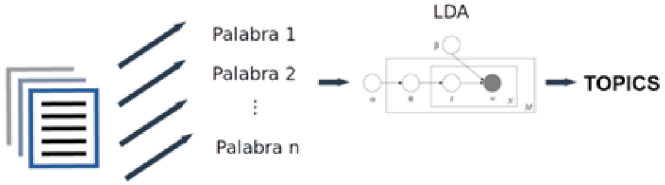

Figura 1.5. Aplicación de LDA y generación de los tópicos

La definición de la cantidad de tópicos se puede ajustar a través de una técnica de coherencia de datos. Una vez definidos tanto el modelo LDA como los tópicos, se procede a crear dos estructuras de probabilidades: una compuesta por el IP y el tópico-wp- (ver parte alta de la Figura 1.6); la otra con la normalización de las palabras por tópico-ipt- (ver parte baja de la Figura 1.6).

Las dos estructuras ayudan a definir una función de *score* (Ecuación 1.1) y el límite (*threshold*) para encontrar los datos atípicos de la comunicación de los nodos de la red.

$$\sum_{a=1}^{n} \mathrm{wp}_{a\,x}\,\mathrm{ipt}_{a} \qquad (1.1)$$

En una prueba de laboratorio con etiquetado humano de los registros maliciosos es posible evidenciar (Figura 1.6), cómo los datos no recurrentes de la red suelen presentar una puntuación mucho menor que los datos benignos.

El anterior ejemplo es una aproximación que se ha realizado para la creación de un perfil de la red a partir de los datos de *proxy*, un punto clave para tener en

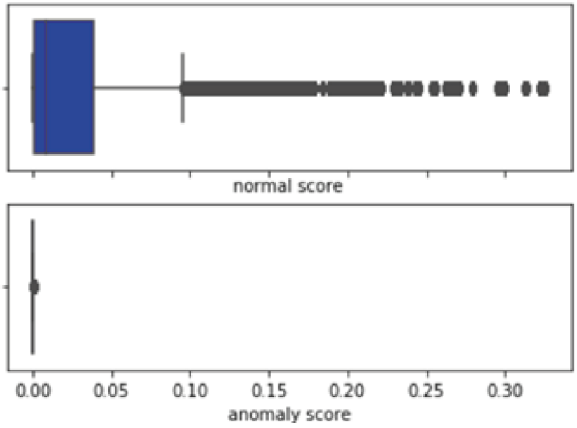

Figura 1.6. Prueba de Apache Spot - LDA

cuenta es que entre mayor sean los datos para la construcción de las estructuras mencionadas, la función de discriminación será mucho más relevante debido a que el conjunto de palabras será mucho más aproximado a la realidad, es decir, la frecuencia de los datos típicos de la red debería ser mucho mayor.

Como se mencionó, Apache Spot es un proyecto soportado por la comunidad de software de libre uso, capaz de detectar ataques de denegación de servicio con una efectividad superior al de un IDS [8]. Shield [9], por su parte, es una solución enfocada en proveer seguridad como un servicio (*Security as a Service*, SecaaS) a través de funciones de red virtualizadas y de la aplicación de analítica en grandes volúmenes de información [10] y marca otra parada en cómo la ciberseguridad ahora se ofrece como un servicio, más aún con la importancia de la migración de soluciones *on-premise* a ambientes de computación en la nube; por lo anterior, hace unos años Alphabet ha incorporado entre su conjunto de soluciones de GCP (*Google Cloud Computing*) las herramientas de *Chronicle Cyber Security Analytics*, otro proveedor de ciberseguridad que parte de los datos de la red de las computadoras para la detección de amenazas cibernéticas [11].

Ahora corresponde revisar los conceptos de *machine learning*, ciencia de datos, ciberseguridad y comunicación en la red de computadoras, es decir, los conceptos teóricos base para el desarrollo de las próximas secciones, las cuales abordarán las metodologías para la creación de los mecanismos de defensa para detección continua de amenazas cibernéticas a partir del uso de la inteligencia artificial y los datos.

Referencias

[1] A. S. Tanenbaum, *Redes de computadoras*. Pearson Educación, 2003.

[2] K. S. Choi, y M. Toro-Álvarez, *Cibercriminología: guía para la investigación del cibercrimen y mejores prácticas en seguridad digital*. Bogotá, Colombia: Fondo Editorial UAN, 2017.

[3] C. C. Urcuqui, "Los datos son la respuesta," ponencia presentada en: IEEE ComSoc Latam, Nov. 2020.

[4] *Apache Hadoop*. Disponible: https://hadoop.apache.org/´

[5] C. C. Urcuqui, M. G. Peña, J. L. Quintero, y A. Navarro, *Ciberseguridad: un enfoque desde la ciencia de datos*. Cali, Colombia: Universidad Icesi, 2018.

[6] *Apache Spot Project*. Disponible: https://spot.apache.org/

[7] C. C. Urcuqui, "Illuminating threads in the network using data analytics," ponencia presentada en: II Congreso de Expertos en Ciberseguridad y Ciberdefensa, Popayán, Colombia, Nov. 2021.

[8] C. M. Mathas, O. E. Segou, G. Xylouris, D. Christinakis, M. A. Kourtis, C. Vassilakis, y A. Kourtis, "Evaluation of Apache Spot's machine learning capabilities in an SDN/NFV enabled environment," en: *Proceedings of the 13th International Conference on Availability, Reliability and Security* (pp. 1-10), Ago. 2018.

[9] *EU SHIELD project, (Securing against intruders and other threats through an NFV-enabled environment)*. Disponible: https://www.shield-h2020.eu/

[10] G. Gardikis et al., "SHIELD: A novel NFV-based cybersecurity framework," en: *2017 IEEE Conference on Network Softwarization (NetSoft)*, 2017, pp. 1-6, doi: 10.1109/NETSOFT.2017.8004228.

[11] *Chronicle*. Disponible https://chronicle.security/

Capítulo 2
Marco conceptual

Brayan Henao, Juan Sebastián Prada, Andrés Felipe Pérez, Steven Bernal, Julio César Gaviria, Ánderson Ramírez, Jhoan Steven Delgado, David Alejandro Huertas, Brayan Vargas, Bayron Campaz, Juan David Díaz, Santiago Gutiérrez, Cristhian Eduardo Castillo, Kevin Zarama, Christian Urcuqui, Andrés Navarro, Javier Díaz Cely

2.1. Ciberseguridad

Ciberseguridad es el área de las ciencias de la computación encargada de proponer mecanismos para la protección de sistemas informáticos, redes de telecomunicaciones e información de posibles acciones maliciosas provenientes de cibercriminales [1], como se explica en la Tabla 2.1, se divide en seguridad de las red, seguridad de las aplicaciones, seguridad de la información, seguridad de las operaciones, recuperación ante desastres y continuidad del negocio; y educación del usuario.

Tabla 2.1. Divisiones de la seguridad informática

División	Descripción
Seguridad de la red	Prácticas para asegurar una red de computadores frente a intrusos.
Seguridad en las aplicaciones	Actividades enfocadas en mantener el software y los dispositivos libres de amenazas, ya que una aplicación comprometida puede proveer acceso a datos sensibles.
Seguridad de la información	Protección de la integridad y privacidad de los datos tanto en almacenamiento como en tránsito.
Seguridad en las operaciones	Procesos y decisiones para manejar y proteger los activos digitales.
Recuperación ante desastres y continuidad del negocio	Define de qué manera una organización responde ante un incidente que compromete su seguridad informática y otros eventos que pueden ocasionar un paro en las operaciones o la pérdida de datos.
Educación del usuario	Asegura el factor de seguridad informática más impredecible: las personas, pues cualquiera puede accidentalmente introducir un virus por no seguir las buenas prácticas de seguridad. Enseñarles a seguir estas prácticas es vital para la seguridad de la empresa.

En el campo de la ciberseguridad se han realizado varias propuestas, entre ellas: marcos de trabajo para la evaluación de ciberamenazas y prácticas y herramientas para el desarrollo de software y hardware seguros. Ellas han permitido defender los sistemas contra cibercriminales y software malicioso, sin embargo, el desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC), la presencia de nuevas vulnerabilidades y los ataques del día cero han hecho que la ciberseguridad requiera de investigación constante para mitigar los riesgos. En consecuencia, se han propuesto métodos para la evaluación de amenazas cibernéticas que han permitido el desarrollo de mecanismos de defensa contra los ataques informáticos, algunos de los más representativos son los análisis estático, dinámico e híbrido.

El análisis estático se encarga de detectar ciberataques web utilizando patrones o reglas, sin llegar a ejecutar el código [2], [3]. La efectividad de este método se basa en que los patrones que utilizan las violaciones se conocen previamente, por lo que pueden ser detectados fácilmente; sin embargo, si no se tiene conocimiento del ciberataque o si este se "camufla" de alguna forma, no podrá ser detectado. Por ello, para este tipo de análisis es importante tener una fuente de datos de patrones de ciberataques actualizada.

La principal ventaja de este método es su rapidez en procesamiento [3], [4]. Al ser un método basado en detección de patrones y comparaciones hace que sea rápido al momento de procesar e identificar un posible ciberataque; su principal debilidad es la ofuscación de código malicioso [2], [3], que consiste en camuflar el código para ocultar su verdadera forma. Una manera de aplicarlo es a través del *encode*, mecanismo utilizado para hacer una traducción de los caracteres no imprimibles o especiales a caracteres aceptados universalmente por servidores web y buscadores, dificultando así su identificación. Para solventar esta debilidad existen aproximaciones que permiten realizar la traducción del código, aunque no siempre son efectivas. Adicionalmente, este método puede producir falsos positivos al encontrar estructuras que pueden parecer maliciosas en peticiones benignas, lo cual es molesto al momento de utilizar una aplicación.

El análisis dinámico se encarga de detectar las amenazas digitales durante su ejecución. Este método se basa en el uso de detección por anomalías, es decir, establece un comportamiento normal de los usuarios de la aplicación o de la página web para luego hacer una clasificación de las peticiones entre normales y anómalas [4]. Para el análisis mediante anomalías, se necesitan

definir una serie de factores para garantizar una cobertura de todos los posibles factores de ciberataque [5]. Estos pueden ser, entre otros: longitud de cadenas, distribución de caracteres y gramática de la cadena. Una vez establecidos estos factores, se realiza una fase de adecuación para definir un rango de aceptación. Todas aquellas peticiones que pasen de estos rangos serán clasificadas como anómalas, lo que indica un posible ciberataque.

Su principal ventaja es el alto porcentaje de detección, ya que al tener diferentes análisis, puede cubrir un rango más amplio que simplemente la estructura o contenido de la petición, lo que le permite ser un método más efectivo [5]. Su principal desventaja es el tiempo que toma para realizar el análisis de una petición. Al ser dinámico, cada petición es diferente y requiere su propio tiempo de análisis, lo cual incrementa considerablemente el tiempo de respuesta. Por otra parte, al igual que en el análisis estático, existe la probabilidad de generación de falsos positivos al encontrar comportamientos anómalos en las peticiones benignas.

El análisis hibrido, por su parte, como su nombre lo indica es la combinación de los métodos estáticos y dinámicos para el análisis de los ataques informáticos [4]. Con ella se busca obtener lo mejor del análisis estático, su rapidez de detección, y lo mejor del análisis dinámico, su efectividad de detección.

2.1.1. Ataques informáticos y amenazas cibernéticas

Un ataque informático es el producto de la motivación de un individuo que aplica al menos un método sobre una o más vulnerabilidades de un objetivo; dependiendo del contexto, un atacante puede aplicar varios tipos de ataque informático, los cuales se pueden clasificar en: pasivos, activos, cercanos, con privilegios o distribuidos, como se explica en la Tabla 2.2.

Tabla 2.2. Tipos de ataque informático

Tipo	Características
Pasivo	Acciones donde el atacante no interactúa directamente con el objetivo, por ejemplo el monitoreo del tráfico de la red y los flujos de datos que se transmiten en una red de computadoras.
Activo	Acciones que conllevan a la interrupción de una comunicación o de los servicios de un sistema o rompen los mecanismos de seguridad digitales, tales como los ataques de denegación de servicios (*Denegation of Services, DoS*], los secuestros de sesión (*sessions hijacking*) y la inyección SQL (*SQL injection*).

Tabla 2.2. Tipos de ataque informático (continuación)

Tipo	Características
Cercano	Acciones donde el atacante tiene una aproximación física al objetivo, es el caso de los ataques realizados mediante ingeniería social, tales como: *shoulder surfing* (mirar por encima del hombro), *eavesdropping* (escuchar a escondidas) y *dumpster diving* (búsqueda en la basura).
Explotación de privilegios	Acciones donde los empleados u otras personas con privilegios dentro una organización forman parte del conjunto de atacantes; son muy peligrosos debido a su capacidad de acceso al perímetro del objetivo.
Distribuido	Acciones donde el atacante altera hardware o software antes de su instalación o adquisición por el usuario.

Existe una amplia variedad de familias de software malicioso (*malware*) creado por los cibercriminales. El *malware* es un conjunto de herramientas utilizado por personas malintencionadas durante el ciberataque a un objetivo. Los ciberataques suelen contar con una serie de fases representadas en la metodología *Cyber Security Kill Chain*, término que se deriva del concepto militar *kill chain*, cuya finalidad es definir los pasos que un enemigo sigue para atacar a un objetivo. En el contexto actual, incluye la serie de pasos que comúnmente recorren los atacantes cuando intentan vulnerar un sistema, los cuales se describen en la Figura 2.1.

Para la detección de ciberataques se han propuesto dos marcos de trabajo: el método basado en firmas, que tiene como finalidad la detección de 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 que pueden ser representadas con *hash* y otra enfocada en el monitoreo y la detección de anomalías o cambios no registrados.

De otro lado están las pruebas de ciberseguridad que buscan encontrar y mejorar las vulnerabilidades de un sistema informático; entre ellas, cabe citar una metodología de pruebas de penetración (*pentesting*), el uso de software o incluso hardware para explotar una vulnerabilidad (*exploit*) y de cómo aprovecharla para la ejecución de otras actividades sobre el objetivo (*payload*). Un *payload* es un software desarrollado para cargar actividades digitales en la máquina objetivo, por ejemplo el *reserve Shell*, útil para otorgar vías de comunicación entre dos agentes, mientras que un *exploit* es un software que se

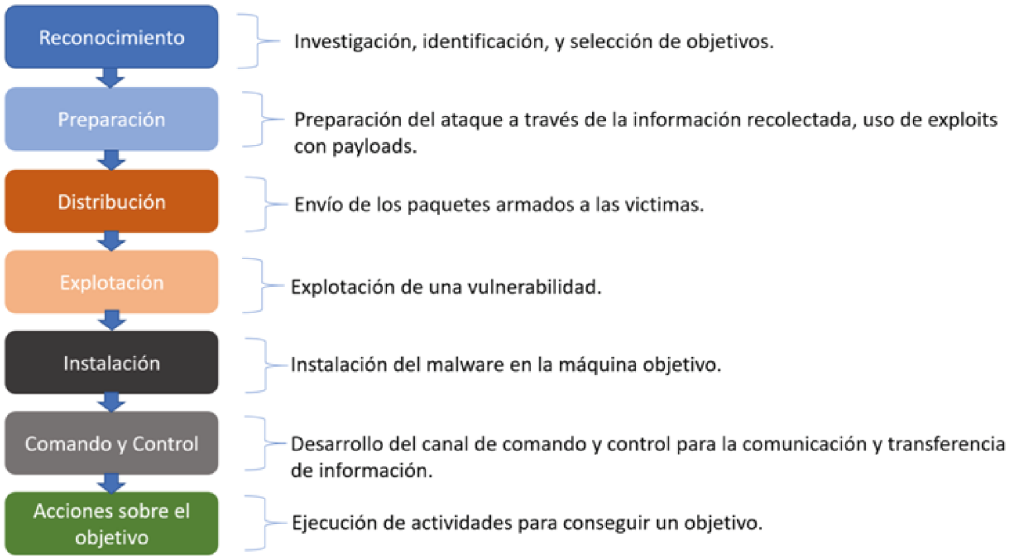

Figura 2.1. Fases de un ciberataque

aprovecha de vulnerabilidades presentes en el software, las redes o el hardware para ejercer una acción sobre el objetivo [6].

Por su parte, el *pentesting* o prueba de penetración es una práctica que se realiza sobre un sistema para verificar las medidas de seguridad, confirmar su correcta aplicación y, en caso de encontrar vulnerabilidades, medir el alcance que pueden tener. Una vez que se encuentran las vulnerabilidades, se aplican los parches, las actualizaciones o las medidas complementarias que se requieran, para así reducir los riesgos y construir un sistema robusto, resistente al mayor número posible de ataques que existan en ese momento [7]. Debido a que es una práctica que requiere estar al tanto de las últimas vulnerabilidades descubiertas, es necesario que se realice periódicamente para así garantizar la actualización de los componentes del sistema.

Este tipo de prueba se realiza desde tres ópticas: caja blanca, caja gris y caja negra, conceptos que se refieren a la cantidad de conocimiento y al alcance qué se tiene sobre un sistema: en la caja blanca o "de cristal" se conoce todo, la arquitectura, las tecnologías, la base de datos y la información relacionada con el objetivo; en la gris, se conocen solo algunos factores; y en la negra no se tiene información y solo se puede interactuar con el sistema como un usuario final.

2.1.2. Riesgos cibernéticos

Un riesgo cibernético se refiere al grado de incertidumbre en que un evento adverso podría ocasionar un impacto; se puede contemplar como la posibilidad de la materialización de una amenaza cibernética por el impacto en un activo. Los riesgos son subyacentes a cada organización, es decir, dependen del contexto y no siempre son los mismos para cada institución. Un método para su categorización y medición es través de una matriz que incluye como variables la probabilidad de ocurrencia y el grado de la consecuencia.

Existen distintos riesgos que son inherentes al tipo de tecnología, para mencionar un conjunto conocido, se cita aquellos relacionados con las aplicaciones web, debido a su frecuencia de uso en la vida diaria de las personas gracias a Internet.

El *Open Web Application Security Project* (OWASP) es una comunidad abierta dedicada a desarrollar y mantener aplicaciones y API que sean confiables. Adicionalmente ofrece una serie de recursos para incrementar la seguridad de las aplicaciones web y publica periódicamente un Top 10 con los riesgos informáticos más críticos, incluyendo información sobre sus características, sus posibles repercusiones y una serie de recomendaciones para evitar el riesgo. La Tabla 2.3 presenta este Top 10 para 2021 en orden de criticidad, de mayor a menor.

Tabla 2.3. OWASP Top 10 / 2021

Riesgo	Descripción
Rotura del control de acceso (A01:2021)	El control de acceso aplica una política tal que los usuarios no pueden actuar fuera de sus permisos previstos, las fallas generalmente conducen a la divulgación, modificación o destrucción no autorizada de todos los datos o a la realización de una función comercial fuera de los límites del usuario. Las vulnerabilidades comunes de este tipo: la violación del principio de mínimo privilegio o denegación por defecto; la omisión de las comprobaciones de control de acceso mediante la modificación de la URL, el estado interno de la aplicación o la página HTML o el uso de una herramienta de ataque que modifica las solicitudes de API; permitir ver o editar la cuenta de otra persona; el acceso a la API sin controles de acceso para POST, PUT y DELETE; la elevación de privilegios (actuar como usuario sin haber iniciado sesión / actuar como administrador cuando se ha iniciado sesión como usuario); la manipulación de metadatos para elevar los privilegios; la configuración incorrecta de CORS, que permite acceso a la API desde orígenes no autorizados o no confiables; y forzar la navegación a páginas autenticadas como usuario no autenticado o a páginas privilegiadas como usuario estándar.

Tabla 2.3. OWASP Top 10 / 2021 (continuación)

Riesgo	Descripción
Fallas criptográficas (A02:2021)	Se deben determinar las necesidades de protección de los datos en tránsito y en reposo, tales como: contraseñas, números de tarjetas de crédito, registros de salud, información personal y secretos comerciales, pues requieren protección adicional, principalmente si están sujetos a las leyes de privacidad. Para todos estos datos se debe preguntar: ¿se transmite algún dato en texto claro?, ¿se utilizan protocolos o algoritmos criptográficos antiguos o débiles de forma predeterminada o en código antiguo?, ¿están en uso las claves criptográficas predeterminadas?, ¿se generan o reutilizan claves criptográficas débiles o hace falta una gestión o rotación de claves adecuada?, ¿se registran las claves criptográficas en los repositorios de código fuente?, ¿se aplica el cifrado?, ¿el certificado del servidor recibido y la cadena de confianza están debidamente validados?, ¿se ignoran, reutilizan o no se generan vectores de inicialización suficientemente seguros para el modo criptográfico de operación?, ¿se utiliza un modo de operación seguro?, ¿se utilizan contraseñas como claves criptográficas en ausencia de una función de derivación de clave base de contraseña?, ¿se utiliza la aleatoriedad con fines criptográficos que no fueron diseñados para cumplir con los requisitos criptográficos?, ¿se utilizan funciones hash en desuso o no criptográficas cuando se requieren funciones hash criptográficas?
Inyección (A03:2021)	Algunas inyecciones comunes son SQL, NoSQL, comando OS, asignación relacional de objetos, LDAP e inyección de lenguaje de expresión y biblioteca de navegación de gráficos de objetos (OGNL). La revisión del código fuente es el mejor método para detectar si las aplicaciones son vulnerables a las inyecciones. Una aplicación es vulnerable a un ataque cuando: no valida ni filtra ni desinfecta los datos proporcionados por el usuario; las consultas dinámicas o las llamadas no parametrizadas se utilizan directamente en el intérprete; los datos hostiles se utilizan dentro de los parámetros de búsqueda de mapeo relacional de objetos para extraer registros confidenciales adicionales; los datos hostiles se utilizan o concatenan directamente; o el comando SQL o contiene la estructura y los datos maliciosos en consultas dinámicas, comandos o procedimientos almacenados.
Diseño inseguro (A04:2021).	Esta categoría incluye varias debilidades generadas por un diseño faltante o ineficaz. La razón por la que hay diferencias entre el diseño inseguro y la implementación insegura, es que tienen diferentes causas y soluciones. Un diseño seguro puede tener defectos de implementación que den lugar a vulnerabilidades que pueden ser explotadas, pero un diseño inseguro no puede solucionarse con una implementación perfecta pues los controles de seguridad necesarios para defenderse de ataques específicos, nunca se crearon.
Configuración incorrecta de seguridad (A05:2021).	La aplicación podría ser vulnerable si: no cuenta con el refuerzo de seguridad adecuado en cualquier parte de la pila de aplicaciones o los permisos están configurados incorrectamente en los servicios en la nube; las funciones innecesarias están habilitadas o instaladas; las cuentas predeterminadas y sus contraseñas aún están habilitadas y sin cambios; el manejo de errores revela rastros de pila u otros mensajes de error que incluyen demasiada información; las funciones de seguridad más recientes están deshabilitadas o no configuradas de forma segura; la configuración de seguridad en los servidores de aplicaciones, los marcos de aplicaciones, las bibliotecas y las bases de datos, entre otras, no se establecen con valores seguros; el servidor no envía encabezados o directivas de seguridad o no están configurados con valores seguros; el software está desactualizad

Tabla 2.3. OWASP Top 10 / 2021 (continuación)

Riesgo	Descripción
Componentes vulnerables y obsoletos (A06:2021).	Ocurre cuando: no se conocen las versiones de todos los componentes que se utilizan de los lados cliente y servidor, lo que incluye los componentes que usa directamente, así como las dependencias anidadas; si el software es vulnerable, no es compatible o está desactualizado, lo que incluye el sistema operativo, el servidor web/de aplicaciones, el sistema de administración de bases de datos, las aplicaciones, las API y todos los componentes, entornos de tiempo de ejecución y bibliotecas; no se buscan vulnerabilidades con regularidad y no se suscribe a los boletines de seguridad relacionados con los componentes que utiliza; no se corrige o actualiza la plataforma, los marcos y las dependencias subyacentes de manera oportuna y basada en el riesgo, lo que suele ocurrir en entornos en los que la aplicación de parches es una tarea mensual o trimestral bajo control de cambios, lo que deja a las organizaciones expuestas innecesariamente por días o meses; los desarrolladores de software no prueban la compatibilidad de las bibliotecas actualizadas, mejoradas o parcheadas; no se aseguran las configuraciones de los componentes.
Fallas en la identificación y autenticación del usuario (A07:2021).	La confirmación de la identidad del usuario, su autenticación y la administración de la sesión son fundamentales para la protección de ataques relacionados con la autenticación. Puede haber debilidades de autenticación si la aplicación: permite ataques automatizados como el relleno de credenciales, donde el atacante tiene una lista de nombres de usuario y contraseñas válidos; permite la fuerza bruta u otros ataques automatizados; permite contraseñas predeterminadas, débiles o conocidas, como "Password1" o "admin/admin"; utiliza una recuperación de credenciales débil o ineficaz y procesos de contraseña olvidada, como "respuestas basadas en el conocimiento", que no se pueden hacer seguras; Utiliza almacenamiento de datos de contraseñas de texto sin formato, encriptadas o con hash débil; tiene una autenticación multifactor faltante o ineficaz; expone el identificador de sesión en la URL; reutiliza el identificador de sesión después de un inicio de sesión exitoso; o no invalida correctamente los ID de sesión.
Fallas en la integridad del software y los datos (A08:2021).	El código y la infraestructura no protegen contra las violaciones de la integridad, por ejemplo, cuando una aplicación se basa en complementos, bibliotecas o módulos de fuentes, repositorios y redes de entrega de contenido que no son de confianza. Una canalización de CI/CD insegura puede presentar el potencial de acceso no autorizado, código malicioso o compromiso del sistema. Muchas aplicaciones actuales incluyen actualizaciones automáticas que se descargan y se aplican sin una adecuada verificación de integridad. Los atacantes, en consecuencia pueden cargar sus propias "actualizaciones", distribuirlas y ejecutarlas en todas las instalaciones.
Fallas de monitoreo y registro de seguridad (A09:2021).	El registro, la detección, la supervisión y la respuesta activa son insuficientes: eventos auditables, como inicios de sesión efectivos o fallidos y transacciones de alto valor no se registran; las advertencias y los errores generan mensajes de registro inexistentes, inadecuados o poco claros; los registros de aplicaciones y API no se supervisan en busca de actividad sospechosa; los registros solo se almacenan localmente; los umbrales de alerta apropiados y los procesos de escalada de respuesta no están implementados o no son efectivos; las pruebas de penetración y los análisis realizados por herramientas de pruebas de seguridad de aplicaciones dinámicas no activan alertas; o la aplicación no puede detectar, escalar ni alertar sobre ataques activos en tiempo real o casi en tiempo real.

Tabla 2.3. OWASP Top 10 / 2021 (continuación)

Riesgo	Descripción
Falsificación de solicitud del lado del servidor (A10:2021).	Ocurre cuando una aplicación web obtiene un recurso remoto sin validar la URL proporcionada por el usuario; permite a un atacante obligar a la aplicación a enviar una solicitud manipulada a un destino inesperado, incluso cuando está protegida por un *firewall*, VPN u otro tipo de lista de control de acceso a la red (*Access-Control List*, ACL).

2.1.3. Malware

Un *malware* (*malicious software*) es un software diseñado para hacer daño a un computador, teléfono inteligente, servidor u otro dispositivo electrónico [8], que la mayoría de los cibercriminales utiliza, principalmente, con fines de lucro o por activismo político [9]. En la Tabla 2.4 se presenta una descripción de las familias de *malware* más conocidas.

Tabla 2.4. Familias de malware [10]

Familia	Descripción
Gusanos de red	Emplean recursos de red, tales como redes locales y redes globales, para distribuirse; su velocidad de propagación es muy alta y tienden a consumir los recursos de la máquina a la que atacan.
Troyanos	Incluyen una gran variedad de programas que efectúan acciones sin que el usuario se dé cuenta y sin su consentimiento: recolectan datos y los envían a los criminales; destruyen o alteran datos con intenciones delictivas, causando desperfectos en el funcionamiento del computador, o usan los recursos del computador para fines criminales, como hacer envíos masivos de correo no solicitado.
Spyware	Software que permite recolectar la información sobre un usuario u organización de forma no autorizada, puede obtener datos sobre las acciones del usuario, el contenido del disco duro, el software instalado y la calidad y velocidad de la conexión, entre otras.
Adware	Malware relacionado con la publicidad que se muestra al usuario; la mayoría de estos programas se instala a través de software de distribución gratuita. La publicidad aparece en la interfaz. Su denominación cambia a spyware cuando ejecutan actividades maliciosas de recolección y envío de datos sensibles del usuario.
Rootkit	Colección de programas usados por una persona para evitar ser detectada mientras busca obtener acceso no autorizado a un computador, lo que logra reemplazando archivos o librerías del sistema o instalando un módulo de kernel. Después de instalar el rootkit como administrador del sistema, obtiene un acceso similar al del usuario, por lo general hackeando una contraseña o explotando una vulnerabilidad, lo que permite usar otras credenciales hasta conseguir el acceso de raíz

2.1.4. Botnets

Las *botnets* maliciosas son mecanismos compuestos por equipos comprometidos ilegalmente, denominados zombies o *bots* que se utilizan para múltiples fines, entre ellos: ataques de denegación de servicios distribuidos (*Distributed Denial of Services*, DDoS), *spam*, robo de información y minado ilegal de criptomonedas.

Estos mecanismos son difíciles de detectar, ya que día tras día los cibercriminales los hacen mucho más potentes mediante módulos de programación (*exploits*) que se introducen e integran a sus sistemas. Muchos de estos módulos se producen y venden en el mercado negro del *malware*, lo que permite desacoplar las partes del mecanismo en funciones importantes. Día a día, mediante técnicas sumamente creativas, los cibercriminales tienden a expandir aún más la red, haciéndola más fuerte, más madura y por ello difícil de erradicar.

Al principio era común ver mecanismos de esta índole utilizando topologías centralizadas, sin embargo, los cibercriminales descubrieron que esa no era la topología adecuada, ya que si se detectaba el servidor de comando y control (nodo central) podrían perder el control total de la *botnet*, por ello empezaron a utilizar topologías basadas en P2P, las cuales permiten que cada nodo administre sus propios recursos de manera autónoma, lo que dificulta la destrucción del mecanismo.

Las *botnet* presentan un ciclo de vida particular, similar al de la Figura 2.2, el cual consta de cuatro etapas: formación e infección; comando y control (C&C); ataque y post-ataque. En la primera, la *botnet* incrementa su tamaño mediante ingeniería social o realizando *inbound scan* directamente a las redes comprometidas; en la segunda, el cibercriminal sincroniza su estructura y los *bots* esperan órdenes para saber cuál será su siguiente objetivo; en la etapa de ataque, los *bots* generan suficiente poder computacional para derribar por medio de DDoS a un objetivo en particular; y en la última, el cibercriminal hace un barrido sobre la red para saber cuántos *bots* perdió durante el ataque.

2.1.5. DoS y DDoS

DDoS es uno de los ataques más populares; años atrás, solo se conocían los ataques de tipo DoS, los cuales eran dirigidos con el fin de deshabilitar la disponibilidad de un recurso, impidiendo el acceso a los usuarios durante un periodo de tiempo. Los recursos a los cuales un ataque DoS puede dirigirse

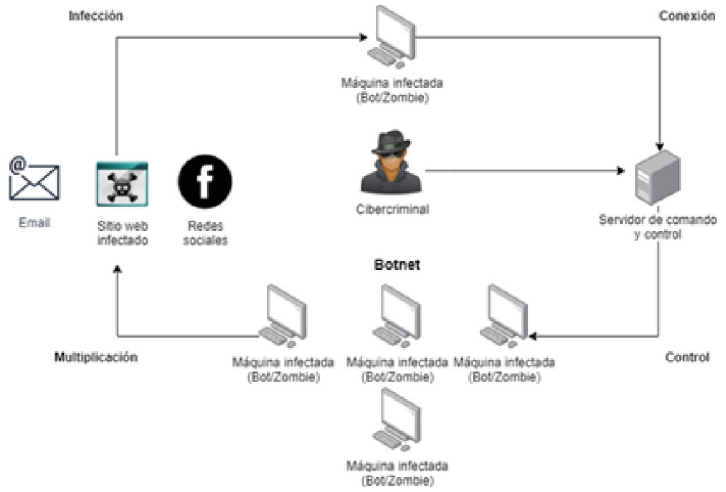

Figura 2.2. Estructura de una botnet maliciosa

abarcan desde nodos de procesamiento únicos o grupos de dispositivos hasta redes normales o empresariales.

De acuerdo con Aamir y Zaidi [11], estos ataques se pueden llevar a cabo contra diferentes capas del modelo OSI o TCP/IP. El tipo de ataque puede variar dependiendo de la técnica usada, el software utilizado, el protocolo a atacar y la naturaleza de la víctima (servidor, red o dispositivos).

Una vez los cibercriminales notaron la necesidad de mayor poder computacional para lanzar un ataque DoS, especialmente para generar tráfico masivo en un corto lapso, decidieron distribuir los ataques de tal forma que se permitiera utilizar muchos más equipos para enviar más tráfico hacia un objetivo. De esta manera, los cibercriminales lograron tomar control de otras máquinas, incluso sin el consentimiento de sus propios dueños, con el fin de incrementar el daño producido por un ataque de este tipo. Estos dispositivos generalmente eran vulnerados mediante *bugs* o brechas de seguridad en sus plataformas.

En la Figura 2.3 se presenta un ejemplo de la naturaleza de un ataque DDoS, donde el cibercriminal sincroniza y controla los nodos computacionales con el fin de denegar el servicio a la víctima, que en este caso puede ser un servidor localizado en una red empresarial.

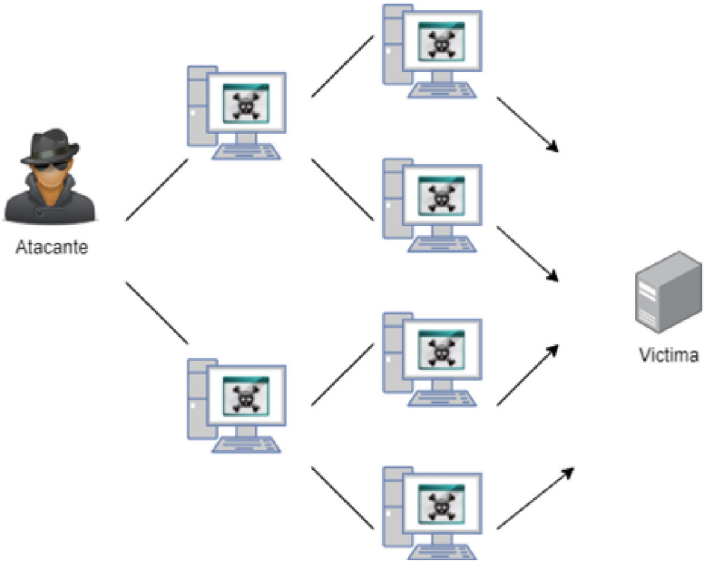

Figura 2.3. Ejemplo de ataque de denegación de servicio distribuido (DDoS)

Hoy en día existen varios tipos de ataques de denegación de servicio, tales como: SYN flood, UDP flood, HTTP flood, ping ofdeath, smurfattack, fraggleattack, slowloris, Application level attacks, NTP amplification, APDoS, Zero Day DDoS attacks [12].

2.1.6. Cryptojacking

El *cryptojacking* es un ataque que normalmente se ejecuta en segundo plano, no necesariamente requiere de archivos binarios para infectar a un equipo, puede usar *malware*, presenta variabilidad en los vectores de ataque, utiliza llamadas al sistema usando procesos autorizados y es multiplataforma. Esta amenaza suele evadir los mecanismos establecidos y genera un tráfico de red que puede pasar desapercibido en una red grande.

Esta variante parte del consumo de recursos de una víctima, la cual debe recibir y enviar información a un tercero (cibercriminal u otro agente malicioso); toda la información se transmite a través de la red de telecomunicaciones; es decir, si ocurrió un acto malicioso, los registros deben tener la historia del evento, independiente del estatus del minero (identificado o no) y del tráfico que exista en una red.

2.2. Redes y comunicaciones

Una red es un "sistema de comunicación que permite el intercambio de información entre un conjunto de dispositivos autónomos, geográficamente dispersos, sobre una base de tiempos total o parcial" [13]. Las redes surgen debido a esta necesidad de compartir datos en cierto momento, incluso cuando los dispositivos estén geográficamente dispersos. Para realizar dicho proceso, se debe tener un modelo referencia que permite "definir el alcance de la arquitectura de un sistema" [13]; es decir, un modelo conceptual que ayuda a entender la estructura de un sistema que cumple un fin, en este caso, transferir los datos. El modelo de referencia para las redes y comunicaciones es OSI (*Open Systems Interconnection*), el cual consta de siete capas (ver Figura 2.4).

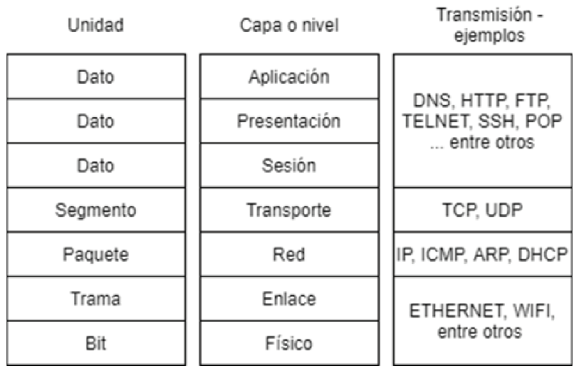

Figura 2.4. Modelo OSI

También existe el modelo TCP/IP, el cual se compone de cuatro capas:

- capa de aplicación, en donde se establecen los servicios enfocados al usuario y las aplicaciones finales;
- capa de transporte, en donde se establecen los protocolos que garantizan la gestión del envío y la recepción de la información, entre los que se destacan el *Transmission Control Protocol* (TCP), el *User Datagram Protocol* (UDP) y el *Internet Control Message Protocol* (ICMP);
- capa de Internet, en donde se establece la ruta que debe seguir un paquete en la comunicación de *host* a *host* (paquetes formados por datagramas que representan el mínimo bloque de información); y

- capa de acceso al medio, que indica cómo debe llevarse a cabo la transferencia de datos entre los dispositivos que se estén comunicando y es una interfaz entre *hosts* y enlaces de trasmisión.

Cuando uno o varios equipos se conectan a una red, las reglas y procedimientos técnicos que gobiernan su interacción y comunicación se llaman protocolos, de ellos existe una gran variedad, cada uno tiene un propósito que depende del tipo de servicio. En la Tabla 2.5 se describen cuatro protocolos que son de utilidad para el desarrollo del contenido del libro.

Tabla 2.5. Protocolos seleccionados [14]

Protocolo	Descripción
TCP	Orientado a la conexión, es capaz de crear una conexión virtual entre dos procesos para enviar datos. Es muy confiable ya que se asegura que los paquetes llegan de forma correcta a su receptor, para lo que cuenta con un control de flujo y error a nivel de transporte. Un ejemplo de este protocolo es la descarga de un archivo desde el navegador.
UDP	No está orientado a la conexión, es decir, no se asegura de que el receptor reciba de forma correcta los paquetes, sino que simplemente los envía "a la deriva", pues no cuenta con un control flujo y error a nivel de transporte. Por lo tanto, no es fiable. Un punto positivo de él es su rapidez, mucho mayor que TCP. Un ejemplo de su uso es la radio en Internet, que se transfiere vía UDP streaming.
DNS	Permite a los usuarios de red utilizar nombres jerárquicos sencillos para comunicarse con otros equipos, en lugar de memorizar sus direcciones IP; puede, en consecuencia, traducir un nombre a una dirección IP y una dirección IP a un nombre.
Stratum	Permite a los mineros recibir carga de trabajo de manera confiable y eficiente de los servidores; está construido sobre TCP/IP y trabaja con el formato JSON-RPC (*Remote Procedure Call Protocol Encoded in JSON*), lo que le permite hacer múltiples llamadas al servidor, sin esperar una respuesta sincrónica [15]. Se utiliza en la comunicación entre dispositivos mineros, servidores y servicios de piscinas mineras; no produce demasiado tráfico de red y permite que los resultados de hash se trasmitan con mayor frecuencia, lo que facilita obtener mediciones de hash rate más precisas [14].

Un servicio de conexión puede especificarse como un conjunto de primitivas que indican cuándo puede desarrollar acciones o informar alguna eventualidad. Las primitivas son llamadas al sistema que causan un salto al modo kernel, al mismo tiempo que retorna el control de la máquina al sistema operativo para enviar los paquetes necesarios.

Las primitivas pueden variar dependiendo del tipo de servicio, sea orientado o no a la conexión [16]. En la Tabla 2.6 se observan algunas de las primitivas más frecuentes.

Tabla 2.6. Primitivas de uso frecuente

Primitiva	Uso
LISTEN	Inicia el bloqueo de una conexión entrante, hasta que un proceso requiera conectarse.
CONNECT	Intenta establecer alguna conexión entre nodos.
ACCEPT	Acepta una conexión entrante.
SEND	Envía información.
RECEIVE	Bloquea en espera de información entrante.
DISCONNECT	Finaliza la conexión.

Análisis de tráfico de red

El análisis de tráfico de red es una poderosa herramienta que permite visualizar las condiciones en las que se encuentra la red en un determinado periodo, para garantizar su seguridad, rendimiento e integridad; permite evidenciar el tráfico producido por los diferentes nodos de procesamiento que se encuentran conectados a la red, a través de cualquier dispositivo que permita la transmisión y recepción de datos, por ejemplo, una tarjeta de red.

Este análisis permite visualizar, no solo los paquetes que se transmiten, sino también los protocolos de comunicación que son utilizados, tales como: TCP, UDP, HTTP; las direcciones IP de los emisores y receptores; y el tamaño del paquete, entre otras. Generalmente, para realizar este tipo de análisis, se hace uso de herramientas de *sniffing*, las cuales permiten interceptar el flujo de paquetes que están atravesando la red. Mediante el tráfico de red es posible evidenciar si una red presenta tráfico legítimo o malicioso, ya que los diferentes tipos de paquetes pueden dar una simple evidencia forense de comunicaciones producidas por *malware* o aplicaciones normales. "Algunas de las anomalías son indicios de cuellos de botella en el rendimiento, los cuales pueden ser causados por muchas razones, tales como *flashcrowds*, DDoS o fallos de componentes de la red" [17]. A partir de las capturas de la comunicación de una red de computadoras es posible analizar su información a nivel de detalle y como un resumen de la comunicación entre dos puntos.

El análisis de paquetes permite analizar una muestra de paquetes y con ello realizar capturas completas del tráfico de la red. Provee suficientes detalles para analizar y plantear soluciones de problemas en la red. Algunas de sus desventajas son cuando se presenta tráfico cifrado y el consumo de recursos que conlleva realizar un análisis de este tipo [18]. Un ejemplo de este tipo de análisis es mediante el uso de herramientas de *sniffing*, como Wireshark (Figura 2.5).

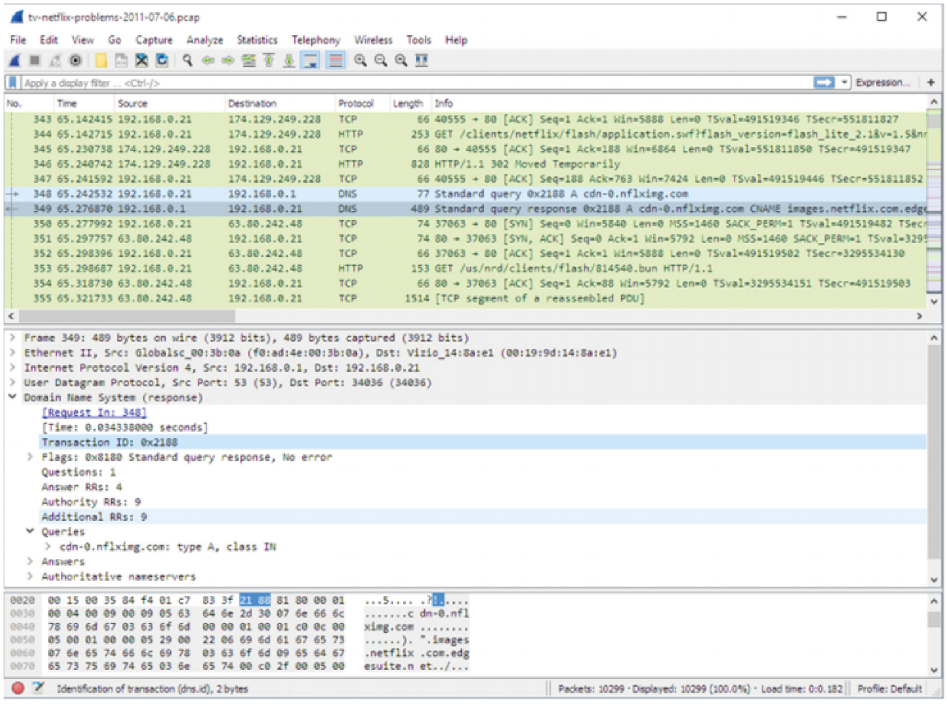

Figura 2.5. Captura de tráfico de red mediante Wireshark

Netfow es una función de enrutadores CISCO [19] que tiene como propósito analizar el tráfico de red que agrupa los paquetes que tienen en común valores como: la IP de origen, la IP de destino, el puerto, el protocolo y el tipo de servicio en un solo flujo. Posteriormente se cuentan los paquetes y *bytes*, con el fin de tomar registros que resuman la inmensa cantidad de información. Cuando se produce un cambio en alguna de las variables del NetFlow, se define un nuevo flujo [20].

2.3. Inteligencia artificial

La inteligencia artificial es el área de las ciencias de la computación que se encarga de la simulación del comportamiento inteligente humano en las máquinas, entendiendo inteligencia humana como los procesos de percepción sensorial (visión, audición etc.) y sus consiguientes procesos de reconocimiento de patrones [21].

Para lograr los procesos mencionados es necesaria la construcción de sistemas inteligentes, es decir de sistemas que presenten las características de la conducta inteligente, esto es: el aprendizaje, el razonamiento y la adaptación. Además, se deben tener en cuenta otros dos elementos: la ontología general del sistema y el medio ambiente en el que operará [22].

"La ontología general se refiere (...) a todo lo que el sistema es capaz de reconocer e interpretar del medio ambiente y que le es útil para tomar una decisión" [22], mientras que el medio ambiente es el sistema en donde se encuentra inmerso el sistema inteligente. Es decir que el sistema inteligente es un subsistema del medio ambiente.

Cuando se habla de aprendizaje se hace referencia a los cambios en la forma de reaccionar de un ente frente a una situación experimentada anteriormente. El razonamiento es la operación en la que, partiendo de uno o varios juicios (premisas), se infiere un nuevo juicio que se desprende lógicamente de las premisas; por otro lado, la adaptación es la capacidad de responder a un determinado objeto o evento a través de una conexión estímulo-reacción.

2.4. Ciencia de datos

La ciencia de datos busca identificar y comprender patrones en un conjunto de datos, para construir modelos que representen el contexto de la información; su propósito es recopilar datos de diversas fuentes de información por medio de técnicas y herramientas que incluyen métodos de estadística, *machine learning*, minera de datos y visualización.

Un modelo define la relación entre atributos de datos y una función matemática o estadística. Existen modelos descriptivos, cuyo objetivo es proveer más información sobre el contexto de los datos, y modelos predictivos, encargados de estimar un objetivo a partir de una serie de variables y sus valores.

Entre las metodologías reconocidas para la aplicación de la ciencia de datos, se encuentran: *Analytics Solutions Unified Method* (ASUM-DM), *Knowledge Discovery in Databases* (KDD) y *Cross-Industry Standard Processfor Data Mining* (CRISP-DM).

La guía de referencia que utilizan los proyectos que se presentan en este libro es CRISP-DM, ella fue seleccionada porque sus actividades cíclicas representan en gran medida a la metodología científica. De acuerdo con Chapman et al. [23] CRISP-DM incluye seis grupos de actividades: entendimiento del negocio, entendimiento de los datos, preparación de los datos, modelado, evaluación y despliegue (Figura 2.6), las que se desarrollan como se describe a continuación.

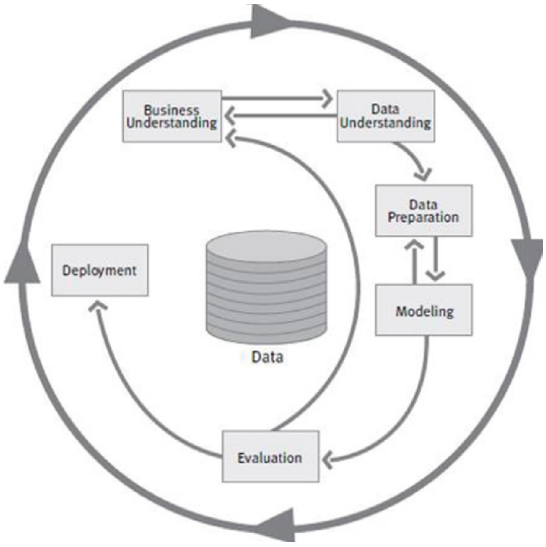

Figura 2.6. CRISP-DM [23, p. 13]

El primer grupo, entendimiento del negocio, corresponde a las actividades necesarias para conocer a fondo el contexto del fenómeno de estudio. Durante esta etapa, se busca el desarrollo de preguntas e hipótesis y el planteamiento de problemas que puedan ser resueltos con información e involucra actividades que van, desde la ejecución de reuniones con distintas áreas de negocio, hasta la investigación del área de estudio.

Una vez definidos los objetivos de la ciencia de datos, se continúa con las actividades de acceso, recolección y estudio de los datos (entendimiento de los

datos), los cuales pueden ser estructurados, semiestructurados o no estructurados. Parte de los objetivos de esta etapa es alinear la información con los resultados conseguidos en el entendimiento del negocio, lo que incluye actividades de exploración de datos (*Exploratory Data Analysis*, EDA).

Luego, en la etapa de preparación de los datos, se realizan las actividades necesarias para la transformación y preparación de la información, es decir, se corrigen los problemas y se adaptan los datos (por ejemplo, el texto en vectores numéricos) para que los algoritmos de *machine learning* se puedan entrenar.

El modelado, por su parte, incluye la selección, el desarrollo y la parametrización de los algoritmos de *machine learning*. Además, en él se plantean los escenarios (experimentos) y los datos que servirán tanto para el entrenamiento como para la evaluación de cada modelo.

Para la evaluación, dependiendo de la cantidad de información y con el fin de valorar el desempeño de cada modelo, los datos pueden ser agrupados en tres conjuntos: entrenamiento, validación y testeo). El de validación tiene como finalidad que el desarrollador pueda ajustar cada experimento, mientras que el de testeo permite un juicio adicional de cada modelo a través de las métricas de evaluación (*underfitting* y *overfitting*). Si a juicio del proyecto y del desarrollador, los datos no dan abasto para crear los tres conjuntos, solo se utilizan dos de ellos: entrenamiento y validación.

Finalmente, el despliegue está formado por las actividades necesarias para incluir los modelos en la solución informática que resuelva las necesidades del negocio. Los modelos son software que puede ser incorporado de varias maneras dependiendo de los requerimientos del sistema, a través de servicios web, tableros de datos, etc.

2.5 Machine learning

El aprendizaje de máquina es un campo de la inteligencia artificial que ha logrado conquistar a la industria, pues juega un papel muy importante en la esencia de los productos tecnológicos que hoy en día se conocen, por ejemplo: el sistema de recomendación de Netflix o el sistema de reconocimiento de voz de los teléfonos inteligentes. Se define como una mezcla de ciencia y arte que se encarga de programar los computadores de tal manera que puedan aprender directamente de un conjunto de datos.

En 1959, Arthur Samuel acuñó el término y lo definió como "el campo de estudio que le brinda a los computadores la habilidad de aprender sin necesidad de estar explícitamente programados". En 1997, Tom Mitchell dio una definición más formal al indicar "se dice que un programa de computador aprende de una experiencia *E* con respecto a alguna tarea *T* y alguna medida de rendimiento *P*, si su rendimiento en *T*, medido por *P*, mejora con la experiencia *E*" [24].

El corazón de esta ciencia, según las definiciones anteriores, es aprender de la experiencia (muy similar a la forma en que aprenden los seres humanos). Esta experiencia está expresada en forma de datos históricos almacenados en un computador, de los que luego se extraen patrones que permiten generalizar el conocimiento. Dentro del mundo de reconocimiento de patrones, hay dos grandes tipos: el aprendizaje supervisado y el no supervisado [25]. Asimismo, existe otro enfoque conocido como aprendizaje por refuerzo.

2.5.1 Aprendizaje supervisado

El aprendizaje supervisado parte de un conjunto de objetos (datos históricos) con cardinalidad N, que están descritos por un vector D-dimensional (Ecuación 2.1) de características (variables predictivas) y una variable de salida y que representa un elemento del conjunto de clases C (que puede ser finito), donde sus elementos son la solución esperada de un problema específico.

$$\vec{X} = \{\alpha_1, \alpha_2, \dots, \alpha_D\} \qquad (2.1)$$

Este conjunto C puede tener valores categóricos o numéricos (números reales). Al conjunto de objetos (datos históricos) descritos por $\vec{X}$, junto con la variable $\mathbf{y}$, se expresan matemáticamente como indica la Ecuación 2.2.

$$D = \{(\vec{X_1}, Y_1), (\vec{X_2}, Y_2), \dots (\vec{X_n}, Y_n)\} \qquad (2.2)$$

A partir del conjunto D, el aprendizaje supervisado construye un modelo o regla general, cuyo propósito es clasificar entradas $\vec{X_i}$ nuevas de las cuales no se conoce su clase Y_i, es decir la solución al problema [25],]26].

Algunos de los más importantes algoritmos de *machine learning* para el aprendizaje supervisado son: *k-Nearest Neighbors* (k-NN), regresión lineal, regresión logística, *Support Vector Machines* (SVM), árboles de decisión, *random forests* y redes neuronales (*neural networks*).

El aprendizaje supervisado consiste en aprender con base en etiquetas o datos bien definidos, es decir con base en una serie de entradas etiquetadas con una clase o valor, llamadas predictores, con los que el algoritmo aprende. Su objetivo es predecir una clase o valor a partir de dichos predictores.

Clasificación

Cuando la variable y es categórica, se trata de un problema de clasificación o de reconocimiento de patrones; entonces:

$$y_i \in \{c_1, c_2, \dots, c_n\} \; \forall_i = 1, \dots, N, \qquad (2.3)$$

para un problema con M clases distintas, donde estas clases son la solución esperada al problema. Si $|c|=2$, se trata de un problema de clasificación binaria (donde la mayoría de las veces se asume que:

$$y_i \in \{0, 1\} \; \forall_i = 1, \dots, N; \qquad (2.4)$$

en cambio, si $|c|>2$, se trata de un problema de clasificación multiclase.

Regresión

Cuando la variable y es numérica, es muy similar al caso de clasificación, sin embargo, en el caso de la regresión, en vez de tener un conjunto de clases C, se considera todo el conjunto de los números reales, es decir $y_i \in R$ [27].

El problema del aprendizaje

Para comprender a un nivel introductorio la construcción del modelo, se procede a formalizar el problema de aprendizaje para el caso de los problemas de clasificación y a presentar un ejemplo sobre la aplicación del aprendizaje supervisado para realizar la detección de *apps* maliciosas en Android [28]. El problema de aprendizaje consta de los elementos descritos en la Tabla 2.7.

La función objetivo ($f: x \longrightarrow y$) es desconocida y tiene: como dominio entrada al conjunto x y como codominio al conjunto de salida y. Este último puede tomar dos valores, *benign* y *malicious*, para las *apps* legítimas y maliciosas, respectivamente. En todos los problemas de ML, la función f es desconocida, pues si se conociera simplemente se implementaría, y ya no tendría sentido el aprendizaje.

Tabla 2.7. Elementos del problema de aprendizaje

Elemento	Expresión	Descripción
Entrada	x	Conjunto de todos los vectores Xx, en este caso características de tráfico de red de Android Apps.
Salida	y	Conjunto de todos los y, es decir la solución esperada al problema, en este caso: benign, malicious.
Función objetivo	$f: x \longrightarrow y$	Fórmula ideal para clasificar Android Apps.
Datos	$D=\{(xX1, y1), (xX2, y2), ..., (xXN, yN)\}$	Datos históricos de Android Apps.
Función hipótesis	$g: x \longrightarrow y$	Muy similar a f.
Algoritmo de aprendizaje	A	
Conjunto de hipótesis	H	Fórmulas candidatas para g

En cambio, sí es posible conocer la función hipótesis ($g: x \longrightarrow y$),

$$y_i = g(x_i) \ \forall_i = 1, \ldots, N \qquad (2.5)$$

Esta función se escoge a través de un algoritmo de aprendizaje *A*, el cual, usando el conjunto de entrenamiento *D*, escoge g entre un conjunto de fórmulas candidatas *H*, el cual podría ser, por ejemplo, el conjunto de todas las fórmulas lineales. El algoritmo de aprendizaje *A* escogerá la que mejor se ajuste al conjunto de entrenamiento *D*). Finalmente, como el conjunto de entrenamiento *D* es un reflejo de la función ideal *f* (debido a que estos datos históricos son una muestra de la población total de Android *apps*), entre más significativa sea la muestra en *D*, $g \approx f$. Esta aproximación es el fin del aprendizaje [29]. La Figura 2.7 corresponde al esquema general del problema de aprendizaje.

Protocolos de evaluación

La función de los protocolos de evaluación es validar la capacidad de generalización de los modelos con respecto a un conjunto de datos nuevos. Así se evita el sesgo causado al evaluar con el mismo conjunto de datos históricos *D*.

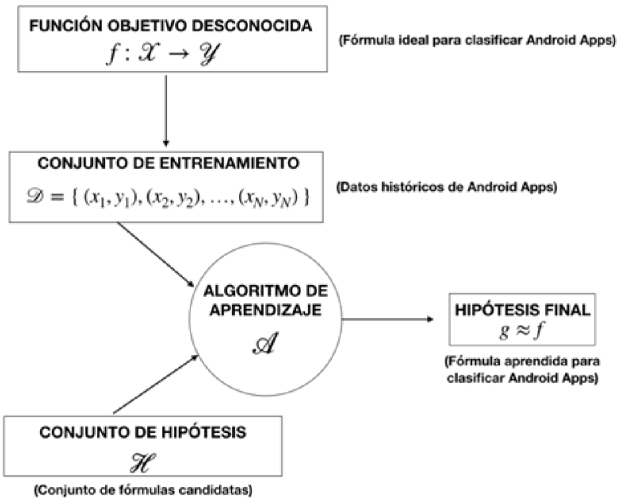

Figura 2.7. Esquema básico del problema de aprendizaje [29]

Los protocolos de evaluación más destacados son *Hold-out* y *K-foldcross-validation* [30]. *Hold-out* parte el conjunto de datos en dos: un subconjunto de entrenamiento, con el cual el algoritmo de clasificación aprende; y otro de validación, que está excluido del conjunto de entrenamiento.

En ambos casos, los datos se seleccionan aleatoriamente; entre más datos, mejor aprendizaje y mejor evaluación. *K-foldcross-validation*, por su parte, divide el conjunto de datos en K conjuntos disyuntos del mismo tamaño (el K permite hacer un balance entre sesgo y varianza); de ellos, $K-1$ se utilizan para entrenamiento y 1 para validación. El proceso se repite K veces y finalmente se agregan las métricas de evaluación. Se estima que los mejores resultados se obtienen con un valor de K entre 5 y 10 [30].

Overfitting y underfitting

Al momento de entrenar un modelo para que tenga una buena capacidad predictiva es necesario lograr un balance entre el sesgo (*bias*) y la varianza, pues un desbalance entre ellos puede producir un *underfitting* u *overfitting*.

Matemáticamente la relación entre el sesgo (*bias*) y la varianza, puede expresarse como aparece en la Ecuación 2.6, en donde b es el sesgo (*bias*), c la complejidad del modelo y v la varianza [31].

$$\frac{db}{dc} = \frac{dv}{dc} \qquad (2.6)$$

El *overfitting* ocurre cuando el modelo se ajusta excesivamente respecto de los datos del conjunto de entrenamiento; este ajuste excesivo se enfoca en datos que son más bien "ruido", memorizándolos y haciendo que no se generalice el conocimiento respecto de un conjunto de datos nuevos. En este caso, el *bias* es mucho menor que la varianza y la complejidad del modelo (es decir, polinomios de un grado más alto) crece junto con la varianza [26], [32]. En el *underfitting*, en cambio, el modelo tiene una *accuracy* muy bajo, incluso respecto del conjunto de entrenamiento (el *bias* es mucho mayor que la varianza). Cuando esto ocurre, por lo general se opta por buscar otro modelo que se ajuste mejor a los datos [32].

Métricas de evaluación

Una vez construido el modelo de *machine learning*, es importante conocer los criterios de evaluación, para así determinar qué tan bien realiza el trabajo de clasificación. La valoración generalmente se realiza con la tasa de error (T_E) del modelo, la cual se calcula como se indica en la Ecuación 2.7.

$$T_E = \frac{Número\ de\ errores}{Cantidad\ de\ casos} \qquad (2.7)$$

Existe una matriz, llamada matriz de confusión (ver Figura 2.8) que ilustra mediante una tabla de contingencia la distribución de los errores cometidos por el modelo respecto al distinto número de clases *C*. En la matriz se intercepta la variable predicha por el modelo con la variable que guarda los valores verdaderos de la clasificación; sus resultados pueden ser Verdadero Positivo (VP), Falso Positivo (FP), Falso Negativo (FN) y Verdadero Negativo (VN).

Los elementos sombreados que se encuentran en la diagonal principal de la matriz de confusión (VP y VN) muestran la cantidad de instancias correctamente clasificadas, mientras que las demás casillas muestran los diferentes tipos de error de clasificación: tipo I, FP; y tipo II, FN. De los elementos de la matriz de confusión también se pueden extraer otro tipo de indicadores útiles para evaluar el modelo, entre ellos los que se describen en la Tabla 2.8.

		Clase predicha		
		Positivo	Negativo	Total
Clase verdadera	Positivo	VP	FP	VP +FN
	Negativo	FN	VN	FN + VN
	Total	VP+FN	FP + VN	N

Figura 2.8. Matriz de confusión

Tabla 2.8. Medidas de desempeño para problemas de dos clases

Medida	Cálculo		Observación
Tasa de corrección (accuracy)	$\frac{VP+VN}{VP+VN+FP+FN}$	(2.8)	
Probabilidad de error	$\frac{FP+FN}{VP+VN+FP+FN}$	(2.9)	
Precisión	$\frac{VP}{VP+FP}$	(2.10)	Probabilidad de que dado que la predicción es "positivo", sea un verdadero positivo.
Recall (TPR o sensibilidad)	$\frac{VP}{VP+FN}$	(2.11)	Probabilidad de que dado que es un VP, la predicción sea "positivo".
Especificidad (TNR)	$\frac{VN}{VN+FP}$	(2.12)	
Coeficiente de concordancia Kappa (K)	$\frac{O.A + E.A}{1 - E.A}$	(2.13)	Probabilidad de que dado que es un VN, la predicción sea "negativo".
F1-score	$2\,x\,\frac{precision\ x\ recall}{precision + recall}$	(2.14)	Útil para sustrae el efecto de concordancia por suerte (E.A) del valor del accuracy (O.A) en datasets desbalanceados.

2.5.2 Aprendizaje no supervisado

El aprendizaje no supervisado no cuenta con una variable dependiente u objetivo *Y*. En este enfoque no se busca una predicción sino una estructura o

de un nuevo punto de vista en los datos. Usualmente se realiza durante la parte exploratoria de los datos. Algunas de las tareas que se pueden realizar con el aprendizaje no supervisado son: segmentación (*clustering*), reglas de asociación, reducción de dimensionalidad y detección de anomalías.

2.5.3 Reinforcement learning

De acuerdo con Ravichandiran, este "es un tipo de ML auto-evolutivo que nos lleva cada vez más cerca de lograr la verdadera inteligencia artificial"[33]. En esta rama de *machine learning*, el aprendizaje ocurre a través de la interacción de un sistema o un agente con su entorno y está orientado hacia una meta. En él, cada acción realizada por el sistema o agente es recompensado por su entorno con un reconocimiento (*reward*), el cual puede ser: positivo, si la acción acerca a meta; o negativo, si la aleja.

Reinforcement Learning (RL) es completamente diferente a otros paradigmas de *machine learning*. Su diferencia más notoria radica en que no necesita datos históricos (*data sets*) para extrapolar y generalizar el conocimiento, como es el caso del aprendizaje supervisado, sino que aprende de su interacción directa con el entorno [33]. Su aplicación requiere de al menos cuatro elementos:

- un agente, esto es un programa o programas que toman decisiones inteligentes (son aprendices);
- una política π, que define el comportamiento del agente en un entorno dado;
- una función valor, expresada como *V(s)* que es igual a la cantidad total esperada de *reward* recibida por el agente desde un estado inicial y denota qué tan conveniente es para él permanecer en un estado particular; y
- un modelo, es decir, la forma del agente para representar el entorno.

El aprendizaje puede ser de dos maneras: basado en el modelo, donde el agente utiliza una información previamente aprendida para llevar a cabo una tarea; o libre del modelo, donde el agente adopta una estrategia de "ensayo-error" para llevar a cabo la acción correcta.

Procesos de decisión de Markov

Los Procesos de Decisión de Markov (PDM), propuestos por Bellman, proveen un marco matemático para modelar y solucionar el problema de *reinforcement learning* [33].

Los PDM contienen cinco elementos:

- un conjunto de estados (S), en los cuales el agente puede permanecer;
  un conjunto de acciones (A) que puede ser ejecutado por el agente para ir de un estado a otro;
- una probabilidad de transición ($P^{a}_{s,s'}$), que es la probabilidad de ir de
- un estado (s) a otro estado (s'), ejecutando la acción (a);
- una probabilidad de *reward* ($R^{a}_{s,s'}$), que es la probabilidad de que el agente obtenga un premio por moverse de un estado (s) a otro estado (s'), ejecutando la acción (a); y
- un factor de descuento (γ), que controla la importancia inmediata y futura de los *reward*.

Q-Learning

El algoritmo de Q-Learning le da importancia al valor del estado-acción, en otras palabras, al efecto de ejecutar una acción A en un estado S. El valor de Q se guarda en una tabla llamada Q-table, donde sus valores son calculados de acuerdo con la Ecuación 2.15, donde: r es el *reward*; γ el factor de descuento; y α la razón de aprendizaje que sirve para la convergencia [22].

$$Q(s,a)=Q(s,a)+\alpha\,(r+\gamma maxQ(s',a')-Q(s,a)) \qquad (2.15)$$

Se trata de expresar que por cada acción tomada, la Q-table se actualiza para incluir un *reward* positivo o negativo. Es así como a partir del "ensayo-error", la Q-table va llenando sus valores.

Dado un estado en el cual se encuentre el agente, siempre se busca realizar una acción tal que desde ese estado sea la máxima o la que mejor *reward* obtenga.

Este proceso implica los siguientes pasos:

- se inicializa la Q-table con valores arbitrarios (generalmente "0");
- se toma una acción desde un estado usando una política epsilon-greedy (ϵ>0) y se mueve al estado nuevo;
- se actualizan los datos de la Q-table con la Ecuación 2.15; y
- se repiten los dos pasos anteriores hasta llegar a un estado terminal.

2.5.4 Redes neuronales artificiales

Como su nombre lo indica, las redes neuronales artificiales son en términos generales, redes computacionales que intentan simular el proceso de decisión que hacen las neuronas del sistema nervioso central biológico [34]. Se tiene entonces que, así como en el sistema nervioso central biológico existe una unidad fundamental —la neurona—, en una red neuronal artificial se tiene un análogo, el perceptrón.

Un perceptrón tiene varias entradas y las combina normalmente con una suma básica; la suma de las entradas se modifica por una función de transferencia; el valor de la salida de esta función de transferencia se pasa directamente a la salida del perceptrón [35], como se aprecia en la Figura 2.9.

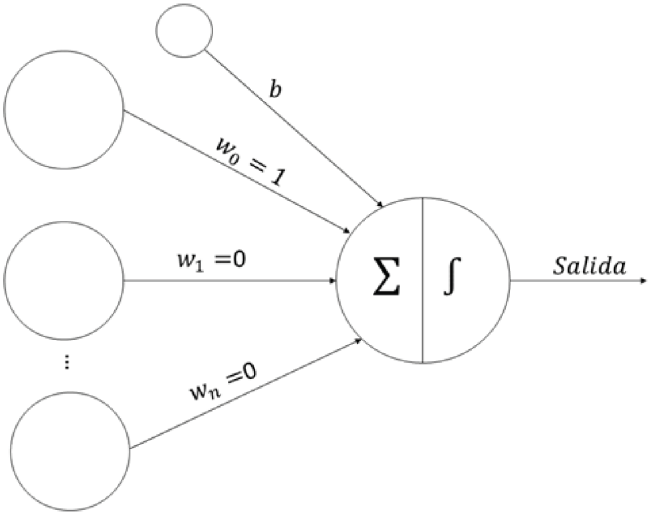

Figura 2.9. Salida del perceptrón

La salida del perceptrón se puede conectar a las entradas de otras neuronas artificiales o perceptrones, por lo tanto, una red neuronal consiste en un conjunto de perceptrones conectados de una forma concreta.

Generalmente, los perceptrones están organizados en grupos llamados niveles o capas. Una red normalmente consiste en una secuencia de capas con conexiones entre capas adyacentes consecutivas. Existen tres tipos de capas en

una red neuronal: una capa de entrada, en donde se le suministran los datos a la red; una capa de salida, en donde se mantienen los resultados de la red; y entre estas capas, un conjunto de capas denominadas capas ocultas [35].

2.5.5. Deep learning

El aprendizaje profundo (*Deep Learning*, DL) es un conjunto de algoritmos de *machine learning* que ha logrado un gran poder y flexibilidad al aprender a representar el mundo como una jerarquía anidada de conceptos; con cada concepto definido en relación con conceptos más simples y veinte representaciones más abstractas, calculadas en términos de conceptos menos abstractos [36]. La arquitectura está basada en redes neuronales con una mayor cantidad de capas ocultas que las redes neuronales convencionales. Este tipo de topología le permite obtener patrones o características simples a partir de entradas complejas. Adicionalmente, permite que cada capa se enfoque en resolver un problema específico. Existen varias arquitecturas de redes neuronales, entre ellas las redes neuronales profundas, las redes neuronales convolucionales y las redes neuronales recurrentes.

Las redes convolucionales (*Convolutional Neural Networks*, CNN) son un tipo especializado de red neuronal que procesa datos que tienen una topología similar a una cuadrícula. Los ejemplos incluyen datos de series temporales, que pueden considerarse como una cuadrícula 1D, que toma muestras a intervalos de tiempo regulares, y datos de imágenes, que pueden considerarse como una cuadrícula de píxeles 2D. Su nombre indica que la red emplea una operación matemática llamada convolución, un tipo especializado de operación lineal. Las redes convolucionales son simplemente redes neuronales que utilizan la convolución en lugar de la multiplicación general de la matriz en al menos una de sus capas [36].

Las redes neuronales recurrentes (*Recurrent Neural Network*, RNN) son un tipo de red neuronal que procesa datos secuenciales; al igual que una red convolucional, es una red neuronal especializada para procesar una cuadrícula de valores, como una imagen. Tal como las redes convolucionales, puede escalar fácilmente a imágenes con gran ancho y alto. Algunas redes convolucionales pueden procesar imágenes de tamaño variable, mientras que las redes recurrentes pueden escalar a secuencias mucho más largas de lo que sería práctico para redes sin especialización, basada en secuencias. La mayoría de las redes recurrentes también pueden procesar secuencias de longitud variable.

De las anteriores se desprenden otras arquitecturas que buscan resolver con mayor eficacia los problemas, a partir de la abstracción del funcionamiento y el propósito de sus antecesoras, entre ellas Xception (Figura 2.10), una arquitectura que parte de Inception y cuenta con 36 capas convolucionales estructuradas en catorce módulos que se conectan entre sí por medio de conexiones residuales, excepto el primer y el último módulo [37]. Lo anterior hace que esta arquitectura sea fácil de implementar y modificar. Es de código abierto y está disponible en el módulo de aplicaciones de Keras.

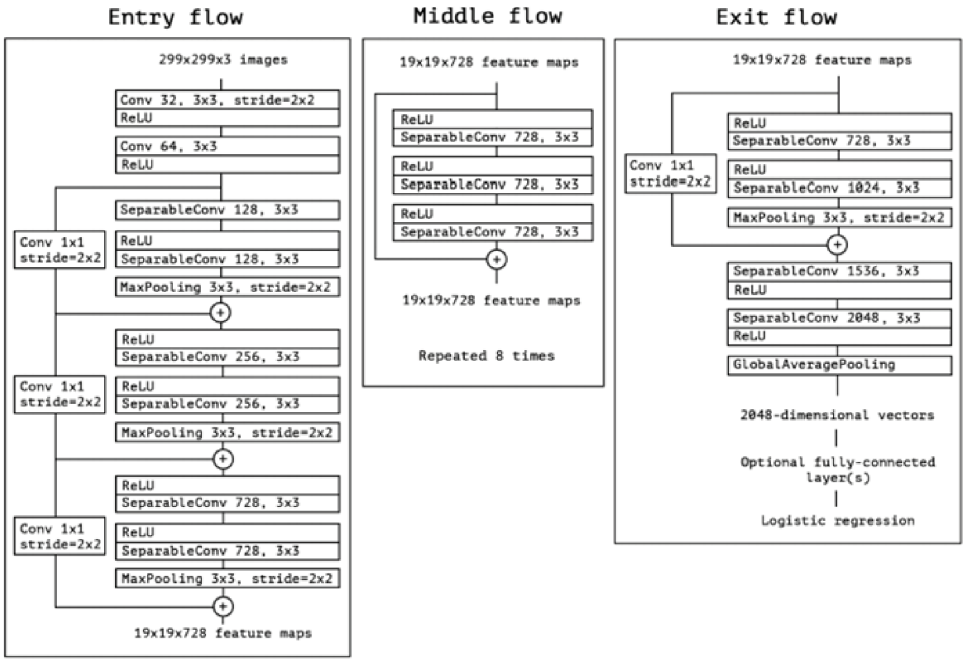

Figura 2.10. Arquitectura Xception [37]

2.5.6. Deep Reinforcement Learning [DRL]

El DRL es una combinación de *Deep Learning* y *Reinforcement Learning*. El RL es muy útil para la resolución de tareas en las que se requiere ejecutar acciones, es decir, controlar el comportamiento de un agente en un modelo de entorno, mientras que el DL es ideal para el tratamiento y procesamiento

de datos complejos, por lo que resulta una buena elección utilizar el RL como un *framework* que implementa algoritmos de DL para manejar los datos de una forma muy eficiente [38].

Un modelo generativo describe cómo un conjunto de datos se genera en términos de un modelo probabilístico, de forma que por muestreo de este modelo es posible generar nuevos datos [39]. La Figura 2.11 corresponde a la estructura de un sistema de DRL, específicamente a la interacción del agente con el modelo de entorno (modelos generativos)

Figura 2.11. Estructura de un sistema de DRL

Un modelo generativo requiere un conjunto de datos (datos de entrenamiento) con muchos ejemplos de la entidad que se quiere generar; cada dato es una observación; y cada observación tiene características que van a permitir crear o generar nuevos datos. El objetivo de un modelo de este tipo es imitar una distribución de datos de la forma más cercana posible, de manera que esos datos generados parezcan haber sido incluidos en el conjunto de entrenamiento original.

2.5.7 Generative adversarial network

Una red generativa antagónica (*Generative Adversarial Network*, GAN) es una estructura donde interactúan dos modelos DL para mutuamente evaluarse y mejorar; funciona asignando los roles de discriminador (evaluador) y generador (evaluado); y a través de las respuestas y de las interacciones permite construir el mecanismo de mejora [36].

Las GAN son comúnmente usadas para generar datos nuevos de una forma precisa, los que se encuentran en el llamado espacio latente, el espacio de

diferencias entre dos muestras conocidas de algún dato. Por ejemplo, en un escenario de dos rostros, el espacio latente es una cantidad inmensa de muestras que comparten características de los dos rostros originales. Inicialmente, en el proceso de entrenamiento de una GAN, los datos del modelo generador son aleatorios, pero a medida que este recibe el *feedback* del discriminador, los datos cada vez se parecen más a los datos reales. Si bien esto dificulta la tarea del discriminador, el continuo de interacciones le permite mejorar.

2.6 Ciberseguridad e inteligencia artificial

2.6.1 Deepfake

Acrónimo de *fake* (falsificación) y *deep learning*. Parte de una técnica que permite editar videos, imágenes o audios falsos de personas creadas por los algoritmos de inteligencia artificial e insumos digitales de personas reales. El espectrograma es la representación visual que resulta de calcular el espectro de frecuencias de tramas de una señal que varía con el tiempo.

El gráfico estándar que se utiliza para mostrar el espectrograma simboliza: en un eje, el tiempo; y en el otro, la frecuencia. Opcionalmente, en una tercera dimensión se indica la amplitud de una frecuencia particular en un momento particular, que está explicada por la intensidad o el color de cada punto de la imagen. Puede ser usado para generar el *deepfake* de audio, mediante métodos de generación de imágenes que luego se transforman en audio; el espectrograma también puede usarse con los métodos de detección de *deepfakes* de audio, encontrando una diferencia en los audios generados por un humano y por un algoritmo de inteligencia artificial.

2.6.2 Adversarial machine learning

El aprendizaje automático adversativo es una técnica útil para engañar a los modelos de *machine learning* a través de entradas ($\vec{x}$) maliciosas [33]. Su principal propósito es vulnerar los modelos y afectar su buen funcionamiento.

Si bien los modelos de *machine learning* fueron diseñados con la premisa de que los conjuntos de datos de entrenamiento y de evaluación pertenecen a la misma distribución estadística, existen adversarios inteligentes, autoadaptativos, que de cierta forma pueden romper esa premisa haciendo que se comprometa la seguridad del modelo [14].

2.6.3 Tipos de ataques en contra de machine learning

Existen tres grandes ramas que categorizan los diferentes ataques contra los modelos de ML [11]: ataques influyentes, violaciones de seguridad y especificidad, como se describe en la Tabla 2.9.

Tabla 9. Ramas de los tipos de ataque a los modelos de ML

Rama	Tipo	Detalle
Ataques influyentes	Ataques causativos	Influyen sobre el aprendizaje del modelo controlando el dataset de entrenamiento.
	Ataques exploratorios	Realizan un exploit para que el modelo haga una clasificación errónea.
Violación de seguridad	Ataques de integridad	Comprometen la fiabilidad del modelo a través de falsos negativos.
	Ataques de disponibilidad	Causan la denegación del servicio, usualmente a través de falsos positivos.
Especificidad	Ataques de disponibilidad	Causan la denegación del servicio, usualmente a través de falsos positivos.
	Ataques indiscriminados	Encierran una amplia cantidad de instancias (registros).

Los sistemas de ML no están exentos de los riesgos presentes en el mundo digital. Los atacantes se enfocan en: la extracción de datos, la copia del modelo, el engaño a través de entradas modificadas utilizando *adversarial attacks* o la corrupción del modelo. Esto último implica una penetración que causa el mal funcionamiento interno, de tal forma que de entradas "limpias" se generan salidas incorrectas [40]. Los tipos de ataque a los modelos de *machine learning* se pueden agrupar en las categorías presentes en la Tabla 2.10.

Tabla 2.10. Tipos de ataque a los modelos de ML [41]

Categoría	Propósito
Reconocimiento	Encontrar información acerca del modelo y su entorno, las tecnologías empleadas y las versiones.
Acceso inicial	Obtener un acceso no autorizado al uso del modelo.
Ejecución	Ejecutar los modelos, en ambientes no seguros, con el fin de comprometerlos.

Tabla 2.10. Tipos de ataque a los modelos de ML [41] (continuación)

Categoría	Propósito
Persistencia	Realizar la implementación de backdoors y el envenenamiento de datos.
Evasión del modelo	Causar fallas u obtener información que permita evadir los modelos clasificación (esta categoría incluye a los ataques adversarios, pieza fundamental del actual proyecto)
Filtración de información	Robar los datos de entrenamiento, incluso clonar el modelo.
De impacto	Alterar la configuración, denegar el servicio, en general, paralizar el correcto funcionamiento del modelo.

Por todo lo anterior, es importante robustecer el modelo para reducir las posibilidades de éxito de los ataques como efecto de la disminución de las vulnerabilidades.

Ese proceso de robustecimiento se puede alcanzar, en el caso del engaño del modelo, a través de *adversarial training*, entrenando otro modelo de ML para generar entradas modificadas específicamente para engañar el modelo [42] y con estos resultados, realizar el entrenamiento, mezclando los datos anteriores con los nuevos datos adversos, logrando así un modelo más robusto.

2.6.4. Secure learning

Uno de los riesgos en el uso de modelos basados en *machine learning* es que los atacantes pueden aprovecharse de la naturaleza auto-adaptativa de los modelos y, a través de un *exploit*, intentar que fallen, es decir que realicen mal una predicción o una clasificación.

Aamir y Zaidi [11] definen el término como la habilidad de un modelo de *machine learning* para tener un buen desempeño incluso en condiciones adversas, es decir cuando los cibercriminales o atacantes intentan realizar un *exploit* en contra del modelo. Para evaluar la seguridad de un sistema, se debe:

- determinar los tipos de ataque al sistema;
- evaluar la resiliencia del sistema con respecto a estos ataques; y
- buscar que el sistema se torne más robusto, es decir, que el modelo sea menos vulnerable ante estos ataques.

2.7. Android

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, Honey Comb, Ice Cream Sandwich, Jelly Bean, Kit Kat, 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 2021 es el más popular de los sistemas operativos para dispositivos móviles.

Al igual que el resto del sistema, el modelo de seguridad de Android también aprovecha las características de seguridad que ofrece el núcleo de Linux, un sistema operativo multiusuario, cuyo kernel puede aislar los recursos de los usuarios entre sí, del mismo modo que aísla los procesos.

En los dispositivos móviles la seguridad juega un papel importante, ya que las aplicaciones maliciosas pueden actuar de forma tal que lean la lista de contactos, averigüen la posición con el GPS y envíen toda esta información por Internet o por SMS.

La seguridad en Android se fundamenta en tres pilares:

- Android impide que las aplicaciones tengan acceso directo al hardware o interfieran con recursos de otras aplicaciones;
- toda aplicación ha de ser firmada con un certificado digital que identifique a su autor, la firma digital garantiza que el archivo de la aplicación no ha sido modificado (Android utiliza la firma del archivo APK para asegurar que las actualizaciones de una aplicación proceden del mismo autor —política de mismo origen—, y para establecer relaciones de confianza entre las aplicaciones;
- si se desea modificar una aplicación, esta tiene que ser firmada de nuevo, lo que solo puede hacer el propietario de la clave privada.

Un elemento de seguridad de Android [16] es el archivo de manifiesto (AndroidManifest.xml), que está incluido en el paquete de instalación de Android (archivo APK), junto con el *byte code Java* y otros recursos relacionados. Este archivo está escrito en XML y proporciona toda la información necesaria a la plataforma Android para la ejecución de la aplicación.

El archivo de manifiesto es crucial para el sistema, ya que en él se definen los permisos de cada aplicación. Estos permisos funcionan de dos formas: la primera es cómo la aplicación interactúa con el sistema mediante el acceso a la API del mismo; y la segunda, la forma cómo el sistema y otras aplicaciones interactúan con la aplicación dada.

A cada aplicación se le asigna un directorio de datos dedicado, donde solamente ella tiene permiso para leer y escribir. De este modo, las aplicaciones son aisladas a través de entornos de pruebas (*sandbox*), tanto a nivel de proceso (haciendo que cada una se ejecute en un proceso dedicado), como a nivel de archivo (mediante el directorio de datos privado).

Para distinguir las aplicaciones instaladas para cada usuario, Android asigna un nuevo *User ID* (UID) efectivo a cada aplicación cuando se instala para un usuario en particular. Este UID efectivo se basa en el ID de un usuario físico de destino y el UID de la aplicación en un sistema de un solo usuario (el ID de aplicación).

Debido a que las aplicaciones de Android son ejecutadas en entornos *sandbox*, solo pueden acceder a sus propios archivos y a cualquier recurso con acceso general en el dispositivo. Sin embargo, una aplicación tan limitada no sería muy interesante, y Android puede conceder derechos detallados de acceso adicional a las aplicaciones para permitir una funcionalidad más completa. Estos derechos de acceso se denominan permisos y pueden controlar el acceso a dispositivos de hardware, conectividad a Internet, datos o servicios del sistema operativo.

Vulnerabilidades en Android

Drake et al. [43] explican que los ataques contra el DNS (*Domain Name System*) son fáciles porque la baja latencia asociada con la adyacencia de red significa que los atacantes pueden responder más rápido que los *hosts* basados en Internet. Los ataques de espionaje contra DHCP (*Dynamic Host Configuration Protocol*) también son bastante efectivos para ganar control sobre un sistema de destino.

Por su parte, los ataques de hombre en el medio (*Man-in-the-Middle*, MitM) son muy poderosos. Los métodos para convertirse en un atacante en el medio incluyen comprometer enrutadores o servidores DNS, usar interceptaciones legales, manipular *hosts* mientras están conectados a la red y modificar las

tablas de enrutamiento global de Internet. Un método que en la práctica parece menos difícil que el resto es secuestrar el servicio DNS a través de los registradores; asimismo, otra manera relativamente fácil de efectuar un ataque MitM es específica para redes inalámbricas, tales como WiFi y celulares.

Los ataques exitosos contra los navegadores Web se pueden llevar a cabo de varias maneras. El método más común consiste en persuadir a un usuario para que visite una URL que está bajo el control del atacante; este método es probablemente el más popular debido a su versatilidad. Un atacante puede fácilmente entregar una URL por correo electrónico, medios sociales, mensajería instantánea u otros medios.

Otra forma es insertando el código de ataque en sitios comprometidos que las víctimas pretenden visitar. Las redes publicitarias representan una pieza interesante y potencialmente peligrosa del rompecabezas por varias razones. La funcionalidad que genera los anuncios publicitarios se basa generalmente en un motor de navegación integrado (un *WebView*). Además del riesgo de ejecución remota de código, los marcos publicitarios también presentan un riesgo significativo para la privacidad.

Las aplicaciones de terceros (tiendas no oficiales) representa otra importante fuente de ataque remoto. Tal vez el mejor ejemplo es una aplicación Android. Como es evidente por ahora, las aplicaciones Android contienen código que se ejecuta directamente en un dispositivo Android. Por lo tanto, la instalación de una aplicación equivale a conceder una ejecución de código arbitraria (aunque dentro del entorno de Android a nivel de usuario) al desarrollador de la aplicación.

Referencias

[1] *Kaspersky.* Disponible: htts://www.kaspersky.com/resource-center/definitions/what-is-cybersecurity

[2] *Cyber kill chain en sistemas de control industrial.* (2006). Disponible: https://www.incibe-cert.es/blog/cyber-kill-chain-sistemas-control-industrial

[3] P. Seshagiri, A. Vazhayil, y P. Sriram, "AMA: static code analysis of web page for the detection of malicious scripts," en *Procedia Computer Science*, 2016, vol. 93, pp. 768–773.

[4] C. Kruegel y G. Vigna, "Anomaly detection of web-based attacks," en *Proceedings of the 10th ACM conference on Computer and communication security - CCS '03*, 2003, p. 251.

[5] I. Corona y G. Giacinto, "Detection of Server-side Web Attacks," *J. Mach. Learn. Res.*, vol. 11, pp. 160–166, 2010.

[6] Avast. (2015). *What are exploits and how to protect yourself from them* [video]. Disponible: https://www.youtube.com/watch?v=a0EHm9dhnMc

[7] G. Weidman, *Penetration testing: a hands-on introduction to hacking*, No Starch Press, 2014.

[8] R. Moir. (2009). *Defining Malware: FAQ — Microsoft Docs*. Disponible: https://docs.microsoft.com/en-us/previous-versions/tn-archive/dd632948(v=technet.10)?redirectedfrom=MSDN

[9] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, y L. Cavallaro, "The evolution of Android malware and Android analysis techniques," *ACM Computing Surveys*, vol. 49, no. 4, pp. 1–41, 2017.

[10] Kaspersky Lab. *Types of known threats: general information.* Disponible: https://www.kaspersky.com/resource-center/threats/malware-classifications

[11] M. Aamir y S. Zaidi. "DoS attack detection with feature engineering and machine learning: the framework and performance evaluation," *Int. J. Inf. Secur.* 18, 761–785, 2019. https://doi.org/10.1007/s10207-019-00434-1.

[12] S. Poremba. (2017). *eSecurity planet*. Disponible: https://www.esecurityplanet.com/network-security/types-of-ddosattacks.html

[13] L. M. Arboleda, *Programación en red con JAVA*. Cali, Colombia: Universidad Icesi, 2004.

[14] *Stratum protocolo*. (2020). Disponible: https://braiins.com/stratum-v2

[15] R. Recabarren y B. Carbunar, (2017), "Hardening stratum: the bitcoin pool mining protocol," en P*roceedings on Privacy Enhancing Technologies*, vol. 3, pp. 57–74, 2017. doi: https://doi.org/10.1515/popets-2017-0028

[16] A. Tanenbaum, y D. Wetherall, *Redes de computadoras*, Pearson, 2012.

[17] J. Kim, A. Sim, B. Tierney, S. Suh, e I. Kim, "Multivariate network traffic analysis using clustered patterns," *Computing*, vol. 101, pp. 339-361, 2019. https://doi.org/10.1007/s00607-018-0619-4

[18] P. Minarik, *How to analyze and understand your network*. Burlington, MA: Flowmon, 2016.

[19] Cisco. (2012). *Introduction to Cisco IO Netflow: a technical overview*. Disponible: https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html

[20] NetFort Technologies Limited. (2014). *Flow analysis versus packet analysis: what should you choose?* Disponible: https://docplayer.net/8214688-Flow-analysis-versus-packet-analysis-what-should-you-choose.html

[21] R. Benítez, G. Escudero, S. Kanaan, y D. M. Rodó, *Inteligencia artificial avanzada*. Barcelona, España: UOC, 2014.

[22] J. J. Romero, C. Dafonte, A. Gòmez, y F. J. Penousal. (2007). *Inteligencia artificial y computación avanzada*. Disponible: http://fmachado.dei.uc.pt/wp-content/papercitedata/pdf/ms07.pdf#page=9

[23] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, y R. Wirth, *CRISP-DM 1.0 Step by step data minnnig guide*, SPSS, 2000. Disponible : https://www.kde.cs.uni-kassel.de/wp-content/uploads/lehre/ws2012-13/kdd/files/CRISPWP-0800.pdf

[24] A. Géron, *Hands-On machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. Sebastopol, CA: O'Reilly Media, 2017.

[25] B. Sierra, *Aprendizaje automático: conceptos básicos y avanzados*, Pearson, 2006.

[26] K. Murphy, *Machine learning: a probabilistic perspective*, Cambridge, MA: MIT, 2012.

[27] I. J. Goodfellow, Y. Bengio, y A. Courville, *Deep learning*, Cambridge, MA: MIT, 2016.

[28] C. Urcuqui, J. Delgado, A. Perez, A. Navarro, y J. Diaz, "Features to detect Android malware," *2018 IEEE Colombian Conference on Communications and Computing (COLCOM)*, 2018.

[29] Y. S. Abu-Mostafa, M. Magdon-Ismail, y L. Hsuan-Tien. *Learning from data: a short course*, AML Books, 2012.

[30] J. Díaz, *Aprendizaje automático: análisis de grandes volúmenes de datos* [notas de clase], Cali, Colombia: Universidad Icesi, 2017.

[31] S. Fortmann-Roe. (2012). *Understanding the bias-variance tradeoff*. Disponible: https://scott.fortmann-roe.com/docs/BiasVariance.html

[32] J. Grus, *Data science from scratch book*. O'Reilly Media, 2015.

[33] S. Ravichandiran, *Hands-On reinforcement learning with Python: master reinforcement and deep reinforcement learning using Open AI Gym and TensorFlow*, Packt, 2018.

[34] D. Groupe, *Principles of artificial neural networks*, World Scientific, 2013.

[35] X. Basogain, R*edes neuronales artificiales y sus aplicaciones*. Bilbao, España: Escuela Superior de Ingeniería de Bilbao.

[36] I. Goodfellow, Y. Bengio, y A. Courville. (2016). *Deep learning*. Disponible: http://www.deeplearningbook.org/front_matter.pdf

[37] F. Chollet, "Xception: deep learning with depth wise separable convolutions," en *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1251-1258), 2017. Disponible: http://openaccess.thecvf.com/content_cvpr_2017/papers/Chollet_Xception_Deep_Learning_CVPR_2017_paper.pdf

[38] B. Brown y A. Zai, *Deep reinforcement learning in action*, Shelter Island, NY: Manning Publications, 2020.

[39] D. Foster, *Generative deep learning: teaching machines to paint, write, compose, and play*, O'Reilly Media, 2019.

[40] S. Chaieb. *Machine learning systems: security*. Disponible: https://sahbichaieb.com/mlsystems-security/

[41] R. Siva y A, Johnson. (2021). "Cyberattacks against machine learning systems are more common than you think," *Microsoft Security Blog*, Disponible: https://www.microsoft.com/security/blog/2020/10/22/cyberattacks-against-machine-learning-systems-are-more-common-than-you-think/

[42] M. Barreno, A. D. Joseph, y J. D. Tygar, "Can machine learning be secure?," en *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, pp. 16–25, 2006.

[43] J. Drake, P. Fora, Z. Lanier, C. Mulliner, S. Ridley, y G. Wicherski, "Understanding Android's attack surface," *Android™ Hacker's handbook*, No Starch Press, 2015, pp. 129- 160.

Capítulo 3
Sistema para el estudio de ciberataques web

Brayan Henao, Juan Sebastián Prada, Christian Camilo Urcuqui, Andrés Navarro

3.1. Introducción

La seguridad informática se ocupa de diseñar las normas, procedimientos, métodos y técnicas destinadas a conseguir que un sistema de información sea seguro y confiable. La seguridad web, en especial, involucra proteger la información mediante la prevención, detección y respuestas a las violaciones. Su objetivo principal es mantener las tres características primordiales de la información [1]: confidencialidad, para asegurar el acceso a la información únicamente a las personas autorizadas; integridad, para la protección contra la modificación no autorizada o la destrucción de información; y disponibilidad, para garantizar el acceso oportuno y confiable a los datos y servicios de información a los usuarios autorizados [2].

Las amenazas web se han venido incrementando a través de los años por problemas tales como las vulnerabilidades de los navegadores, los defectos incluidos en los procesos de elaboración de las aplicaciones y defectos en algunos *plugin*. A la par con las amenazas, también se han diseñado métodos de detección para prevenirlas, las más comunes son: el análisis estático, el análisis dinámico y el análisis híbrido.

El análisis estático es un método focalizado a encontrar elementos maliciosos en el código fuente o en archivos de un elemento digital. Para la identificación de actividades maliciosas, este método puede ser incorporado a través de un enfoque de detección con firmas, donde cada elemento malicioso es plasmado en una base de datos ("lista de bloqueados") que servirá como medio de verificación.

El análisis dinámico adopta el enfoque opuesto, se ejecuta mientras un programa está en funcionamiento, supervisando la memoria del sistema, el comportamiento funcional, el tiempo de respuesta y el rendimiento general. El modelo de detección basado en anomalías incluido en él, consiste en generar un perfil de las peticiones que se hacen a un servidor, a partir del cual se identifican —y se marcan como posibles violaciones—, las peticiones que difieren de dicho perfil.

El análisis hibrido es una combinación de los métodos anteriores que busca obtener lo mejor de ellos. En consecuencia, primero, se realiza un análisis estático, para identificar estructuras maliciosas, y posteriormente se hace un análisis dinámico, para encontrar peticiones anómalas y posibles violaciones.

Los sistemas de detección de ciberataques web son vulnerables a nuevas metodologías de ataque. La existencia de diferentes tipos de ciberataques web, la utilización de sistemas de detección dinámicos, estáticos o híbridos —en los cuales las variables de detección no son cubiertas en su totalidad—, y la introducción de nuevas tecnologías de desarrollo de aplicaciones web han generado una incertidumbre en la seguridad de la información de las aplicaciones o páginas web.

Por lo anterior, se desarrolló la investigación que se reporta en este capítulo, la cual tuvo como objetivo general "diseñar un sistema para el estudio de ciberataques a aplicaciones web", y como objetivos específicos: generar un *dataset* que plasme al menos dos tipos de ciberataques asociados a tres riesgos informáticos del Top 10 de OWASP 2017; entrenar un método de detección que sea capaz de clasificar una petición HTTP a un servidor web como segura o insegura; diseñar un sistema para el estudio de ciberataques web que haga uso del método entrenado; y validar el método de detección.

3.2. Estado del arte

Las violaciones web pueden ser evaluadas desde de distintos enfoques, como es el caso de los análisis estático, dinámico e híbrido [3], sin embargo, para el proyecto que se reporta en este documento, se decidió usar *machine learning* como método de clasificación principal, debido a su particular enfoque para el manejo e interpretación de los datos y sus mejores resultados en comparación con los métodos tradicionales [4]. Cada una de las soluciones que se presentan en esta sección propone distintos enfoques para la detección de ciberataques

web —el enfoque de este proyecto—, y para la detección de páginas web maliciosas. En todos los casos, al final de la presentación se presenta la diferencia entre su enfoque y el proyecto que se reporta en este capítulo (denominado "el proyecto actual").

Detection of Server-side Web Attacks (DSSA)

Corona y Giacinto [5] proponen un sistema de detección de intrusos basado en anomalías, con reconocimiento de patrones para la detección de ciberataques hacia servicios web; utilizan modelos probabilísticos, en donde se le asigna una probabilidad (de normalidad) a cada petición realizada hacia el servidor. Usaron un *dataset* propio en el cual plasmaron cerca de 450.000 peticiones hacia un servidor web de su institución académica, durante una semana y crearon una serie de entradas (507) para probar el sistema, en el cual registraron violaciones diseñadas por ellos mismos; el *dataset* tuvo un proceso previo utilizando un algoritmo de detección de valores atípicos (*outliers*) para reducir su ruido. Además, utilizaron nueve variables asignadas a tres modelos estadísticos (A, B y C): el primero, basado en la secuencia de símbolos; el segundo, con la distribución estadística de un valor entero no negativo; y el tercero, con la distribución estadística de símbolos.

Los modelos obtuvieron una tasa de detección plena (232/232 violaciones) para el *dataset* generado y de 505/507 violaciones (99,6 %) para el *dataset* adicional que contenía las violaciones, y una tasa de falsas alarmas de 1.252 alertas/447.178 peticiones (0,28 %). Los investigadores concluyeron que el sistema diseñado es aceptable, pues presenta tasas altas de detección y muy bajas de falsas alarmas, y que los resultados podrían mejorar refinando el *dataset*, es decir, el algoritmo de detección de valores atípicos. Si bien este sistema da una solución aproximada al problema de detección de ciberataques, difiere de lo propuesto en el proyecto actual en la forma en que se aborda, pues lo hace desde una perspectiva probabilística.

Anomaly Detection of Web-based Attacks (ADWA)

Kruegel y Vigna [6] presentan un sistema de detección de violaciones web basado en anomalías el cual aprovecha la estructura de una petición HTTP que contiene parámetros; en la evaluación de este sistema se utilizaron tres *datasets*: el primero, con los registros de un servidor web Apache en producción de Google; y los dos restantes obtenidos de servidores ubicados en la Universidad

de California Santa Barbara (UCSB) y en la Universidad Técnica de Vienna (UTV). Para la experimentación de los obtenidos de sitios de seguridad populares, se utilizaron once *exploits* reales. Este sistema utiliza seis modelos para la detección denominados: de tamaño de atributos; de distribución de caracteres; de inferencia estructural; *token finder*; de presencia o ausencia de atributos; y de orden de atributos.

El sistema detecto la totalidad de las violaciones (11/11) y reportó un número bajo de falsos positivos: 206 alertas/490.704 peticiones (0,04 %) con el *dataset* de Google, 3 alertas/4.617 peticiones (0,06 %) con el de la UCSB y 151 alertas/713.500 peticiones (0,02 %) con el de la UTV. Concluyeron: que las violaciones web deben abordarse mediante herramientas y técnicas que mezclen la precisión de la detección basada en firmas, con la flexibilidad de un sistema de detección basado en anomalías; y que las tasas de falsos positivos podrían reducirse con el refinamiento de los algoritmos utilizados. Este sistema, al igual que al anterior, soluciona el problema de detección de violaciones, pero lo aborda desde una perspectiva basada en anomalías, diferente a lo propuesto en el proyecto actual.

Hybrid Approach for detecting Malicious Web pages using decision tree and naive Bayes algorithms (HAMW)

Adebayo y Onashoga [7] proponen un modelo de detección de URL maliciosas híbrido utilizando árboles de decisión y *naive* Bayes como algoritmos de clasificación de páginas web entre maliciosas y legítimas. Para la evaluación de este sistema usaron un *dataset* con tres mil páginas web legítimas y 355 maliciosas obtenidas de los sitios web del *ranking* Alexa [8] y PhishTank [9] respectivamente. El sistema utiliza un método de tres fases: la primera, el mecanismo innato para la validación basada en firmas de la página; la segunda, enfocada en la extracción de variables de manera dinámica; y la tercera, la clasificación mediante los algoritmos citados.

El sistema obtuvo una tasa de detección del 93,1 % y una tasa de falsos positivos de 6,9 %. La evaluación de los algoritmos de clasificación utilizados mostró mejores resultados para árboles de decisión (83,1 % de detección y 16,9 % de falsos positivos) en comparación con *naive* Bayes (66,1 % de detección y 33,9 % de falsos positivos). Concluyeron que el sistema híbrido ofrece mejores resultados que el uso de los dos algoritmos de clasificación por separado y propusieron continuar la investigación con la aplicación de ese método a un sistema de

detección de intrusiones. Si bien este sistema cuenta con un método que utiliza modelos de *machine learning*, difiere del proyecto actual porque no se centra en la detección de ciberataques web, sino en la detección de páginas maliciosas.

Detection of Malicious Web pages based on Hybrid analysis (DMWH)

El sistema propuesto por Wang et al. [10] aborda la utilización de un sistema híbrido que combina los aspectos positivos de los análisis estático y dinámico, este último aplicado a las páginas desconocidas para decidir si se trata o no de páginas maliciosas. Para validar este sistema, se utilizaron 1.469 páginas web: 787 legítimas, provenientes de un rastreador web que permite solicitar datos de las páginas web del *ranking* Alexa [8]; y 682 maliciosas, recolectadas desde Malware Domain List [11]. El sistema propuesto utiliza dos conjuntos de variables (uno con estáticas, otro con dinámicas) y cuenta con un proceso dividido en tres métodos: extracción de características estáticas y dinámicas de la página, estas últimas obtenidas a través de un método dinámico de análisis que ejecuta directamente el código JavaScript de la página en un ambiente controlado para su posterior análisis; selección de las características más representativas, mediante su correlación; y clasificación mediante un algoritmo de árboles de decisión.

El sistema obtuvo una precisión del 95,2 %, una tasa de predicción del 88 % y una tasa de falsos positivos de 0,048 %. De su aplicación, se concluyó que, debido a que los atacantes web actualizan sus técnicas para burlar los sistemas de detección existentes, no basta con solo un análisis estático, sino que él debe estar acompañado por un análisis dinámico que compruebe en tiempo de ejecución el comportamiento de la página; y que los resultados pueden mejorar si se refinan los métodos de clasificación y se aumenta el tamaño del *dataset* utilizado. Este sistema no soluciona el problema planteado en el proyecto actual, porque no cuenta con un método de *machine learning* para la detección de amenazas y se enfoca en el análisis de páginas web y no en peticiones hechas a ellas.

Machine Learning Classifiers to detect Malicious Websites (MLCMW)

Urcuqui et al. [12] proponen un sistema de detección que utiliza una serie de variables que permite realizar la detección de sitios web con contenido malicioso mediante cuatro algoritmos de *machine learning*. Utilizaron un *dataset* propio con 967 registros (861 observaciones legítimas y 106 maliciosas), obtenidos de varios sitios web. El sistema propone la división por capas de las características

a evaluar: diez variables en la de aplicación, doce variables en la de red. Estas características se evaluaron usando cuatro algoritmos de clasificación: *Support Vector Machine* (SVM), regresión logística, *naive* Bayes y C 4.5 en su versión de Weka, denominado J48.

Este sistema, basado en *machine learning*, obtuvo una exactitud de: 94,71 % para SVM, 90,58 % para regresión logística, 10.96 % para *naive* Bayes y 98,76 % para J48. Los investigadores concluyeron que el método de obtención de los datos no está documentado, por lo que proponen una búsqueda exhaustiva para mejorar los resultados presentados y destacaron el mejor desempeño del algoritmo J48 al momento de identificar una página web maliciosa. Este sistema, aunque cuenta con un método de *machine learning* (que es el método propuesto por el proyecto actual), no se enfoca en el mismo problema, pues se centra en la detección de páginas maliciosas.

Defending Malicious Script attacks using Machine Learning classifiers (DMSML)

Khan et al. [13] proponen un sistema que presenta un interceptor entre el navegador y el servidor que analiza el código JavaScript para clasificar la petición como legítima o maliciosa. Utilizaron un *dataset* con 1.924 registros, de los cuales 1.515 eran legítimos y 409 maliciosos. Su sistema propone la extracción de características del código JavaScript, extrayendo únicamente aquellas que son relevantes, para lo que utilizan un método *wrapper*. Estas características se evaluaron mediante cuatro algoritmos de clasificación: *naive* Bayes, C 4.5 (J48), SVM y k-*Nearest Neighbors* (k-NN), utilizando tres pruebas: 100 % *training*, 80 % *training* 20 % *test* y 10- *fold cross-validation*. Sus resultados, en términos de exactitud, se presentan en la Tabla 3.1.

Tabla 3.1. Resultados de las pruebas del sistema DMSML [13] (% de exactitud)

Algoritmo	Prueba		
	100 % training	80 % training 20 % test	10- fold cross-validation
Naive Bayes	97,25	95, 06	97,99
J48	97,09	99,22	98,64
SVM	96,51	94,55	95,42
k-NN	100,00	97,14	96,41

Se concluyó que los resultados del sistema propuesto fueron buenos (mejores para k-NN) y que el método usado para la extracción de características tuvo un papel importante en ese buen desempeño. Este sistema, si bien usa *machine learning*, no soluciona completamente el problema propuesto en el proyecto actual, ya que se centra únicamente en la detección de un tipo de ciberataque (*Cross Site Scripting*, XSS).

Un resumen de lo encontrado en la realización del estado del arte frente a lo previsto en el proyecto actual, se presenta en la Tabla 3.2.

Tabla 3.2. Comparativo de sistemas presentes en el estado del arte

Algoritmo	Proyecto						
	DSSA	ADWA	HAMW	DMWH	MLCMW	DMSML	Actual
Incluye machine learning	No	No	Si	Si	Si	Si	Si
Detecta ataques web	Si	Si	No	No	No	Si	Si
Realiza análisis híbrido	No	No	Si	Si	No	No	Si
Valida mediante simulador de ataques	No	No	No	No	No	No	Si

3.3 La investigación

3.3.1. Selección de riesgos informáticos

OWASP es una comunidad abierta dedicada a desarrollar, mantener y comprar aplicaciones y API que sean confiables. Los recursos de OWASP dirigidos a incrementar la seguridad de las aplicaciones web incluyen la publicación periódica de un Top 10 con una descripción de los riesgos informáticos más críticos, sus posibles repercusiones y recomendaciones para evitar el riesgo. Con base en el Top 10 para 2017, el proyecto seleccionó cuatro riesgos informáticos: inyección de datos, *broken authentication*, exposición de datos sensibles y *Cross-Site Scripting* (XSS).

La inyección de datos consiste en el envío de datos no confiables que son interpretados como parte de la búsqueda, mediante ella, los atacantes pueden acceder a información confidencial o ejecutar comandos; *broken authentication* se refiere al manejo inseguro del control de las sesiones o las autenticaciones,

mediante el cual los atacantes pueden obtener acceso a las aplicaciones y realizar acciones en nombre de un usuario determinado; la exposición de datos sensibles, se refiere a deficiencias en la protección de datos considerados confidenciales, cuya exposición podría permitirle a los atacantes acceso a ellos con fines maliciosos; finalmente, el XSS consiste en la inyección de código HTML o JavaScript en aplicaciones, de tal modo que le permitan a un atacante ejecutar este código en el navegador de la víctima, lo que puede ser usado con diferentes intenciones, entre ellas conseguir sesiones de usuarios, *realizar defacement* o redirigir al usuario a sitios maliciosos.

Una vez definidos estos riesgos, el proyecto decidió enfocarse en dos: la inyección de datos, específicamente SQL (*SQL Injection*, SQLI) y XSS. Estos ataques web están directamente relacionados con dos riesgos informáticos e indirectamente relacionados con los otros dos, por lo que representan las fallas de seguridad más graves, lo que amerita que la investigación se centre en la extracción de sus características.

3.3.2. Selección del ambiente a utilizar

Un *honeypot* es un ambiente computacional monitoreado que funciona como anzuelo. Existen de diversos tipos, cada uno con funciones particulares. De ellos, el proyecto analizó dos: los *honeypot servers* y los *honeypots web*. Los primeros, tienen como función emular un servidor vulnerable para atraer a los atacantes y obtener información acerca de los ataques realizados, y pueden tener herramientas de monitoreo de red y sistemas de alertas para notificar la presencia de un intruso o de un ataque. Los segundos, se enfocan únicamente en aplicaciones web y frecen herramientas para monitorear las peticiones y determinar si son maliciosas o no.

En el proyecto, el uso de un *honeypot* tiene como fin poder monitorear y extraer información acerca de peticiones maliciosas realizadas al servidor y a la aplicación web. Por lo tanto, las características deseables son: capacidad de adaptabilidad, automatización de la captura de la información, clasificación de las peticiones y generación de una captura de tráfico. Para realizar su análisis y comparación, se tuvieron en cuenta características como: la última fecha de actualización, las vulnerabilidades asociadas, el tipo de interacción y la descripción de su funcionamiento.

Durante la primera iteración se realizó el análisis y clasificación de 17 h*oneypot servers* (ver [14], Anexo 1), de los cuales se seleccionaron tres. Con ellos se realizó

una instalación y prueba de sus funcionalidades, con el fin de determinar si cumplían con el objetivo de generación del *dataset*.

Para la selección de los *honeypot servers* a analizar, se tuvo en cuenta: su capacidad de detección, la generación de capturas de tráfico, la clasificación de peticiones y el tipo de enfoque de la herramienta. Los *honeypot servers* seleccionados para análisis se presentan en la Tabla 3.3.

Tabla 3.3. Honeypot servers seleccionados

Honeypot	Descripción
Beeswarm	*Honeypot* tipo server que tiene como objetivo determinar el comportamiento normal de una red, para así poder clasificar el comportamiento anómalo. Actualmente no cuenta con documentación del proyecto y no tiene soporte. Se intento realizar una instalación y prueba del *Honeypot* pero no fue posible por la falta de guías y una serié de errores en el código fuente que no fueron posibles de solucionar.
Conpot	Proyecto en desarrollo que tiene como objetivo simular una red empresarial grande, que cuenta incluso con algunos tipos de dispositivos diferentes a servidores. Al ser un proyecto aún en desarrollo al realizar la instalación y configuración del ambiente se encontró que algunas de las herramientas tenían errores y no era posible realizar una configuración que supliera las necesidades del proyecto.
KFSensor	Herramienta comercial en constante desarrollo, tiene herramientas de monitoreo, clasificación y generación de las capturas de tráfico, cuanta con soporte y facilita la generación de escenarios y servidores de FTP y SMTP, entre otros. Su instalación es sencilla, pero está limitada a Windows. A pesar de sus ventajas, no fue posible realizar una configuración de los servicios HTTP y HTTPS puesto que solo tiene como opción arrojar un banner acerca del tipo de servidor que se está corriendo.

La revisión de los *honeypot servers* seleccionados mostró que no satisfacen las necesidades del proyecto porque no permiten una emulación completa del servicio web o realizar la captura de la información, características necesarias para poder generar el *dataset*. No poder emular completamente el servicio web y sus vulnerabilidades hace imposible obtener un *dataset* con todas las características necesarias; y no poder tener un registro de la captura de información, hace imposible generar el conjunto de datos necesario para el estudio.

Al encontrar estas limitaciones, se decidió buscar un *honeypot web* que cumpliera con las características de emulación de vulnerabilidades de aplicaciones web y tuviera además la capacidad de caracterizarlas. El análisis de diecinueve

honeypots web (ver detalle en [14], Anexo 2) mostró limitaciones parecidas a las del tipo *server*: la mayoría tiene un alcance muy específico, no tiene soporte o es obsoleta. De las diecinueve aplicaciones, se seleccionaron tres para su instalación y prueba de funcionalidad, las mismas que se describen en la Tabla 3.4.

Tabla 3.4. Honeypots web seleccionados

Honeypot	Descripción
Gastpof	Proyecto de *honeypot web* para el estudio de vulnerabilidades en las aplicaciones, actualmente se encuentra obsoleto y tiene una segunda versión en dos proyectos llamados Snare y Tanner. No se pudo probar debido a errores en la versión accesible en GitHub. De acuerdo con su descripción, tenía las herramientas necesarias para el estudio realizado en este proyecto.
Snare/Tanner	Aunque se trata de dos proyectos separados, están orientados para funcionar juntos: Snare se encarga de realizar una copia de una aplicación web insertando vulnerabilidades conocidas y desplegándola como un anzuelo, mientras que Tanner realiza el análisis de las peticiones HTTP que se realizan a Snare. Ambos proyectos aún están en desarrollo, por lo cual aún tienen errores en su instalación y clonación de las páginas.
Shadow-Daemon	*Honeypot* tipo *firewall* de alta interacción enfocado en el control de las peticiones que se realizan a una aplicación, con la limitante de que las aplicaciones deben estar escritas en PHP, Perl o Python. Su funcionamiento se basa en la generación de una *whitelist* para realizar un filtro a partir de anomalías y una blacklist para identificar un conjunto de posibles formas de ataque. Permite clasificar las peticiones asociadas a un ataque realizado. Sus desventajas son: la dificultad para generar un conjunto de reglas (que funcionará para la aplicación) para la blacklist; y la imposibilidad de generar una whitelist, pues hacerlo requiere de un tiempo mínimo de uso normal por diferentes usuarios (que hizo imposible exportar la información generada por el *honeypot*).

Adicionalmente, fue necesaria la comparación entre aplicaciones web vulnerables (BTSLab, Bodge It, Juice Shop Project, Bricks, bWAPP y DVWA), con el fin de determinar la más adecuada para el proyecto. Luego de analizar las vulnerabilidades ofrecidas, la fecha de actualización y el lenguaje (ver detalles en [14], Anexo 3), se seleccionó DVWA.

Al no poder utilizar estas herramientas para la obtención de la información, debido a sus diferentes restricciones, se optó por utilizar un ambiente vulnerable (ver Figura 3.1).

Para exponer la aplicación vulnerable, se utilizó un servidor Ubuntu, y como herramienta de captura se seleccionó TShark, la cual genera una captura de

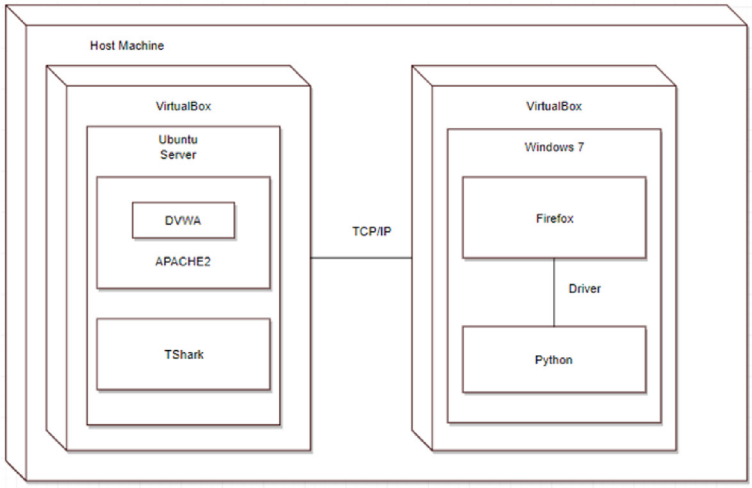

Figura 3.1. Arquitectura del sistema vulnerable planteado en el proyecto

tráfico por tiempo y en formato PCAP. Para la parte del cliente, se utilizó Firefox como navegador predeterminado y a través de un *script* de Python se realizó su control para el acceso y la realización de los ataques sistemáticos. Como fuente de información para los ataques se utilizaron listas de ataques tipo XSS y SQLI.

5.1.6. Análisis de las variables de detección

La identificación de variables es un paso clave en la detección de ciberataques, pues de su correcta identificación depende el éxito que tendrá el clasificador utilizado. Estas características describen el comportamiento de un ciberataque y plasman la interacción que el ciberataque tiene en sus distintas fases, tanto en la capa de aplicación como en la capa de red.

Se analizaron varias propuestas, en ellas se evaluaron varias características para la identificación de ciberataques, tomando como base las variables identificadas por Urcuqui et al. [12].

Salem y Karim [15] proponen una serie de características para el protocolo HTTP que pueden ser útiles al momento de detectar ciberataques, tales como XSS, SQLI, y DoS, entre otros: información general de la petición, contenido de la petición, respuesta del servidor e historial de solicitudes.

Por su parte, Wazzan y Awadh [16] listan una serie de variables identificadas en una petición HTTP: URL, método, *host*, parámetros, *cookies*, *referer* URL, información delicada, archivo sospechoso y patrón sospechoso.

Por su parte, Khan et al. [13] proponen setenta características de detección para ciberataques XSS obtenidas mediante un método de extracción de características. Asimismo, Patil y Patil [17] proponen el análisis de algunas funciones JavaScript utilizadas frecuentemente en ciberataques, como lo son: eval(), attachEvent() y parseInt(); y Fraiwan et al. [18] hablan sobre la manipulación del DOM mediante código JavaScript, y analizan el comportamiento de ciertas funciones de JavaScript juntas, como: eval(), unescape() y find().

De otro lado, Nunan et al. [19] resaltan una característica importante de los ciberataques web mediante peticiones HTTP: la ofuscación del código, un mecanismo que consiste en cambiar la codificación del código enviado para ocultar su estructura maliciosa. Con base en esta premisa, para la extracción de variables utilizaron funciones para convertir el código ofuscado. Debido a la ofuscación del código, la longitud de la URL puede crecer, por lo que el autor presenta una característica importante: la longitud de la URL. Además, analizan dos características nuevas, el redireccionamiento hacia otras páginas mediante el código y la presencia de *tags* HTML en él.

Como se puede observar, los autores proponen clasificaciones diferentes para las características, pero estas no difieren mucho de una investigación a otra; en todas se tiene en cuenta el análisis de características comunes en los ciberataques: el análisis de las cabeceras de la petición; el llamado a métodos o funciones JavaScript, en los ciberataques XSS; y el ingreso de sentencias SQL, en los ataques SQLI).

Las características citadas fueron la base para seleccionar las características finales que se usarían en el proyecto. Se realizó su depuración, dejando únicamente aquellas que en mayor medida puedan identificar correctamente los ciberataques con los que se va a trabajar. Las trece variables seleccionadas se presentan en la Tabla 3.5.

3.3.3. Ambiente vulnerable y simulación ciberataques

Durante este proyecto se simuló un ambiente vulnerable. A su vez, en él se simularon ciberataques XSS y SQLI, de acuerdo con el procedimiento descrito a continuación.

Tabla 3.5. Variables de detección seleccionadas

Nombre	Tipo	Descripción
req_uri_length	Entero positivo	Longitud de la URI en la petición.
req_method	Simbólico	Tipo de petición (POST/GET)
non_printchars	Entero positivo	Número de caracteres no imprimibles en la petición.
contains_sql_commands	Booleano	¿La petición contiene comandos SQL?
sensitive_files	Booleano	¿La petición contiene llamados a archivos especiales (../etc/passwd)?
directory_traversal	Booleano	¿La petición contiene recorridos entre directorios (../ , ../../)?
default_login_credentials	Booleano	¿La petición contiene credenciales de acceso predeterminadas (admin, guest, user)?
contains_html_tags	Booleano	¿La petición contiene tags HTML (<iframe>, <script>)?
dom_manipulation	Booleano	¿La petición contiene llamados para la manipulación del DOM?
url_chain	Booleano	¿La petición contiene redirecciones hacia otras páginas?
eval_func_presence	Booleano	¿La petición contiene llamados a la función de JavaScript eval()?
unescape_func_presence	Booleano	¿La petición contiene llamados a la función de JavaScript unescape()?
write_func_presence	Booleano	¿La petición contiene llamados a la función de JavaScript write()?
getelementsbytagname_func_presence	Booleano	¿La petición contiene llamados a la función de JavaScript getElementByTagName()?
alert_func_presence	Booleano	¿La petición contiene llamados a la función de JavaScript alert()?
fromcharcode_func_presence	Booleano	¿La petición contiene llamados a la función de JavaScript fromCharCode()?

1. Se recaudaron diferentes *payloads* asociados a ciberataques XSS y SQLI encontrados en la web, como base de la simulación.
2. Se simuló un ambiente vulnerable, utilizando dos máquinas virtuales: la primera, una Intel Core i7, 3770K, 16GB RAM, con Ubuntu 16.04 LTS, en el rol de servidor atacado, y DVWA [20] como aplicación vulnerable; la segunda, una Intel Core i7, 7700HQ, 16GB RAM, con Windows 7 Ultimate, en el rol de máquina atacante.

3. Se creó un *bot* en Python, utilizando Selenium [21], en el cual se simulaba el acceso al ambiente vulnerable y se inyectaban los *payload* descritos.
4. Se utilizó TShark [22] en la máquina atacada para la captura del tráfico generado por la máquina atacante.
5. Cada captura tenía asociado un *payload* distinto.

El tiempo total de cada captura fue de tres minutos y se realizaron, en total, 547 capturas de tráfico así: 115 asociadas a ciberataques SQLI y 432 ciberataques XSS.

3.3.4. Análisis exploratorio

Para el desarrollo del proyecto se generó un *dataset* utilizando un algoritmo de extracción de las variables ya mencionadas (ver detalle en la Figura 3.2. El *dataset* cuenta con 1.129 peticiones HTTP, de ellas, 609 legítimas y 520

```
req_uri_length   req_method non_printchars    contains_sql_commands
Min.   :  1.00   GET :585   Min.   :0.00000   Mode :logical
1st Qu.: 15.00   POST:544   1st Qu.:0.00000   FALSE:1048
Median : 28.00              Median :0.00000   TRUE :81
Mean   : 28.74              Mean   :0.06555
3rd Qu.: 28.00              3rd Qu.:0.00000
Max.   :235.00              Max.   :7.00000
sensitive_files directory_traversal default_login_credentils contains_html_tags
Mode :logical   Mode :logical       Mode :logical            Mode :logical
FALSE:1129      FALSE:1129          FALSE:657                FALSE:947
                                    TRUE :472                TRUE :182


dom_manipulation url_chain       eval_func_presence unescape_func_presence
Mode :logical    Mode :logical   Mode :logical      Mode :logical
FALSE:1113       FALSE:1127      FALSE:1129         FALSE:1129
TRUE :16         TRUE :2


write_func_presence getlementsbytagname_func_presence alert_func_presence
Mode :logical       Mode :logical                      Mode :logical
FALSE:1127          FALSE:1129                         FALSE:1063
TRUE :2                                                TRUE :66


fromcharcode_func_presence        attack_type       type
Mode :logical              NONE          :609   BENIGNA:609
FALSE:1129                 SQL INJECTION: 88   MALIGNA:520
                           XSS           :432
```

Figura 3.2. Estructura del dataset utilizado

maliciosas. A su vez, de estas 520 peticiones malignas, se etiquetaron 88 ciberataques SQLI y 432 ciberataques XSS.

3.3.5. Entrenamiento, validación y selección del método de detección

En el proyecto se planteó la evaluación de cuatro algoritmos de clasificación: C 4.5 [23], *random forest* [24], *naive* Bayes [25] y k-NN [25]. Para el entrenamiento se utilizó el método *k-fold cross validation* [26], el cual asegura que el modelo se entrene adecuadamente con el *dataset*, y se utilizó un K=10 para este algoritmo, como sugiere Kohavi [26]. En el caso específico del entrenamiento del algoritmo k-NN, para la selección del K óptimo, aunque normalmente se opta por una selección empírica basada en la experimentación, en este proyecto fue determinado automáticamente por la librería Caret [27].

Para la validación se utilizó un *dataset* (fusión de dos *datasets* tomados de Torrano, Pérez y Álvarez [28]) que consta de 60.668 peticiones HTTP (36.000 legítimas y 24.668 anómalas, incluyendo ciberataques como SQLI, XSS, desbordamiento de *buffer*, inyección CRLF y manipulación de parámetros, entre otros).

Las peticiones anómalas no se encuentran etiquetadas con ningún tipo de ciberataque, por lo que no se puede saber con exactitud cuántas peticiones corresponden a cada ciberataque. Para cada clasificador se evaluaron tres indicadores principales: Kappa [29], precisión [30] y *F-Measure* [30], los que se muestran al final de cada experimento, en un cuadro comparativo.

Se utilizó también la matriz de confusión de cada clasificador, herramienta que permite la visualización del desempeño de un algoritmo de clasificación (Figura 3.3) y presenta otros indicadores importantes, tales como:

- el número de instancias legítimas clasificadas como maliciosas (falsos positivos);
- la cantidad de instancias maliciosas clasificadas como legítimas (falsos negativos);
- el *recall* (proporción de verdaderos positivos clasificados correctamente); y
- la especificidad (proporción de verdaderos negativos clasificados correctamente).

Estos últimos indicadores se muestran en cada algoritmo de clasificación.

		Predicción	
		Uva	Fresa
Valor real	Uva	10	3
	Fresa	5	13

Figura 3.3. Ejemplo matriz de confusión

Para mayor claridad de los indicadores mencionados, se realizó la matriz de la Figura 3.4, la cual responde a las siguientes convenciones: A = Verdaderos Positivos (VP); B = Falsos Positivos (FP); C = Falsos Negativos (FN); D = Verdaderos Negativos (VN); A+B = Predicciones Positivas; C+D = Predicciones Negativas (PN); A+D = valores Reales Positivos (RP); y B+D = valores reales negativos (RN).

		Predicción		
		Uva	Fresa	Total
Valor real	Uva	A	B	A +B
	Fresa	C	D	C + D
	Total	A + C	B + D	N

Figura 3.4. Explicación de los indicadores en la clasificación binaria

Se realizaron dos experimentos de validación: en el primero se utilizó el *dataset* completo como medio de prueba; el segundo fue creado con base en los resultados del experimento 1.

En este segundo caso, siguiendo la filosofía de la ciencia de datos, que indica que los resultados de un experimento se pueden refinar para servir como entradas del siguiente, se adaptó una parte del *dataset* (solo el considerado anómalo por Torrano et al. [28]), utilizando el algoritmo de extracción de características, enfocado en ciberataques XSS y SQLI, para generar un nuevo *dataset*. En él se etiquetaron como peticiones maliciosas aquellas que están asociadas a estos ciberataques, mientras que el resto se consideró legítimo, aunque no lo es, ya que el alcance del proyecto se definió específicamente en ciberataques XSS y SQLI, sin entrar a evaluar los demás (desbordamiento de *buffer*, inyección CRLF, manipulación de parámetros, etc.).

Dado el enfoque de analítica y ciencia de datos del proyecto, se utilizó una metodología propuesta por IBM llamada ASUM-DM [22] (Figura 3.5), en la cual, dados los resultados del experimento 1, se realiza una iteración al momento de resultados; es decir, se hace una retroalimentación de los resultados a los clasificadores y un refinamiento del *dataset*, que da como resultado el experimento 2.

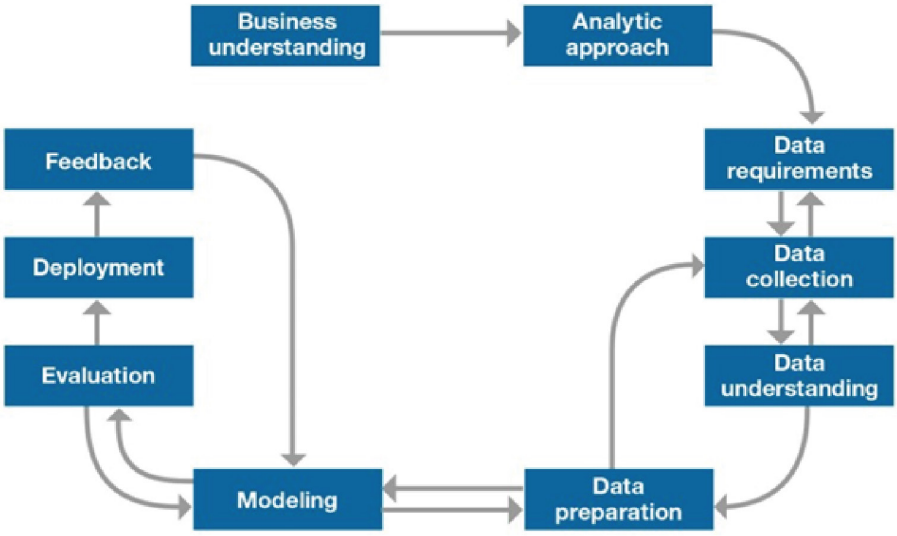

Figura 3.5. Metodología ASUM-DM

3.4. Resultados

Experimento 1

Random forest (ver Figura 3.6) presenta una gran cantidad de falsos positivos, un error muy grave en la seguridad, ya que clasifica como legítima una petición maliciosa, es decir que no detecta el ciberataque. Aunque cuenta con buen *recall*, su especificidad es bastante mala. En conclusión, no es un clasificador adecuado para este proyecto.

Tal como ocurre con el clasificador anterior, k-NN (ver Figura 3.7) presenta una gran cantidad de falsos positivos, a lo que se le suma una cantidad considerable de falsos negativos; su *recall* y especificidad son mejores, pero la

cantidad de errores sigue siendo bastante considerable, en especial los falsos positivos. En conclusión, no es un clasificador adecuado para este proyecto.

C 4.5 (Figura 3.8) presenta resultados similares a los de *random forest*. Una explicación probable para esta similitud es que, al ser ambos árboles de decisión, se comportan de manera similar. La conclusión es parecida también, no es adecuado para este proyecto debido a la alta cantidad de falsos positivos que presenta.

Naive Bayes (Figura 3.9), por su parte, presenta un número muy bajo de falsos positivos, pero un gran número de falsos negativos y un *recall* de 0, lo que indica que no se clasificó adecuadamente ninguna petición legítima; su especificidad es buena. Se concluye con estos indicadores que, aunque tiene un alto número de falsos negativos, ese es un error para tener presente en entornos productivos, sin embargo, es un modelo que logra capturar en mayor medida la cantidad de casos maliciosos.

		Valores reales	
		Legítima	Maliciosa
Predicción	Legítima	35.994	24.122
	Maliciosa	6	546

Indicador	FP	FN	Recall	Especificidad
Valor	24.122	6	0.999	0.022

Figura 3.6. Experimento 1: indicadores para random forest

		Valores reales	
		Legítima	Maliciosa
Predicción	Legítima	26.995	17.795
	Maliciosa	9.005	6.874

Indicador	FP	FN	Recall	Especificidad
Valor	17.795	9.005	0.749	0.278

Figura 3.7. Experimento 1: indicadores para k-NN

		Valores reales	
		Legítima	Maliciosa
Predicción	Legítima	35.994	24.122
	Maliciosa	6	546

Indicador	FP	FN	Recall	Especificidad
Valor	24.122	6	0.999	0.0221

Figura 3.8. Experimento 1: indicadores para C 4.5

		Valores reales	
		Legítima	Maliciosa
Predicción	Legítima	0	29
	Maliciosa	36.000	24.639

Indicador	FP	FN	Recall	Especificidad
Valor	29	36.000	0	0.998

Figura 3.9. Experimento 1: indicadores para naive Bayes

Los resultados finales del experimento 1 se resumen en la Tabla 3.6. Con base en los indicadores individuales de cada clasificador, para el proyecto se podrían descartar: *random forest*, C 4.5 y k-NN, decisión que se ratifica al revisar los resultados presentes en la Tabla 3.6, en donde para ellos se observa una precisión ligeramente más alta que la del *baseline* y un Kappa demasiado bajo. La única opción parecía ser *naive* Bayes, que si bien tuvo un gran número de

Tabla 3.6. Experimento 1: comparativo de resultados de los algoritmos de clasificación

	Accuracy	Kappa	F Score
Baseline	0.539	-	-
Random forest	0.602	0.025	0.748
k-NN	0.558	0.030	0.668
C 4.5	0.602	0.025	0.748
Naive Bayes	0.406	-	-

falsos negativos, mostró buena especificidad, sin embargo, se debe descartar también porque su precisión es inferior al *baseline* y su Kappa es cero. En conclusión, para el experimento 1, no se selecciona ningún clasificador.

Experimento 2

En este caso, *random forest* (Figura 3.10) presenta un número considerable de falsos positivos, que como se dijo es muy grave en seguridad; sin embargo el hecho de no presentar falsos negativos, tener un *recall* muy bueno y una especificidad alta, hace que sea mejor esperar los resultados de los demás clasificadores (Tabla 3.7).

k-NN (Figura 3.11), por su parte, presenta un número muy alto, tanto de falsos positivos como de falsos negativos y una especificidad muy baja; por ello, aunque cuenta con un buen *recall*, no es un buen clasificador para este proyecto.

Tal como sucedió en el experimento 1, el clasificador C 4.5 (Figura 3.12) presenta resultados similares a los del clasificador *random forest*. Por lo tanto, la

		Valores reales	
		Legítima	Maliciosa
Predicción	Legítima	23.242	542
	Maliciosa	0	884

Indicador	FP	FN	Recall	Especificidad
Valor	542	0	1	0.619

Figura 3.10. Experimento 2: indicadores para random forest

		Valores reales	
		Legítima	Maliciosa
Predicción	Legítima	16.779	944
	Maliciosa	6.463	48

Indicador	FP	FN	Recall	Especificidad
Valor	944	6.463	0,721	0.338

Figura 3.11. Experimento 2: indicadores para k-NN

conclusión es similar, su conveniencia se debe evaluar comparativamente con los resultados de los demás. Sin embargo, no deja de ser preocupante el alto número de falsos positivos.

Finalmente, *naive* Bayes, aunque no presenta ningún falso positivo, si muestra una gran cantidad de falsos negativos; y aunque su especificidad es excelente, tiene un *recall* bastante bajo (0.001), por lo que la decisión de descartarlo o no debería tomarse comparando sus resultados con los de los demás.

		Valores reales	
		Legítima	Maliciosa
Predicción	Legítima	23.242	542
	Maliciosa	0	884

Indicador	FP	FN	Recall	Especificidad
Valor	542	0	1	0.619

Figura 3.12. Experimento 2: indicadores para C 4.5

		Valores reales	
		Legítima	Maliciosa
Predicción	Legítima	29	0
	Maliciosa	23.213	1.426

Indicador	FP	FN	Recall	Especificidad
Valor	0	23.213	0.001	1

Figura 3.13. Experimento 2: indicadores para naive Bayes

Los resultados finales del experimento 2 se resumen en la Tabla 3.7. Con base en los indicadores de cada clasificador, se descarta k-NN por su excesiva cantidad de falsos positivos y porque aún cuando su precisión es ligeramente superior al *baseline*, tiene un Kappa muy bajo. *Naive* Bayes también se descarta, porque aun cuando no posee falsos positivos sí tiene un excesiva cantidad de falsos negativos, una precisión muy baja, inferior al *baseline*, y el peor Kappa.

Tabla 3.7. Experimento 2: comparativo de resultados de los algoritmos de clasificación

	Accuracy	Kappa	F Score
Baseline	0.539	-	-
Random forest	0.978	0.754	0.988
k-NN	0.699	0.021	0.819
C 4.5	0.978	0.754	0.988
Naive Bayes	0.059	0.001	0.002

En ese contexto, aunque *random forest* y C 4.5 presentan una cantidad considerable de falsos positivos (cerca de 38 % de las peticiones maliciosas se clasificaron como legítimas), muestran una muy buena precisión y un buen Kappa. En consecuencia, son los más indicados para este proyecto, dejando de lado los falsos positivos, presentan muy buenos resultados.

Propuesta de un sistema de software para el estudio de ciberataques web

Como un componente adicional al desarrollo de un método de detección de ciberataques web, se tiene como objetivo realizar un sistema que permita su estudio, para lo cual se realizó un proceso de ingeniería de software empezando por la licitación y finalizando en la etapa de diseño. Como trabajo a futuro se propone iniciar un desarrollo del software propuesto.

Para la licitación de los requerimientos se realizaron tres sesiones: en la primera se realizó un esquema preliminar del sistema; las dos sesiones siguientes se utilizaron a modo de validación y retroalimentación para el proceso. El Anexo 1 corresponde a este documento que tiene como finalidad esclarecer los requerimientos iniciales del proyecto. Al ser una propuesta inicial del proyecto no se tuvo en cuenta requerimientos no funcionales organizacionales.

Posteriormente se realizó un proceso de análisis de requerimientos usando el método Dorfman para realizar su subdivisión y sub-especificación. Este análisis tuvo como fruto una división en nueve categorías y veinticinco requerimientos (disponible en [14], Anexo 8); como complemento se preparó una serie de casos de uso [14, Anexo 9].

Para el diseño global se realizaron dos iteraciones, en una se desarrolló la propuesta del modelo de datos y en la segunda, el diagrama de *deployment* ([14, Anexo 10]. Para el diagrama de *deployment* se tuvo en cuenta un patrón cliente-

servidor para facilitar el acceso al software; además provee herramientas para la escalabilidad en caso de ser necesaria.

3.5. Conclusiones y recomendaciones

Las pocas facilidades para encontrar herramientas de libre acceso dificulta la creación de nuevos métodos de ciberseguridad. Aunque esto se hace para prevenir que los atacantes obtengan información sensible sobre el funcionamiento de los métodos, también imposibilita a los investigadores de seguir un camino concreto cuando se quieren retomar trabajos futuros propuestos. En este caso, aún es necesaria una mejora en la generación automatizada de información que permita conseguir una base de información más robusta sobre ataques y expandir la cantidad de ataques que se puedan automatizar. Para ayudar con la investigación en este campo, se propone un sistema que permita su estudio, de este modo y sin necesidad de ofrecer información sensible, se pueden realizar pruebas acerca de la efectividad de los métodos y crear los ambientes requeridos por un proyecto específico.

Dentro del análisis realizado por el proyecto, se encontró que los *honeypots* existentes tienen un tiempo de vida corto y no necesariamente se ajustan a las necesidades de la investigación realizada. En el caso específico de este proyecto, no fue posible utilizar estas herramientas debido a su especificidad, la baja cantidad de configuraciones y la incapacidad de generar una captura de la información. Esto ilustra la necesidad de generar una herramienta que permita satisfacer las necesidades de los investigadores al momento de obtener información sobre ataques.

Adicionalmente, al momento de buscar *payloads* de XSS y SQLI se encontró que existen algunas herramientas para realizar este tipo de ataques y así obtener la información necesaria, pero debido a la necesidad de automatización, no pudieron ser utilizadas. La solución que se encontró fue buscar listas de *payloads*, de hecho, durante este proceso, se encontraron varias fuentes no oficiales, que fueron utilizadas en la investigación. Dada la importancia que tiene esta información, es necesario tener una fuente confiable, que valide la veracidad de lo expuesto y que regule el acceso únicamente para realizar los estudios.

Los resultados obtenidos en la validación de los métodos de detección no fueron los mejores, lo que se puede atribuir a la diversidad del *dataset* de entrenamiento seleccionado, pues de las 25.000 peticiones anómalas, muchas correspondían a

comportamiento anómalo en una aplicación —por ejemplo, insertar texto en un campo para el número de teléfono—, y otros ciberataques no definidos en el alcance de este proyecto. También incluía ciberataques XSS y SQLI, pero estos eran una parte mínima del total del *dataset*, por lo que los algoritmos de detección entrenados con el *dataset* que se generó (creado únicamente con muestras de ciberataques XSS y SQLI) no estaban preparados para detectar las demás anomalías en el *dataset*, lo que dio como resultado unos indicadores no tan buenos, así hayan superado la exactitud (*accuracy*) de la línea base.

Cuando se realizó el experimento con el *dataset* procesado por el algoritmo de extracción de variables, se obtuvo muy buenos resultados, lo que demuestra que las variables para los ataques SQLI y XSS son idóneas y que además, los métodos de detección basados en modelos de *machine learning* son efectivos.

Como se habla de la ciencia de los datos, se propone como trabajo a futuro una retroalimentación a los algoritmos de clasificación propuestos en este proyecto, mejorando las características de clasificación y ajustándolas no solo a los dos ciberataques aquí trabajados, sino a diferentes tipos de ciberataque. Asimismo, se sugiere realizar mejoras en el algoritmo de extracción para reducir la ocurrencia de falsos positivos.

Los recursos (documentos, gráficos, códigos, etc.) obtenidos durante la ejecución del proyecto están disponibles en [31].

Referencias

[1] S. Maconachy y W. Ragsdale, "A model for information assurance: an integrated approach," *Proc. 2001 IEEE Work. Inf. Assur. Secur.* US Mil. Acad. West Point, NY, pp. 5–6, 2001.

[2] E. Corchado y Á. Herrero, "Neural visualization of network traffic data for intrusion detection," en *Applied Soft Computing Journal*, vol. 11, no. 2, pp. 2042–2056, 2011.

[3] C. C. Urcuqui, M. G. Peña, J. L. O. Quintero, y A. Navarro, "Antidefacement," *Sist. Telemática*, vol. 14, no. 39, pp. 9–27, Feb. 2017.

[4] J. Saxe e Invincea Labs, "Why security data science matters and how it's different: pitfalls and promises of data science-based breach detection and threat intelligence," in *Black Hat USA*, 2015.

[5] I. Corona y G. Giacinto, "Detection of server-side web attacks," *J. Mach. Learn. Res.*, vol. 11, pp. 160–166, 2010.

[6] C. Kruegel y G. Vigna, "Anomaly detection of web-based attacks," en *Proceedings of the 10th ACM conference on Computer and communication security - CCS '03*, 2003, p. 251.

[7] A.-A. Adebayo y S. A. Onashoga, "A hybrid approach for detecting malicious web pages using decision tree and naïve Bayes algorithms," *Georg. Electron. Sci. J. Comput. Sci. Telecommun.*, vol. 48, pp. 9–17, 2016.

[8] I. Alexa Internet, *Alexa*. Disponible: https://www.alexa.com/

[9] OpenDNS, *PhishTank*. Disponible: http://www.phishtank.com/

[10] R. Wang, Y. Zhu, J. Tan, y B. Zhou, "Detection of malicious web pages based on hybrid analysis," *J. Inf. Secur. Appl.*, vol. 35, pp. 68–74, Aug. 2017.

[11] *Malware Domain List*. Disponible: http://www.malwaredomainlist.com/mdl.php

[12] C. C. Urcuqui, A. Navarro, J. Osorio, y M. García, "Machine learning classifiers to detect malicious websites," en: P*roceedings of the Spring School of Networks,* Pucón, Chile, October 2017. Disponible: http://ceur-ws.org 2017.

[13] N. Khan, J. Abdullah, y A. S. Khan, "Defending malicious script attacks using machine learning classifiers," *Wirel. Commun. Mob. Comput.*, vol. 2017, pp. 1–9, 2017.

[14] B. A. Henao, y Prada, J. S. "Sistema para el estudio de ciberataques web," [proyecto de grado], Cali, Colombia: Universidad Icesi, 2018.

[15] B. Salem y T. Karim, "Classification features for detecting server-side and client-side web attacks," *IFIP International Federation for Information Processing*, vol. 278, pp. 729–733, 2008.

[16] M. A. Wazzan y M. H. Awadh, "Towards improving web attack detection: highlighting the significant factors," en *2015 5th International Conference on IT Convergence and Security, ICITCS 2015 - Proceedings*.

[17] D. R. Patil y J. B. Patil, "Detection of malicious JavaScript code in web pages," *Indian J. Sci. Technol.*, vol. 10, no. 19, pp. 1–12, 2017.

[18] M. Fraiwan, R. Al-Salman, N. Khasawneh, y S. Conrad, "Analysis and identification of malicious JavaScript code," *Inf. Secur. J.*, vol. 21, no. 1, pp. 1–11, 2012.

[19] A. E. Nunan, E. Souto, E. M. Dos Santos, y E. Feitosa, "Automatic classification of cross-site scripting in web pages using document-based and URL- based features," *Proc. - IEEE Symp. Comput. Commun.*, pp. 702–707, 2012.

[20] Dewhurst Security, *DVWA - Damn Vulnerable Web Application*, 2016. Disponible: http://www.dvwa.co.uk/

[21] Selenium (Hrsg.), *Selenium - web browser automation*," 2013. Disponible: http://www.seleniumhq.org

[22] J. Rollins, "Why we need a methodology for data science," *IBM Big Data & Analytics Hub, 2015*. Disponible: http://www.ibmbigdatahub.com/blog/why- we-need-methodology-data-science

[23] H. Chauhan y A. Chauhan, "Implementation of decision tree algorithm C 4. 5," *Int. J. Sci. Res. Publ.*, vol. 3, no. 10, pp. 4–6, 2013.

[24] G. Biau, "Analysis of a random forests model," *J. Mach. Learn. Res.*, vol. 13, pp. 1063–1095, 2010.

[25] J. Han y M. Kamber, *Data mining: concepts and techniques*, Elsevier, 2006.

[26] R. Kohavi, "A study of cross validation and bootstrap for accuracy estimation and model selection," en *14th International Joint Conference on Artificial Intelligence (IJCAI)*, 1995, vol. 2, pp.1137–1143.

[27] M. Kuhn et al. *Package 'CARET' classification and regression training description misc functions for training and plotting classification and regression models*. Disponible: https://cran.r-project.org/web/packages/caret/index.html

[28] C. Torrano, A. Pérez, y G. Álvarez, *HTTP Dataset CSIC 2010*. Disponible: http://www.isi.csic.es/dataset/.

[29] J. Cohen, "A coefficient of agreement for nominal scales," *Educ. Psychol. Meas.*, vol. 20, no. 1, pp. 37–46, Apr. 1960.

[30] D. M. W. Powers, "Evaluation: from precision, recall and f-measure to ROC: informedness, markedness & correlation," *J. Mach. Learn. Technol.* vol. 2, no. 1, pp. 2229–3981, 2011.

[31] C.C Urcuqui. (2019). Sistema para el estudio de ciberataques web (PDG). *i2tResearch* [repositorio GitHub]. Disponible: https://github.com/i2tResearch/Ciberseguridad_web/tree/master/Sistema%20para%20el%20estudio%20de%20ciberataques%20web%20(PDG)

Anexo 1. Requerimientos iniciales del sistema

Se requiere un sistema, que denominaremos Sniff, que tenga capacidad para realizar estudios de ataques web, para lo cual debe permitir crear proyectos. Cada proyecto tendrá un sistema controlado (*honeypot*) que será objeto de estudio; dependiendo del objetivo del estudio, se podrá probar la efectividad de un algoritmo de detección entrenado con un determinado *dataset* o generar el mismo mediante emulación de ataques y captura de ataques reales. Es importante para esta parte que los ataques que sean realizados puedan ser plenamente identificados para su estudio.

El sistema debe generar reportes de los estudios realizados, los cuales deben ser presentados en tablas y gráficos, mostrando comparaciones entre los resultados obtenidos por los diferentes *datasets* usados o los algoritmos probados. Estos reportes deben ser sobre: porcentajes de error como: falsos positivos, falsos negativos y aciertos, e información adicional, dependiendo de lo que se haya testeado.

El sistema debe permitir el manejo de usuarios y roles. Existen tres roles principales: administrador del sistema, investigador y usuario común de Sniff. El administrador del sistema podrá modificar cualquier parámetro de él; el investigador podrá realizar las pruebas en los ambientes creados y realizar las modificaciones a los proyectos de los cuales haga parte; y el usuario común de Sniff no podrá realizar modificaciones en estos ambientes y solo podrá observar los reportes generados.

El sistema debe tener un subsistema de alertas de ataques; cuando la aplicación se encuentre bajo un ataque de cualquier clase, debe lanzar una alerta al administrador del sistema con la información del ataque que está ocurriendo (tipo de ataque, recursos afectados, etc.), para que haga las labores necesarias para proteger al sistema del ataque. Adicionalmente se debe realizar una captura de todo el tráfico que ocurra durante el ataque para su posterior análisis y adición a los entornos de investigación.

Para el estudio de los ciberataques web, el sistema debe permitir:

- crear un proyecto con nombre, colaboradores, fecha de creación, autor, estado y tipo de estudio;
- adicionar y remover colaboradores de un estudio;
- modificar el estado del proyecto;

- modificar el tipo de algoritmo de clasificación y su método de entrenamiento;
- modificar el *dataset* con el cual se entrenó el algoritmo de clasificación;
- modificar el ambiente controlado para añadir o remover vulnerabilidades;
- volver al estado inicial de un ambiente controlado;
- realizar ataques a un ambiente controlado mediante un simulador; y
- configurar los parámetros del simulador de ataques (tipo de vulnerabilidades y frecuencia).

El sistema debe generar reportes: comparativos, con los cambios realizados en los proyectos e indicadores estadísticos para su evaluación (aciertos, falsos positivos, falsos negativos); de los ataques que se hayan realizado a la aplicación; y de funcionamiento del sistema de detección

En cuanto a roles y usuarios, el sistema debe permitir:

- crear roles, modificarlos y eliminarlos;
- crear: nombre, correo, contraseña, rol, estado y modificar un usuario;
- modificar el rol a un usuario;
- modificar la contraseña de un usuario;
- cambiar el estado de un usuario; y
- limpiar una contraseña mediante un correo.

El sistema debe además tener la capacidad de analizar una captura de tráfico de red para: determinar si contiene un ciberataque y determinar su tipo; poder modificar sus propios parámetros; y ser escalable, interoperable, mantenible, modificable, parametrizable y seguro.

Capítulo 4
Sistema de análisis de tráfico web para la detección de malware en dispositivos Android

Andrés Felipe Pérez, Christian Camilo Urcuqui, Andrés Navarro

4.1. Introducción

El uso de dispositivos móviles ha tenido un crecimiento significativo en los últimos años. Y la tendencia se mantendrá porque los teléfonos inteligentes y otros dispositivos móviles, tales como tabletas, relojes de pulsera, auriculares y dispositivos para televisión, seguirán jugando un papel dominante. Android, el sistema operativo desarrollado y mantenido por Google para dispositivos móviles, ofrece un ambiente de desarrollo de software libre y tiene herramientas, aplicaciones y emuladores para desarrollar aplicaciones en Java. Es uno de los sistemas operativos más famosos en el mundo y cuenta con una participación de mercado de 84 % para 2021; su competidor, iOS, tiene el resto del mercado. En 2011, la cuota de mercado combinada de ambos apenas llegaba a 40% [1].

El crecimiento y la popularidad de Android han ocasionado un gran interés en los cibercriminales para crear aplicaciones maliciosas que causan en los usuarios muchos problemas (robo de información o dinero), empleando todo tipo de código malicioso. Y esto no es nuevo, para 2014, por ejemplo: los piratas informáticos pagaban alrededor de dos mil dólares para obtener un programa cifrador de datos (*ransomware*), con él bloqueaban los datos del equipo de un usuario y cobraban cien dólares por el desbloqueo [2]. Y la variedad es amplia, con un troyano se pueden robar los datos de una cuenta bancaria, donde el daño puede ser significativamente mayor.

En vista de esta situación, se han creado métodos para el análisis de *malware* en dispositivos móviles, los cuales, de acuerdo con Tam et al. [3], son: el análisis de la firma del *malware*, el análisis estático, el análisis dinámico y el análisis híbrido. En el análisis de la firma del *malware*, se extraen patrones o

fragmentos aleatorios de una muestra. Sin embargo, este método es costoso y poco efectivo, ya que no solo se pueden obtener muchas firmas, sino que usando técnicas de ofuscación es posible lograr que ciertos tipos de *malware* no sean detectados. El análisis estático examina el código y los datos de la aplicación usando archivos tales como META-INF, AndroidManifest.xml y classes.dex. Al igual que la técnica anterior, los métodos de ofuscación lo hacen poco efectivo. El análisis dinámico consiste en ejecutar un programa y observar su comportamiento, lo que generalmente se hace de una forma instrumentada o monitoreada para recolectar más información concreta de su comportamiento. La desventaja de este tipo de análisis es que cierto tipo de *malware* puede detectar el comportamiento emulado que muchas veces usa este método y tomar medidas para no hacer nada cuando se encuentra en ese entorno. Finalmente, el análisis híbrido aunque obtiene los mejor de los análisis estático y dinámico, también es falible.

La pregunta entonces es si es posible encontrar un método, con mejores resultados. Para ayudar a responder esto, Tam et al. [3] mencionan otras técnicas para el análisis de *malware* que no han sido suficientemente investigadas, entre ellas el análisis de tráfico en Internet. Esto es importante puesto que actualmente la mayoría de aplicaciones requiere de una conexión a la red. De acuerdo con Chen et al. [4], los *malware* para Android tienen una mayor tendencia a solicitar permisos y conexión a Internet, a diferencia de las aplicaciones legítimas.

Para aprovechar los datos transmitidos por la red, actualmente se emplean métodos de análisis de tráfico en la red, para el estudio del comportamiento de cada uno de los equipos (*endpoints*), buscando encontrar anomalías, desarrollar un perfil benigno o crear o ajustar las reglas para detección de amenazas cibernéticas, entre otros, tanto en computadoras de escritorio como en dispositivos móviles, y se estudia la forma en que estos diferentes análisis se pueden aplicar o complementar. Una de las aproximaciones es el estudio de las variables relacionadas con la Web, tema explorado por algunas investigaciones tendientes al desarrollo de soluciones que permitan identificar aplicaciones maliciosas [5].

El objetivo general del proyecto es desarrollar un sistema de análisis de tráfico de red para detectar aplicaciones maliciosas en dispositivos Android. Como objetivos específicos fueron definidos: investigar, examinar y diseñar un método para la generación de tráfico web de aplicaciones Android; implementar dicho

método y extraer las variables del tráfico generado; evaluar un algoritmo de clasificación que utilice dichas variables para detectar si una aplicación es maliciosa o no; y comparar el sistema propuesto y los resultados obtenidos con un estudio previo.

4.2. Estado del arte

Previo al inicio del proyecto se revisaron cuatro investigaciones desarrolladas para el análisis de *malware* en dispositivos Android: Credroid: detección de *malware* en Android mediante el análisis de tráfico de red; Detección de *malware* mediante análisis de tráfico de red en dispositivos móviles basados en Android; DroidAlarm: una herramienta de análisis estático multifuncional para el *malware* de escalado de privilegios en Android; y Una primera mirada al tráfico de *malware* de Android en los primeros minutos. En la Tabla 4.1 se presenta un comparativo entre ellos y el proyecto actual, a partir de las características más relevantes para el tema en cuestión.

Tabla 4.1. Comparativo entre proyectos similares y el actual

Características	Credroid	Detección de malware…	DroidAlarm	Una primera mirada…	Proyecto propuesto
Dataset para pruebas	Si	Si	Si	Si	Si
Extracción e identificación de características	Si	Si	Si	Si	Si
Método de análisis	Dinámico	Dinámico	Estático	Dinámico	Dinámico
Uso de tráfico en la red	Si	Si	No	Si	Si
Entorno virtualizado	Si	Si	No	Si	Si
Análisis del tráfico de aplicaciones no maliciosas	Si	Si	No	No	Si
Uso de herramientas de simulación de movimientos	No	No	No	No	Si

Como se evidencia, uno de los principales diferenciadores del proyecto actual es el uso de una herramienta que permite la simulación de movimientos del usuario con una aplicación, ya que al parecer esto no se tuvo en cuenta en los trabajos evaluados, y esta es una característica importante debido a que

la interacción resultante puede agregar resultados a través de interacciones dinámicas. A continuación, se presenta un mayor detalle de cada uno de los proyectos revisados.

Credroid: detección de malware en Android mediante el análisis de tráfico de red

Credroid es un método que identifica las aplicaciones maliciosas con base en sus consultas a los DNS y a los datos que transmite al servidor remoto, realizando un análisis en profundidad de los registros de tráfico de la red en modo *offline*. En lugar de realizar una detección basada en firmas, que es incapaz de detectar *malware* polimórfico, Mallik y Kaushal [6] proponen una detección basada en patrones, en donde el patrón que se trabaja se refiere a la fuga de información sensible que se envía al servidor remoto.

Credroid tiene un enfoque semiautomático que trabaja sobre diversos factores, tales como: el servidor remoto donde se está conectando la aplicación; los datos que se envían; y el protocolo utilizado para la comunicación para identificar la fiabilidad de la aplicación. En este trabajo, se observó que el 63 % de las aplicaciones de un conjunto de datos estándar de *malware* está generando el tráfico de red foco de este trabajo.

A pesar de que este es un método que ha implementado un sistema de monitoreo muy completo, los investigadores no se enfocaron en obtener características del tráfico sino en monitorear las interacciones de las aplicaciones con el servidor, que es solamente una de las características a analizar en la detección de *malware* usando en tráfico en la red.

Detección de malware mediante análisis de tráfico de red en dispositivos móviles basados en Android

Arora, Garg y Peddoju [5] analizan las características del tráfico de red y crean un clasificador basado en reglas para la detección de *malware* Android. Los resultados experimentales sugieren que el enfoque es notablemente preciso e identifica correctamente más del 90 % de las muestras de tráfico. Su objetivo principal es detectar *malware* controlado remotamente por algún servidor, que obtiene comandos desde dicho servidor o filtra información privada de los usuarios. El funcionamiento del marco de detección se divide principalmente en dos fases: la primera consiste en analizar el tráfico de red del software malicioso y encontrar la lista de aquellas características que distinguen completamente

el tráfico de *malware* del tráfico móvil legítimo; la segunda, en la construcción de un clasificador basado en una regla que incluye las variables distintivas encontradas y en la ejecución del clasificador sobre los datos de prueba para demostrar su exactitud.

Las fases de este método (Figura 4.1) permiten que la detección de *malware* mediante el tráfico en la red sea muy eficiente; su mayor desventaja es la antigüedad de su *dataset*, pues fue realizado en 2014, por lo que no es posible conocer su comportamiento hoy, luego de la gran evolución que ha tenido el *malware* en los últimos años.

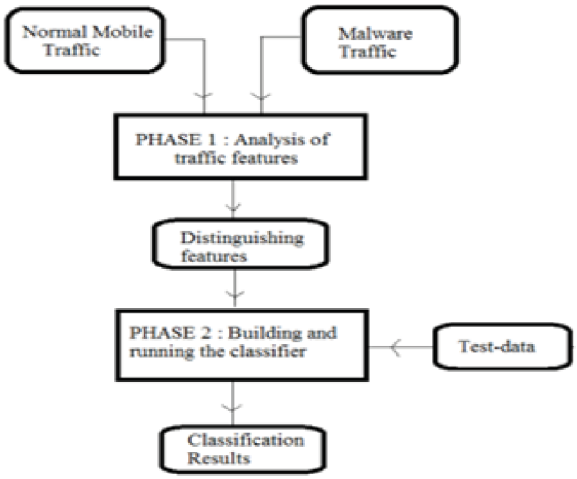

Figura 4.1. Fases del método utilizado

DroidAlarm: una herramienta de análisis estático multifuncional para el malware de escalado de privilegios en Android

Android contiene recursos sensibles a los que solo se puede acceder a través de las API correspondientes y estas solo se pueden invocar cuando el usuario tiene permisos autorizados en el modelo de permisos de Android. Sin embargo, una nueva amenaza llamada ataque de escalamiento de privilegios puede pasar por alto este mecanismo de seguridad; se presenta cuando una aplicación con menos permisos accede a recursos sensibles a través de las interfaces públicas de una aplicación con mayores privilegiada, lo que es especialmente útil para que el *malware* oculte funciones sensibles, dispersándolas en múltiples programas [7].

Yibing, Zhi, Bing y Li [7] exploran las técnicas de evolución del software malicioso de escalamiento de privilegios en muestras del Android *Malware* Genome Project, y han demostrado una gran eficacia frente a un conjunto de potentes herramientas antivirus proporcionadas por VirusTotal. Las relaciones de detección presentan una reducción diferente y diferenciada, comparada con una relación de detección media del 61 % antes de la transformación. Con el fin de conquistar este modelo de amenaza, se desarrolló una herramienta llamada DroidAlarm, para llevar a cabo la identificación de posibles fugas y de las rutas mediante el análisis estático en aplicaciones Android.

Un aspecto muy positivo de este trabajo es la reducción de falsos positivos (la clasificación errónea de software no malicioso como *malware*); su aspecto menos favorable es que al utilizar un método de análisis estático, no proporciona una metodología muy aplicable al análisis de tráfico en la red, el cual está más orientado al análisis dinámico.

Una primera mirada al tráfico de malware de Android en los primeros minutos

En este trabajo, Chen et al. [4] diseñan un esquema de monitoreo del comportamiento del tráfico de *malware* Android para datos de muestras en un entorno real de Internet, para lo cual capturan el tráfico de la red a partir de 5.560 *malware* en los primeros cinco minutos y analizan las principales composiciones de los datos. El método emplea un algoritmo automatizado de generación y recolección de tráfico, mediante él cual descubren que el tráfico HTTP y DNS representa más del 99 % del tráfico de la capa de aplicación

Los investigadores presentan un análisis de las características de red relacionadas: consulta DNS; longitud del paquete HTTP; relación entre la cantidad de tráfico de enlace de subida y bajada; petición HTTP; y característica de tráfico publicitario. Los resultados estadísticos evidencian que más del 70 % de los programas maliciosos generan tráfico malicioso en los primeros cinco minutos y que la consulta DNS y la solicitud HTTP se pueden utilizar para identificar el *malware*, sus tasas de detección alcanzan el 69,55 % y el 40,89 % respectivamente [4].

Pese a las características que proporcionan y a sus hallazgos, este trabajo solo tuvo en cuenta un *dataset* de aplicaciones maliciosas, lo que no permite una comparación de las características obtenidas frente a aplicaciones no maliciosas.

4.3. La investigación

Para el desarrollo del proyecto se empleó el modelo de ciclo de vida incremental, en el cual se hace énfasis en las demostraciones frecuentes del progreso, la verificación y validación del trabajo. Los requerimientos se asignan a varios elementos de la arquitectura de software y esta se divide en una secuencia priorizada de construcciones. Cada compilación añade nuevas capacidades al producto en crecimiento incremental, en cada iteración se obtiene una versión *i* del proyecto. El proceso de desarrollo finaliza cuando la versión N (la versión final) es verificada, validada, demostrada y aceptada por el cliente [8]. El desarrollo del proyecto se cubrió mediante cuatro fases: análisis e investigación, diseño, implementación, y pruebas y validación.

4.3.1 Fase 1. Análisis e investigación

En esta fase se realizó la investigación previa al desarrollo del proyecto, por lo que se constituye en la base de decisión de todas las herramientas, los ambientes de desarrollo, las tecnologías y los *dataset* de *malware* usados, entre otros. La investigación comenzó con la formulación del marco teórico y la investigación del estado del arte, en donde se analizaron los trabajos previos de otros investigadores y los *frameworks* propuestos por ellos.

Además, se identificaron posibles *datasets* para cumplir con el primer objetivo. Se seleccionaron 650 aplicaciones maliciosas provenientes del *dataset* del proyecto Drebin [9] y 530 legítimas, 80 % de las legítimas se tomó de un *dataset* de la Universidad de Columbia [10] y el 20 % restante de la página APKPure [11]. Las aplicaciones legítimas fueron verificadas con VirusTotal [12] para garantizar que no contuvieran algún tipo de *malware*.

Hecho esto, se dio inicio a la investigación de la metodología para el desarrollo del sistema propuesto. Aunque inicialmente se evaluaron varios *frameworks* creados para el análisis dinámico de aplicaciones Android (CuckooDroid, Androl4b, *Android Security Evaluation Framework* y *Mobile Security Framework*), todos ellos se descartaron por motivos de control, complejidad y limitación de recursos; se decidió entonces usar AVD (*Android Virtual Device*).

4.3.2 Fase 2. Diseño

Durante esta fase, en cada uno de los incrementos se le añadieron elementos al diagrama de *deployment* (Figura 4.2). Dado que el primer incremento estuvo

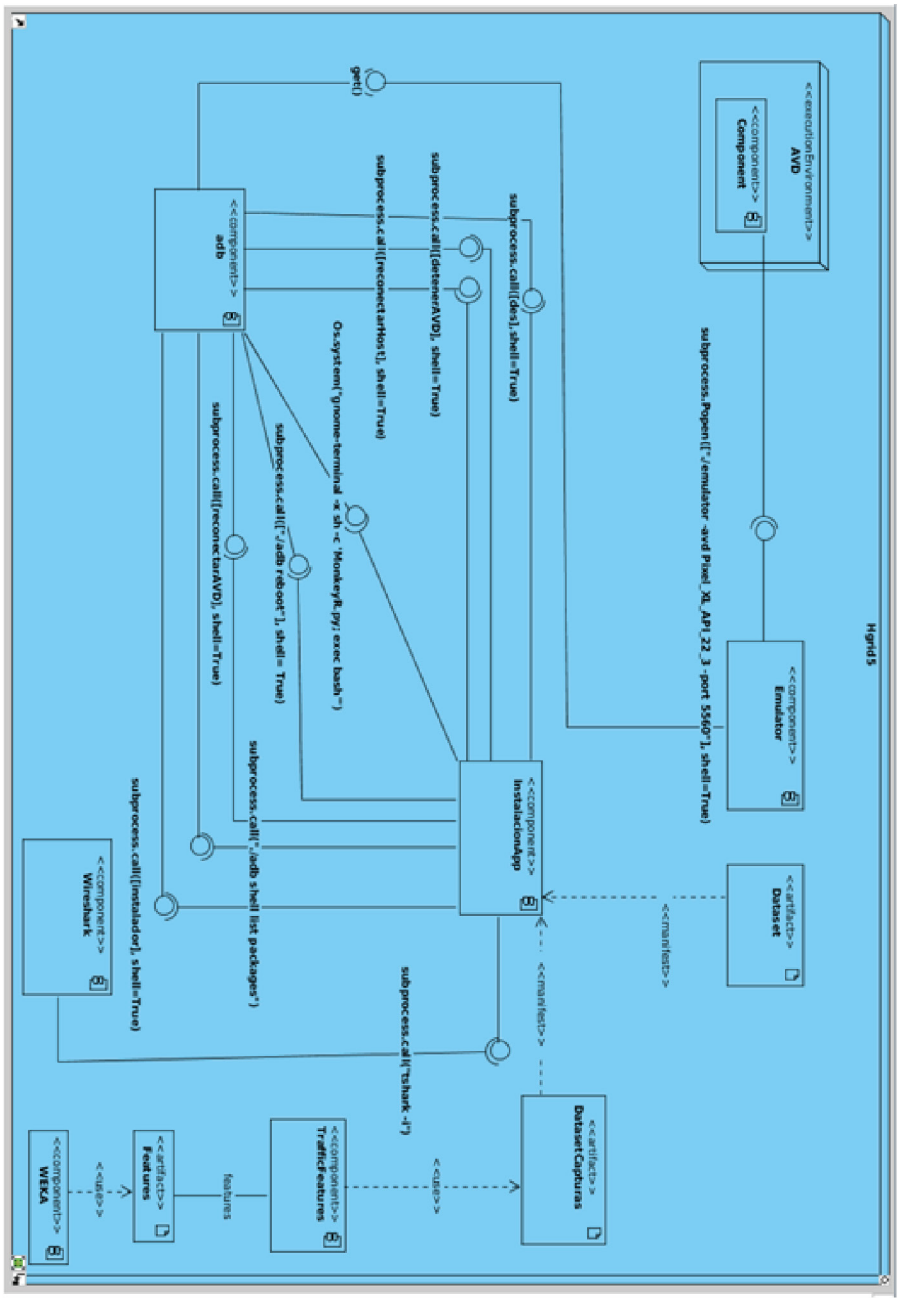

Figura 4.2. Diagrama de deployment

enfocado en la generación de tráfico, el diseño solo incluyó el dispositivo AVD y algunos módulos necesarios para dicho proceso; en el segundo incremento se añadieron varios componentes, entre ellos, el de captura de tráfico y el de simulación de acciones del usuario de la aplicación en ejecución; y en el tercer incremento, se incluyó el módulo de clasificación de aplicaciones.

4.3.3 Fase 3. Implementación

Esta fase incluye tres etapas: preparación del ambiente de ejecución de aplicaciones Android; generación de tráfico; y características y clasificación.

Para la preparación del ambiente, se usó la versión 3 de Android Studio, el IDE de Android, herramienta que contiene un módulo que permite utilizar un emulador Android para la gestión de aplicaciones (en desarrollo o descargadas). Una vez creado el emulador, se configuraron y realizaron pruebas con aplicaciones y se investigaron distintos tipos de comandos para realizar tareas por consola, usando Python v.2.7 y las librerías *os* y *subprocess*.

En la segunda etapa, para la generación de tráfico se utilizó un *script* creado en Python para realizar: la instalación, búsqueda de paquetes y desinstalación de aplicaciones; el encendido y apagado del emulador; la ejecución de la herramienta Monkey y la captura de tráfico mediante TShark.

Con base en la recomendación metodológica de Chen et al. [4]: el tiempo que permaneció una aplicación instalada e interactuando con Monkey fue de cinco minutos; y la instalación y posterior desinstalación de una aplicación, se hizo mediante un reinicio (*reboot*) del emulador. La Figura 4.3 muestra el proceso que se lleva a cabo para la generación de tráfico.

Una de las principales dificultades al instalar las aplicaciones fue el error de configuración "INSTALL_FAILED_NO_MATCHING_ABIS", que impidió la instalación de aproximadamente ciento cincuenta aplicaciones maliciosas y doce legítimas. Según algunas opiniones en foros y páginas como Stack Overflow, este error ocurre cuando se intenta instalar una aplicación que no es compatible con la arquitectura de un emulador. Existen muchas aplicaciones compiladas para ser usadas en arquitectura ARMv7, que si se intenta instalarlas en un emulador con arquitectura Intel (o el caso contrario) no funcionarán, es decir que la arquitectura de la compilación de las aplicaciones debe ser igual a la del emulador, pues de lo contrario, la aplicación no se podrá instalar. Este error implicó la reducción del *dataset* original en un 15 %.

```
Entrada: Un archivo con las rutas de los archivos APK a ser procesados
Resultado: Archivos de tráfico (PCAP)
mientras La línea actual del archivo no sea una cadena vacía :
    Instale el archivo APK
    Si la instalación es exitosa entonces:
            Reinicie el servicio del adb;
            Reinicie el emulador (reboot);
            Obtenga la información del paquete;
            Inicie la captura con tshark;
            Inicie en paralelo la ejecución de monkey:
            Desinstale la aplicación;
            Espere 30 segundos;
            Reinicie el emulador (reboot);
            Espere 10 segundos;
    Sino:
            Escriba un log conteniendo el nombre de la aplicación y el error producido;
            Incremente la cantidad de instalaciones fallidas en 1;

fin
```


```
Entrada: Un archivo con las rutas de los archivos APK a ser procesados
Resultado: Archivos de tráfico (PCAP)
mientras La línea actual del archivo no sea una cadena vacía :
    Instale el archivo APK
    Si la instalación es exitosa entonces:
            Reinicie el servicio del adb;
            Reinicie el emulador (reboot);
            Obtenga la información del paquete;
            Inicie la captura con tshark;
            Inicie en paralelo la ejecución de monkey:
            Desinstale la aplicación;
            Espere 30 segundos;
            Reinicie el emulador (reboot);
            Espere 10 segundos;
    Sino:
            Escriba un log conteniendo el nombre de la aplicación y el error producido;
            Incremente la cantidad de instalaciones fallidas en 1;

fin
```


Figura 4.3. Algoritmo de generación de tráfico

Como se indicó, en todo este proceso se utilizó el *dataset* del proyecto Drebin, que es libre para uso académico y contiene aplicaciones Android (APK) de tipo malicioso (*malware*) provenientes de distintas familias de *malware* (troyanos, *spyware*, *adware*, etc.). El proceso de generación y captura de tráfico tardó aproximadamente ocho minutos por aplicación y tuvo como resultado final 1.053 archivos pcap obtenidos con TShark (518 legítimos y 535 maliciosos).

En la etapa de características y clasificación, se creó un *script* en Python, el cual utiliza la librería Pyshark, para la extracción de las siguientes variables del tráfico web: cantidad de paquetes TCP; número de puertos distintos a los puertos TCP; número de IP externas; cantidad de *bytes* de la capa de aplicación; número de paquetes UDP; número de paquetes originados; número de paquetes recibidos; cantidad de *bytes* enviados por la aplicación; cantidad de *bytes* recibidos por la aplicación; número de paquetes HTTP; y número

de consultas al DNS. Algunos de los algoritmos usados para la extracción de características se basaron en el trabajo de Peña y Quintero [13]

A partir de un archivo CSV (*Comma-Separated Values*) generado por el *script*, se procedió a evaluar los árboles de decisión mediante la herramienta WEKA (*Waikato Environment for Knowledge Analysis*) [14], que es un entorno para la experimentación de análisis de datos, desarrollado en Java, que permite aplicar, analizar y evaluar las técnicas más relevantes de análisis de datos, principalmente las provenientes de *machine learning*, sobre cualquier conjunto de datos del usuario, para lo cual requiere únicamente que los datos a analizar se almacenen en formato ARFF (*Attribute-Relation File Format*). WEKA se distribuye como software libre y está constituido por una serie de paquetes de código abierto con diferentes técnicas de preprocesado, clasificación, agrupamiento, asociación y visualización, y de facilidades para su aplicación y el análisis de prestaciones, cuando son aplicadas a los datos de entrada seleccionados.

4.3.4 Fase 4. Pruebas y validación

Una vez terminado cada módulo del sistema, se realizaron pruebas tipo *smoke* test (prueba de humo), en las cuales se corrobora el correcto funcionamiento del módulo verificando su funcionalidad. Se realizaron dos pruebas: una para el módulo de generación y captura de tráfico, en donde se verificó la correcta instalación, interacción, captura de datos y desinstalación de la aplicación, utilizando software compatible y no compatible y asegurando el registro, en un archivo de *log*, de los errores que pudiesen surgir; otra para la extracción de características, en donde se compararon los datos tomados en las capturas con los archivos PCAP originales, para validar que los datos extraídos fueran consistentes con los de los archivos.

En la etapa de validación se pudo comprobar que los datos propios obtenidos eran muy distintos a los ofrecidos por Drebin, lo que podría explicarse con que existe la posibilidad de que en el proceso de generación de tráfico, en Drebin no se usaran herramientas de simulación de las acciones de un usuario en las aplicaciones, algo que si fue considerado en el proyecto actual, lo que evidentemente tiene una incidencia en la cantidad de tráfico recolectado. Asimismo, en esta fase se compararon los datos obtenidos en el proyecto actual, con los que reportan Urcuqui et al. [15], quienes desarrollaron un análisis de características tanto a nivel de red como de permisos en aplicaciones Android, tema que se amplía en la siguiente sección.

4.4. Resultados

4.4.1 Fases 1 a 6

La fase 1 corresponde a los requerimientos generales del sistema, al respecto, el sistema debe:

- permitir la clasificación de aplicaciones Android (APK) como maliciosas o legítimas (R1);
- generar, capturar y almacenar datos de tráfico Web de aplicaciones legítimas y maliciosas provenientes de un *dataset* de prueba (R2);
- permitir la instalación y desinstalación de aplicaciones Android (R3);
- contar con un módulo de simulación para las acciones realizadas por un usuario (R4);
- permitir la automatización del R2 (R4); y
- procesar datos de captura de tráfico (PCAP) y realizar la extracción de características en un archivo CSV (R5).

La fase 2 corresponde al particionamiento final del sistema (análisis Dorfman paso 1), como se aprecia en la Figura 4.4, el subsistema emulador es la entidad que representa el entorno emulado en donde las aplicaciones van a

Figura 4.4. Particionamiento del sistema propuesto

ser ejecutadas; el subsistema aplicaciones se refiere al *dataset* de aplicaciones tanto legítimas como maliciosas; el subsistema de captura de tráfico hace referencia al proceso de captura y generación de tráfico y a los archivos que allí se generan; los subsistemas que se desprenden de "archivos de ejecución" son los responsables de la automatización de tareas a la hora de generar tráfico y de obtener las características deseadas de dicho tráfico; por último, el subsistema de árboles de decisión que se deriva de "clasificación", se encarga de la clasificación de las aplicaciones como maliciosas o legítimas.

La fase 3 se refiere a la asignación de los requerimientos citados en la fase 1, a los subsistemas que se presentan en la fase 2 (análisis Dorfman paso 2), la cual se presenta en la Tabla 4.2.

Tabla 4.2. Asociación de los requerimientos (fase 1) a los subsistemas (fase 2)

Subsistema	R1	R2	R3	R4	R5
Emulador			X	X	
Aplicaciones			X		
Captura de tráfico		X			
Instalación de aplicaciones				X	
Extracción de características					X
Árboles de decisión	X				

La fase cuatro corresponde a la subespecificación (análisis Dorfman, paso 3):

- R1_arbolDes_1: el sistema debe evaluar cuatro tipos de árbol de decisión, utilizando la herramienta WEKA;
- R2_capturaTrafico1: el sistema debe capturar el tráfico generado por cada una de las aplicaciones (legítimas y maliciosas) usando filtros por la IP del dispositivo;
- R2_captura_trafico2: el sistema debe almacenar los datos de las capturas de tráfico, al igual que los *logs* de errores que puedan surgir;
- R3_emulador1: el sistema debe usar la consola de comandos para la instalación y desinstalación no visual (interfaz gráfica) de las aplicaciones;
- R3_aplicaciones1: el sistema debe descartar las aplicaciones que no sean compatibles con el emulador y dejar un registro de dichas aplicaciones en un archivo de texto plano;

- R4_emulador1: el sistema, por medio de la consola de comandos Linux, debe permitir el encendido, apagado, reinicio (*reboot*) y suspensión de los servicios del emulador Android;
- R4_instalacionAplicaciones1: el *script* de automatización debe automatizar los procesos de instalación y desinstalación de aplicaciones, generación de tráfico, gestión del emulador y del simulador de movimientos;
- R4_instalacionAplicaciones2: el *script* debe registrar los errores surgidos en cualquier proceso en un archivo de *log* en formato txt;
- R5_extraccionCaracteristicas1: el sistema debe contar con un módulo de extracción de las características definidas en la fase de investigación y generar un archivo CSV y un archivo txt de *log* para los errores surgidos.

La fase 5 corresponde al diseño, en la Figura 4.5 se presenta un panorama general del sistema propuesto y muestra el proceso desde cuando el archivo APK se instala en el entorno emulado, hasta cuando este llega a la fase de clasificación.

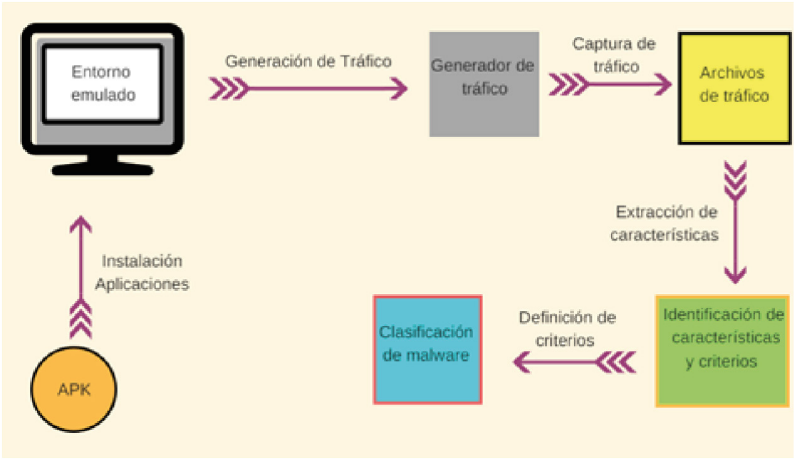

Figura 4.5. Vista general del sistema propuesto

La fase 6 corresponde a la implementación, sus resultados se pueden consultar en el repositorio de Github andres-180 / Traffic algorithm [16], en donde se encuentran los archivos de generación, captura y extracción de características.

4.4.2 Fase 7. Validación

Para este proceso se tomaron los datos del estudio realizado por Urcuqui et al. [15], para comparar: el análisis descriptivo y el análisis de los métodos de clasificación.

Análisis descriptivo de los resultados

En la Tabla 4.3 se relacionan los promedios de las características obtenidas de las aplicaciones (maliciosas y legítimas) del proyecto actual. Con base en esos resultados, se puede concluir que al igual que en el experimento realizado por Arora, Garg y Peddoju [5], en las capturas de tráfico se evidencia que el promedio del tamaño de los paquetes enviados y recibidos (en el caso del *malware*) es menor que las aplicaciones legítimas, lo mismo sucede con los *bytes* enviados y recibidos.

Tabla 4.3. Resultados del análisis estadístico del proyecto

Características	Promedio legítimas	Promedio maliciosas
tcp_packets	779.1428571429	244.9082397004
disp_port_TCP	568.4749034749	214.3501872659
external_ips	14.167953668	9.8951310861
app_bytes	145881.932432432	77397.6779026217
udp_packets	24.7895752896	2.2752808989
source_app_packets	825.2316602317	258.6441947566
remote_app_packets	798.6370656371	230.5280898876
source_app_bytes	1314641.34749035	240620.966292135
remote_app_bytes	147629.525096525	78317.1741573034
HTTP_packets	11.5347490347	1.0711610487
dns_query_times	21.6332046332	11.456928836

Una de las características más notorias fue el envío de paquetes TCP, ya que en las aplicaciones legítimas esta cantidad fue más de tres veces mayor al de las aplicaciones maliciosas (779 versus 244). Pese a que en las aplicaciones legítimas la cantidad de llamados o consultas DNS es casi el doble que en las maliciosas, al hacer una inspección más detallada se puede observar que

para cada aplicación, la cantidad de consultas fue un factor muy variable. En cuanto al protocolo HTTP, hubo una diferencia notoria a nivel individual entre las aplicaciones maliciosas y las legítimas, ya que si bien a primera vista la tabla muestra que el promedio de las aplicaciones maliciosas es once veces menor, una observación más detallada permite notar que solo en nueve casos el protocolo HTTP superó los cuatro paquetes, mientras que en el caso de las aplicaciones legítimas este protocolo fue usado en 108 paquetes. Es importante mencionar que existen otros protocolos, como el HTTPS, que no fueron estudiados debido al alcance del proyecto.

Continuando con los resultados de la validación, en la Tabla 4.4 se pueden observar los datos obtenidos por Urcuqui et al. [15] en el estudio objetivo de la comparación del proyecto. Como se puede notar, en este estudio se generó una menor cantidad de tráfico, lo que puede haber sido causado porque en él no se emplearon herramientas de simulación de movimientos del usuario. Cabe resaltar que la característica relacionada con la cantidad de paquetes HTTP no aparece en la Tabla 4.4 porque dicha característica es diferenciadora, ya que no se tuvo en cuenta.

Por último, al realizar un análisis estadístico, se observa que aquí también se cumplen algunos hallazgos del proyecto actual, como la cantidad de tráfico generado por las aplicaciones maliciosas versus el generado por las legítimas, la cantidad de paquetes TCP y los *bytes* enviados y recibidos, entre otras.

Tabla 4.4. Características obtenidas por Urcuqui et al. [15]

	Promedio legítimas	Promedio maliciosas
tcp_packets	197.565476	72.71792423
disp_port_TCP	11.4005102	2.253422477
external_ips	3.33971088	1.863100923
app_bytes	21082.8236	9745.983445
udp_packets	0.08737245	0.010824578
source_app_packets	202.772109	78.24068768
remote_app_packets	259.689626	97.3865011
source_app_bytes	274748.518	94291.5476
remote_app_bytes	21471.4581	10110.18434
dns_query_times	5.03954082	4.688315823

Análisis de resultados usando el método de clasificación

Con el *dataset* compuesto por 535 aplicaciones maliciosas y 518 aplicaciones legítimas, se procedió con la evaluación de los cuatro algoritmos de clasificación (J48, LMT, *random forest* y *random tree*), utilizando parámetros generales, tales como la validación cruzada con un número de iteraciones k = 12. En cada iteración, los datos de muestra se dividieron en un subconjunto de entrenamiento y otro de datos de prueba (más pequeño que el de entrenamiento). Los resultados de este proceso se presentan en la Tabla 4.5.

Tabla 4.5. Desempeño individual de los cuatro clasificadores

Algoritmo	Precision		Recall		F-measure		FP rate		CCI (%)	Kappa
	Si	No	Si	No	Si	No	Si	No		
J48	0.901	0.919	0.923	0.896	0.912	0.907	0.104	0.077	90.97	0.8193
Random forest	0.926	0.935	0.938	0.923	0.932	0.929	0.077	0.062	93.06	0.8612
LMT	0.888	0.898	0.903	0.882	0.895	0.890	0.118	0.097	89.26	0.7851
Random tree	0.872	0.879	0.884	0.867	0.878	0.873	0.133	0.116	87.55	0.7508

Como se observa, el modelo con mejor desempeño es *random forest*, el cual obtuvo: el 93,06 % de CCI (*Correctly Classified Instances*); una capacidad de 92,6 % en identificación de comunicaciones legítimas; un indicador Kappa (coherencia o concordancia entre las variables) de 0,86; y una baja tasa de falsos positivos (FPR). Cabe resaltar que los algoritmos tenían como condición inicial en el árbol de decisión distintas características. Por ejemplo, el algoritmo J48 tenía como primera condición la cantidad de IP externas, al igual que el algoritmo LMT, mientras que en el caso de *random tree*, la característica principal fue el número de consultas al DNS.

En cuanto a la comparación con el estudio anterior, el método de *random forest* fue el que mejor desempeño tuvo, con una precisión de 0,93 para los casos de aplicaciones legítimas, un valor muy cercano al del proyecto actual. Sin embargo, el kappa del estudio anterior es de solo 0,74, un valor menor al de este proyecto, lo que indica una mayor concordancia entre las variables del proyecto actual, ya que tiene un kappa más cercano a 1. Por último, se puede concluir con respecto al estudio realizado y a este proyecto, que *random forest* es

el mejor modelo de predicción a nivel de red, y aunque el estudio anterior tiene muy buenos resultados, el proyecto propuesto le añade algunas mejoras que han ayudado a obtener algunos resultados aún más interesantes, tales como la cantidad de tráfico y la característica del tráfico HTTP.

4.5. Conclusiones y recomendaciones

Con base en los resultados del análisis descriptivo, se puede concluir:

- las aplicaciones maliciosas presentan un porcentaje muy bajo de tráfico en las características seleccionadas, a diferencia de las aplicaciones legítimas, la explicación del porque puede ser la cantidad de información que debe ser consultada por algunas de estas aplicaciones, que es muy recurrente, como por ejemplo en las aplicaciones sociales, los juegos y las aplicaciones de reproducción de contenido;
- es posible utilizar algoritmos de clasificación para la detección de aplicaciones maliciosas basados en algunas características del tráfico Web, los resultados muestran que los mejores modelos fueron *random forest*, seguido de J48.

En cuanto al trabajo a futuro, aún se pueden mejorar muchos aspectos, tales como:

- el seguimiento de los sitios a los cuales las aplicaciones maliciosas acceden, ya que es de mucho interés conocer los sitios en Internet a los cuales se hacen las conexiones, para así obtener otro tipo de resultados interesantes; y
- la emulación de las acciones del usuario que se utilizaron para cada aplicación, ya que aun cuando Monkey hace un buen trabajo, en algunas ocasiones no fue muy efectivo, esto disminuye la cantidad de tráfico generado por algunas aplicaciones, en especial las que requieren un *login* o aquellas que requieren datos reales

Por último, dado que existen estudios, como el de Alzaylaee et al. [17], que demuestran como algunas aplicaciones pueden detectar entornos emulados y restringir sus actividades, haciendo que trabajar en ellos sea muy distinto a hacerlo en un entorno real, se propone llevar a cabo el mismo estudio, pero con dispositivos.

Los recursos (documentos, gráficos, códigos, etc.) obtenidos durante la ejecución del proyecto están disponibles en [18].

Referencias

[1] M. Mena. (2021, ago. 30). *Android e iOS dominan el mercado de los smartphones*. Disponible: https://es.statista.com/

[2] Infografía: ¿Cuánto ganan los 'hackers'? Kaspersky explica el lucro de la piratería. (2014, ago. 30). *RT*. Disponible: https://actualidad.rt.com/actualidad/view/138751-ganan-hackers-expertos-kaspersky-explican

[3] K. Tam, A. Feizollah, B. Anuar, R. Salleh, y L. Cavallaro, "The evolution of Android malware and Android analysis techniques. A*CM Computer Surveys*, vol. 49, no. 76, pp. 1-41, 2017. https://doi.org/10.1145/3017427

[4] Z. Chen, H. Han, Q. Yan, B. Yang, L. Peng, L. Zhang, y J. Li, "A first look at Android malware traffic in first few minutes," en *Proceedings - 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2015*. https://doi.org/10.1109/Trustcom.2015.376. P 8

[5] A. Arora, S. Garg, y S. K. Peddoju, "Malware detection using network traffic analysis in Android based mobile devices," en *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*, 2014, pp. 66-71, https://doi.org/https://doi.org/10.1109/NGMAST.2014.57

[6] J. Malik, y R. Kaushal. (2016). *CREDROID: Android malware detection by network traffic analysis*. Disponible: https://www.researchgate.net/publication/304995756_CREDROID_Android_malware_detection_by_network_traffic_analysis

[7] Z. Yibing, X. Zhi, M. Bing, y X. Li, (2013), *DroidAlarm: An all-sided static analysis tool for Android privilege-escalation malware*. Disponible: https://www.researchgate.net/publication/262403167_DroidAlarm_An_all-sided_static_analysis_tool_for_Android_privilege-escalation_malware

[8] R.E. Fairley, *Managing and leading software projects*, Wiley-IEEE Computer Society, 2009.

[9] Technische Universität Braunschweig, (2016), *The Drebin dataset*. Disponible: https://www.sec.cs.tu-bs.de/~danarp/drebin/

[10] Columbia University, Department of Computer Science. (2015). *PlayDrone: a measurement study of Google Play*. Disponible: http://systems.cs.columbia.edu/projects/playdrone/

[11] *APKPure* [portal]. (2018), Disponible: https://apkpure.com/es/

[12] *VirusTotal* [portal]. (2018). Disponible: https://www.virustotal.com/#/home/upload

[13] M. Peña, y J. Quintero, "Sistema open source para la detección de páginas Web maliciosas", proyecto de grado, Cali, Colombia: Universidad Icesi, 2017.

[14] Universidad de Waikato. (2018). *Weka 3: Data mining software in Java* [proyecto open source]. Disponible: https://www.cs.waikato.ac.nz/ml/weka/

[15] C.C. Urcuqui, J. Delgado, A. Pérez, A. Navarro, y J. Díaz, (2018), "Features to detect Android malicious applications" presentado en IEEE Colombian Conference on Communications and Computing (Colcom), 2018).

[16] Traffic-algorithm. (2018). *Andres-180* [repositorio de GitHub]. Disponible: https://github.com/andres-180/Traffic-algorithm

[17] M. Alzaylaee, S. Y. Yerima, y S. Sezer, "Emulator vs real phone: Android malware detection using machine learning," en *Proceedings of the 3rd ACM on International Workshop on security and Privacy Analytics, 2017*, pp. 65-72, ACM.

[18] C.C Urcuqui. (2022). Sistema de análisis de tráfico web para la detección de malware (PDG). *i2tResearch* [repositorio GitHub]. Disponible: https://github.com/i2tResearch/Ciberseguridad_web/tree/master/Android/Sistema%20de%20analisis%20de%20trafico%20web%20para%20la%20deteccion%20de%20malware%20(PDG)

Capítulo 5
Detección de cryptojacking a través del tráfico de red usando técnicas de machine learning

Steven Bernal, Christian Camilo Urcuqui, Andrés Navarro

5.1. Introducción

La adopción de las criptomonedas es más significativa en el entorno financiero porque representa una alternativa al dinero convencional y a los sistemas de pago tradicional, al permitir el intercambio de bienes y servicios y la realización de transferencias de fondos e inversiones, sin la participación de un intermediario financiero. Una gran cantidad de estas criptomonedas aplican algoritmos complejos de criptografía, para cifrar sus respectivas transacciones y así reservar la privacidad de sus emisores y receptores y hacer compleja la trazabilidad de sus movimientos, lo que ha incentivado su uso en actividades delictivas tales como: la evasión de impuestos, el lavado de activos y la financiación del terrorismo [1].

Existen dos formas de adquirir una criptomoneda: la primera es comprarla en un mercado financiero de criptodivisas; la segunda, obtenerla por medio de la minería. El proceso de minería consiste en validar una transacción de criptomonedas entre dos usuarios, en una conexión *pear to pear*, por medio de la resolución de un problema matemático; este proceso lo realizan otros usuarios, conocidos como mineros, quienes utilizan sus dispositivos computacionales para tal fin. Una vez validada la transacción se crea un bloque con información de los movimientos relacionados a las criptomonedas transferidas, que luego se verifica por el resto de usuarios, para que el bloque sea anexado al sistema de *blockchain* [2]. El minero que resuelva el problema obtiene una nueva criptomoneda y a su vez, el problema de validación incrementa su dificultad [3]. Cabe resaltar que la recompensa, así como la dificultad, puede variar, dependiendo del tipo de criptomoneda.

El proceso para extraer criptomonedas requiere del uso intensivo de unidades de procesamiento, lo que causa un incremento en el consumo de electricidad y hace inviable la minería para un solo dispositivo. En 2018, extraer un *bitcoin* por medio de una computadora sin modificaciones de herramientas mineras podría tomar cerca de 425 años [4]. Por eso, algunos mineros optaron por el trabajo colectivo a través de piscinas mineras, donde la carga de trabajo se distribuye entre los participantes; así, si un integrante obtiene la solución del problema, la recompensa se distribuye entre todos los mineros de la piscina, con base en su aporte de recursos computacionales [5].

Personas con dudosa moralidad iniciaron a minar criptoactivos con recursos computacionales de otras personas, sin autorización. Esta actividad se denominó *cryptojacking*. Inicialmente los ataques de esta índole se hacían con ejecutables; después el enfoque cambio hacia la minería a través del navegador, donde los atacantes o *criptojackers* empezaron a usar procesos benignos y autorizados del equipo afectado, con *scripts* de minería que se cargan en el navegador; y en los últimos años, los *criptojackers* están atacando la infraestructura y los servidores de proveedores de servicios de computación de la nube [6].

El *cryptojacking* se hizo popular durante 2017, con el uso indebido del *script* de Coinhive, una firma de minería de navegador especializada en extraer "Monero", una criptomoneda que utiliza el protocolo cryptoNote y aprovecha la firma de anillo enlazable (Ring CT), para proteger la identidad del emisor de las transacciones [7]. Durante el funcionamiento de Coinhive, se reportaban hasta diez millones de usuarios víctimas del *script* de esta compañía por mes, antes de su cierre, en marzo de 2019 [8].

En la actualidad, el *cryptojacking* afronta un futuro incierto debido al cierre de Coinhive, el precio volátil de las criptomonedas y la complejidad para extraerlas. Pese a lo anterior, se han detectado versiones nuevas de los mineros conocidos, además de *scripts* únicos, que demostrarían que los cibercriminales están especializándose en la extracción de criptomonedas. Un estudio realizado por investigadores de las universidades de Cincinnati y Lakehead, demostró que el 99 % de los sitios web analizados por la herramienta de detección minera CMTracker en 2018, ya no usan código minero, solo el 1 % aún ejecuta código de minería. De este grupo se detectaron ocho *scripts* únicos que permitieron rastrear 632 sitios web de *cryptojacking* únicos. Los investigadores concluyeron, a partir de esos resultados, que el *cryptojacking* no "murió" con el cierre de Coinhive [8].

Cabe resaltar que la razón de existir del *cryptojacking* es la popularidad de las criptomonedas, cuyo mercado está creciendo por diversas razones, especialmente por: su aceptación en diferentes países, como China, que está regulando una moneda virtual para competir con el dólar [9]; su acogida en diferentes empresas, como PayPal, Facebook, Microsoft, Shopify, JPMorgan y Tesla, que las usan o las han incluido en sus ecosistemas comerciales [10]; y la recuperación o incremento de su precio producto de la pandemia COVID-19.

El problema del *cryptojacking* persiste porque es un ciberamenaza multiplataforma, en su gran mayoría consiste en un ataque sin *malware*, lo que significa que no requiere usar código malicioso, sino que basta con visitar un sitio corrupto, sin descargar un archivo, para comprometer el sistema [11]. Los *criptojackers* también implementan ataques de *living off the land*, donde usan herramientas instaladas por los sistemas nativos, para ejecutar código minero y explotar vulnerabilidades, con directivas *bash* de Linux o *powershell* de Windows [6], sobre todo para atacar la infraestructura de servicios de computación en la nube.

El problema del *cryptojacking* persiste, pese a que existen diversos métodos para detectar mineros en la red. El primer método consiste en detectar la dirección IP (*Internet Protocol*) de mineros en el tráfico de red [12], sin embargo, este método presenta algunos inconvenientes, ya que los servidores de minería pueden cambiar la dirección IP del sitio donde están alojados. El segundo mecanismo es la inspección profunda de paquetes (DPI, *Deep Packet Inspection*) para detectar los protocolos utilizados por mineros; este mecanismo representa mucha carga computacional, ya que todos los paquetes deben ser analizados para detectar posible actividad minera. El tercer método es la solicitud del sistema de nombres de dominio (DNS, *Domain Name System*), en él se busca detectar las direcciones que pueden estar relacionadas con minería; este método no siempre da indicios correctos de que exista actividad minera en la red, ya que algún usuario puede tan solo estar visitando sitios que ofrecen servicios de minería [12]; además, las direcciones pueden ser modificados por el atacante. También se tiene que considerar el constante crecimiento de las redes, el cual se estima en 37 % para 2023, que hace cada vez más complejo analizar el tráfico de red [13].

La realización de este proyecto se justifica en la importancia de tener una solución basada en firmas o perfiles, que permita la detección de *malware* conocido. El problema con estas soluciones radica en su ineficiencia ante

nuevas amenazas [14], sobre todo aquellas que no requieren de archivos para comprometer la máquina de una víctima, como el *cryptojacking*. De esta manera, no se puede capturar y asociar una firma a la actividad del *malware* minero, el cual se aprovecha de procesos benignos y autorizados por el sistema para extraer criptomonedas [11]. Por lo anterior, es necesario realizar un análisis dinámico que permita determinar las trasmisiones de información en la red entre el equipo que procesa los datos de los mineros (puede ser un servidor, como la computadora del *cryptojacker*) y la víctima. Una vez procesada la información, se le aplica analítica de datos para evaluar un modelo que tenga la capacidad de detectar amenazas mineras en tiempo real.

El reconocimiento de patrones y características en común de la actividad minera de criptomonedas permitirá la identificación oportuna del *cryptojacking*, de esta manera un administrador de red podrá eliminar la actividad minera de los equipos afectados, previniendo que los dispositivos tecnológicos degraden su hardware y desperdicien energía eléctrica, evitando así costos por el remplazo de equipos deficientes, incremento en la factura de energía por causa de la actividad minera, a la vez que reduce la huella ambiental derivada del consumo inútil de electricidad.

El *cryptojacking* es un ataque que normalmente se ejecuta en segundo plano y no necesariamente requiere de archivos binarios para infectar a un equipo, aunque también puede usar *malware*; presenta variabilidad en los vectores de ataques, utiliza llamadas al sistema usando procesos autorizados y es multiplataforma. Esta amenaza suele evadir los mecanismos establecidos y genera un tráfico que puede pasar desapercibido en una red grande.

El *cryptojacking* parte del consumo de recursos de una víctima, la cual debe recibir y enviar información a un tercero (cibercriminal u otro agente malicioso); toda transmisión viaja a través de la red, es decir, que si ocurrió un acto malicioso, los registros deben tener la historia del evento, independientemente del estado del minero (ya identificado o no) y del tráfico que exista en una red. El problema abordado por este proyecto fue identificar la información y el tráfico de red generado en la comunicación entre los procesos del equipo minero del atacante y la máquina víctima del minero. Su objetivo general es desarrollar un modelo de *machine learning* que tenga la capacidad para detectar actividad de *cryptojacking* a través de registros de red, para cuyo logro ha definido como objetivos específicos: encontrar patrones en tráfico minero; evaluar diferentes modelos de clasificación que tengan la capacidad de predecir actividad minera;

crear un *dataset* de *pcaps* de tráfico de *cryptojacking*; crear un *dataset* de ventanas de tiempo de actividad de *cryptojacking*; y analizar variables de hardware con actividad minera.

5.2. Estado del arte

La revisión realizada incluyó cinco proyectos que abordan parcial o completamente lo propuesto en el proyecto reportado en este documento: *Detection of bitcoin miners from network measurements*; *How you get shot in the back: a systematical study about cryptojacking in the real world; The browsers strike back: countering cryptojacking and parasitic miners on the web; Detecting cryptocurrency miners with NetFlow/IPFIX network measurements; Dine and dash: static, dynamic, and economic analysis of in-browser cryptojacking.*

Detection of bitcoin miners from network measurements

En este proyecto, Muñoz et al. [12] presentan un método basado en el aprendizaje automático que usa el algoritmo J48 para la detección de mineros a partir de mediciones de Netflow/IPFIX. Logra una precisión cercana a la inspección de paquetes y lo hace ahorrando recursos considerables, porque no inspecciona todo el contenido del paquete. El proyecto se realizó en un ambiente controlado para simular actividad de *cryptojacking*, en el que se usó Netflow v5 y dos computadores, uno como servidor de minería, otro como equipo minado, además del siguiente software: Softflowd, como exportador; nfcapd como recopilador y nfdump para transformar los archivos nfcapd en datos legibles.

Durante su ejecución, se capturó tráfico con el fin de identificar el protocolo minero Stratum y los métodos que utiliza para permitir la comunicación entre el servidor-cliente. Los investigadores determinaron que el servidor transmite datos a un ritmo constante, ya que sigue comunicando los datos necesarios para extraer la criptomoneda, mientras que el equipo afectado envía poca información, normalmente durante el inicio de la conexión o cuando comparte recursos con otro *malware* minero. También se identificaron variantes del protocolo Stratum, que usan variaciones de los métodos de comunicación. El uso de Netflow permitió extraer datos que podrían ser relevantes para identificar estratos de tráfico, como: el tiempo de inicio y finalización de la conexión, el número de paquetes, el número de *bytes*, los protocolos y las banderas.

Asimismo, se pudo determinar que el tráfico generado por los mineros de las criptomonedas podría llegar a ser menor al tráfico generado por un usuario que ingresa a sitios de juegos y películas. También se evidenció que no hay un tráfico homogéneo cuando se extraen diversas criptomonedas y que existe el riesgo de tener tráfico encriptado, imposible de identificar.

Los investigadores usaron cuatro algoritmos de clasificación: árboles de decisión, SVM, *naive* Bayes y C 4.5. Al probarlos en un entorno real, SVM y *naive* Bayes tuvieron un pésimo rendimiento, mientras que aquellos basados en árboles de decisión evidenciaron el mejor rendimiento, con una *accuracy* de 99 % y una precisión de 98 %, así como la implementación del algoritmo J48, dentro de un conjunto de 1.795.394 datos.

How you get shot in the back: a systematical study about cryptojacking in the real world

En este proyecto, Hong et al. [15] se enfocaron en la naturaleza del *criptojacking* y en componentes de la carga de trabajo de minería: computación regular, repetida y basada en *hash*. Para su desarrollo, crearon CMTracker, un detector basado en el comportamiento, con dos perfiladores de tiempo de ejecución para rastrear de forma automática *scripts* de minería y sus dominios relacionados: el primero, basado en *hash*, aprovecha la naturaleza de un sistema de prueba de trabajo, para la detección de funciones de *hash* de bajo nivel; y el segundo, basado en estructuras de pilas de llamadas de *scripts* de minería, registra pilas de tiempo de ejecución e identifica las páginas de minería buscando el punto de acceso de los contextos de llamadas.

El generador de perfiles basados en *hash* está centrado en funciones de *hash* de bajo nivel. Se usaron nueve interfaces de biblioteca de *hash* accesibles y comunes, que se identifican mediante un conjunto de firmas fijas de múltiples criptomonedas de código abierto o servicios de minería comercial (por ejemplo, Cryptonight Hash, Sha256 y Crypto), y después se calculó el tiempo acumulado de los sitios web que dedicaron al *hash*, para identificar si una página web está minando, considerando que los sitios web normales, por lo general dedican muy poco tiempo a procesar funciones de *hash*, mientras que los *scripts* de minería de criptomonedas le dedican la mayor parte del tiempo.

El generador de perfiles basado en estructuras de pilas se usa para la detección de patrones repetidos, revelados por la pila de ejecución de sitios mineros, ya que rara vez un sitio web repite la misma pila de llamadas durante más del 5,60

% del tiempo de ejecución; dado que la minería de criptomonedas es pesada, para evitar cualquier cosa notable por el usuario, la mayoría de las tareas de minería no se realiza en el hilo principal al cargar la página web, sino en uno o varios hilos dedicados.

Entre los principales sitios web de Alexa de 100K, CMTracker identificó 868 dominios que contenían *cryptojacking* y detectó 1.902 dominios en *links* externos, 53,9 % de los cuales no se hubieran podido detectar usando listas negras. Se estima que a partir de estos 2.770 sitios, se afecta a diez millones de usuarios web por mes.

Primero, los investigadores recolectaron 853.936 muestras de páginas web como conjunto de datos y aprovecharon la interfaz para registrar y perfilar las páginas visitadas por muestreo de pila; luego detectaron páginas afectadas por la amenaza minera usando dos perfiladores basados en comportamiento; después identificaron qué sitios correspondían a minería legítima; y por último detectaron las páginas con presencia de actividad de *cryptojacking*.

Los investigadores determinaron que las cargas de trabajo de las muestras de *criptojacking* costarían un promedio de 278K kWh de energía eléctrica por día, cifra que equivale al consumo de energía de una pequeña población de 9.300 personas. Para realizar la estimación anterior se tuvo en cuenta la cantidad de visitantes de cada dominio identificado, el tiempo promedio que permanece un visitante y la potencia de la CPU disponible para extraer en los navegadores (ecuación 5.1).

$$Energía = Visitantes \times Duración \times Potencia\ de\ la\ CPU \qquad (5.1)$$

The browsers strike back: countering cryptojacking and parasitic miners on the web

En este proyecto de investigación, Tahir et al. [16] exploran el *cryptojacking* en navegadores donde los mineros lo implementan en secreto, dentro del código de navegador, sin el conocimiento del usuario. Analizaron 50.000 sitios web de la lista de Alexa y encontraron un porcentaje notable de sitios que practican *cryptojacking* a menudo, usando código ofuscado. Los investigadores afirman que complementos, como NoMiner, no pueden mostrar instancias ocultas, y por ello proponen una solución de aprendizaje automático basado en el perfil asistido por hardware de código del navegador en tiempo real y

una microarquitectura de grano fino que permite calcular las aplicaciones de minería con 99 % de precisión e incluso conocer si el código de minería ha sido ofuscado o encriptado. Se creó además una extensión de navegador que aplica todo el conocimiento anterior con una sobrecarga insignificante en la máquina del usuario.

Para desarrollar el producto se recopiló un conjunto de datos de sitios web y se analizó cada uno para encontrar mineros ocultos; se identificaron las principales categorías de sitios web (transmisión de video, sitios para adultos, *torrents*, etc.), que son predominantemente abusados por *cryptojackers*. Aprovechando los desarrollos de hardware de los últimos años, se propició el uso de contadores de rendimientos de hardware (*Hardware Performance Counters*, HPC) para realizar actividades en el navegador y crear un clasificador de bosque aleatorio con una precisión del 99,35 %. El esquema de monitoreo está basado en el algoritmo de PoW (*Proof of Work*) de una moneda, por lo que detecta la actividad minera independientemente de la ofuscación o de las técnicas evasivas usadas.

Los HPC son registros internos de un procesador que representan el estado del sistema en un momento dado, sus valores resaltan las características de los programas que ejecuta el sistema. Estos contadores se pueden sondear con bastante rapidez y proporcionar una idea de los comportamientos de los programas que se ejecutan a nivel de microarquitectura.

El enfoque de este proyecto es ejecutar la muestra de sitios web y registrar los valores de los HPC, para luego clasificarlos como comportamiento normal del usuario o actividad minera. Para ello: el sitio web se ejecuta durante un cierto tiempo para cargar y representar el DOM (*Document Object Model*) HTML (*Hyper Text Markup Language*); luego se sondea, registra y marca la totalidad de HPC del sistema, para cada sitio web; y finalmente los datos se pasan a un clasificador de aprendizaje automático para su predicción. Este algoritmo, denominado *random forest*, funciona al construir un conjunto de árboles de decisión y decidir el resultado final con base en la votación del conjunto. Se dividió el conjunto en 80 % de entrenamiento y 20 % de prueba, para después hacer diez veces una validación cruzada (*cross validation*). Este proyecto logró una precisión del 99,35 % en el conjunto de prueba, con tasas de falsos positivos cercana a cero y de verdaderos positivos del orden de 100. El proyecto presentó entonces un enfoque de aprendizaje automático para marcar y mitigar actividades mineras secretas y ocultas basado en el uso de los HPS, y logró su identificación incluso cuando estás utilizaban técnicas de ofuscación.

Detecting cryptocurrency miners with NetFlow/IPFIX network measurements

En esta ocasión, Muñoz et al. [12] presentan un método basado en aprendizaje automático capaz de detectar criptomonedas utilizando mediciones de red NetFlow / IPFIX. Dado que el método no requiere inspeccionar la carga útil de los paquetes, logran una detección de mineros rentable, similar a las técnicas basadas en DPI.

Los investigadores identificaron que el protocolo Stratum es el responsable de permitir la comunicación entre el software y los servidores de minería. Sin embargo, precisan que aun cuando Stratum es el protocolo más utilizado para minar, existen otros protocolos para ese fin, como getblocktemplate.

Stratum tiene un conjunto limitado de mensajes que se utilizan en la comunicación:

- mining.subscribe, usado al comienzo de la conexión para indicar al servidor que el cliente está listo para comenzar la minería;
- mining.authorize, que envía información de identificación al servidor;
- mining.notify, que le indica al cliente los datos que se van a utilizar para minar; y
- mining.set difficulty: un mensaje corto utilizado por el servidor para indicar el nivel de dificultad utilizado cuando mina.

La Figura 5.1 muestra los datos de la capa TCP trasmitidos entre un minero un servidor de grupo de minería (*pool server*) durante una comunicación.

Los investigadores procesaron los datos generados por el tráfico de minería a través Netflow, seleccionaron los relacionados con el enrutamiento e ignoraron los relativos a la administración de red; también tuvieron en cuenta datos que tenían que ver con el número de paquetes y *bytes* relacionados con la duración total de los flujos del procesamiento de Netflow.

Una vez determinaron que las conexiones de Stratum son asimétricas, mientras el servidor transmite una gran cantidad de datos a los clientes, el cliente envía una cantidad muy pequeña de datos al servidor, decidieron combinar los registros de Netflow correspondientes a la salida y los flujos entrantes para explotar la asimetría.

Ya que no tenían acceso a una gran cantidad de capturas de tráfico, iniciaron recogiendo el flujo de las mediciones del tráfico obtenido (cada flujo tomo

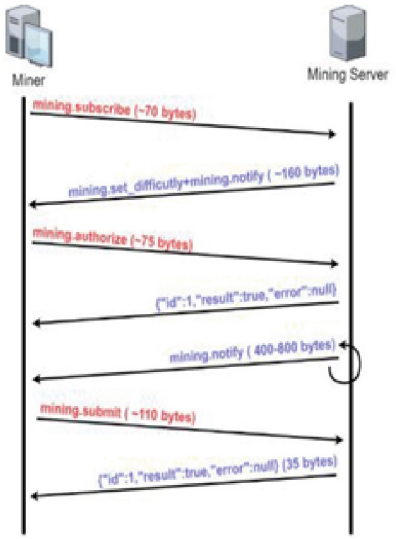

Figura 5.1. Conexión TCP entre un minero (rojo) y un pool server (azul) usando Stratum

entre cinco y treinta minutos), buscando así recolectar suficientes datos para entrenar el modelo de aprendizaje automático generando tráfico de minería por su propia cuenta. De esta manera, realizaron minería en diferentes servidores de criptomonedas, capturaron el tráfico y recolectaron 1'795.408 datos.

Los investigadores probaron cuatro modelos de *machine learning* en WEKA: SVM, CART (*Classification And Regression Tree*), *naive* Bayes y árbol de decisión C4.5. Este último obtuvo los mejores resultados, pues fue capaz de detectar con precisión el tráfico generado por la minería; la máxima profundidad del árbol fue de 13 nodos y en la mayoría de los casos empleó solo cinco operaciones divididas para clasificar correctamente el tráfico. La principal ventaja de C4.5 es que no necesita procesar una cantidad grande de datos de NetFlow para clasificar el tráfico de minería.

Dine and dash: static, dynamic, and economic analysis of in-browser cryptojacking

En este trabajo, Saad, Khormali y Mohaisen [17] analizan los aspectos de los ataques del *criptojacking* desde las perspectivas estática, dinámica y económica. Para el análisis estático realizaron categorizaciones basadas en contenido,

moneda y código de muestras de *criptojacking* con el fin de medir su distribución en sitios web, la afinidad con la plataforma minera y la complejidad del código, utilizando aprendizaje no supervisado, para así distinguir los *scripts* de *criptojacking* de los legítimos y de otros ejemplos maliciosos de JavaScript. Obtuvieron una precisión de 96,4 %. Para el análisis dinámico analizaron el efecto del *criptojacking* en los recursos críticos del sistema, como uso de CPU y batería y realizaron huellas digitales del navegador web para analizar intercambios entre un nodo víctima y el servidor de minería. Asimismo, construyeron un modelo analítico para evaluar la viabilidad del *cryptojacking* como alternativa a la publicidad en línea.

A modo de resumen, en la Tabla 5.1 se presenta una comparación de los trabajos citados teniendo en cuenta: el uso de técnicas de *machine learning*, el abordaje de variables de hardware (*behavior hardware*), el uso de Netflow para resumir el tráfico y el uso de variables de red (*Netflow features*), variables importantes en el proyecto reportado en este documento.

Tabla 5.1. Comparativo de proyectos

Proyecto	Machine learning	Behavior hardware	Netflow	Netflow features
Detection of bitcoin miners from network measurements	Si	No	Si	Si
How you get shot in the back: a systematical study about cryptojacking in the real world	No	Si	No	No
The browsers strike back: countering cryptojacking and parasitic miners on the web	Si	Si	No	Si
Detecting cryptocurrency miners with NetFlow/IPFIX network measurements	Si	No	Si	Si
Dine and dash: static, dynamic, and economic analysis of in-browser cryptojacking	Si	Si	No	Si

5.3. La investigación

Para este proyecto de investigación se decidió usar la metodología CRISP-DM (*Cross-Industry Estándar Process for Data Mining*) porque su estructura beneficia la resolución de problemas relacionados con la ciencia de datos, el aprendizaje de máquina y la minería de datos. Esta metodología consta de seis etapas:

entendimiento del negocio, entendimientos de los datos, preparación de los datos, modelado, evaluación y despliegue.

En la fase de entendimiento del negocio se analizaron las investigaciones reportadas en el estado del arte con énfasis en el trabajo de Muñoz et al. [12]. El entendimiento de los datos, por su parte, se dio tanto a nivel de red como a nivel de hardware.

A nivel de red, se tuvo acceso al *dataset* de capturas de tráfico de Muñoz et al. [12]. Este conjunto de datos se denominó minería pura, puesto que se conoce qué criptomoneda le corresponde a cada registro, e incluye cinco tipos de criptomonedas: BitCash, con 79 archivos pcap; Bitcoin, con 41 archivos pcap; Ethereum, con 106 archivos; Litecoin, con 77 archivos y Monero, con 55 archivos.

También se recopilaron archivos pcaps de tráfico de red almacenados en sitios web, como packetTotal, malware-traffic, contagiodump, que tienen evidencia de actividad de *cryptojacking* en entornos no controlados. Esto sitios, si bien no están apoyados por una entidad académica o gubernamental, cuentan con un nivel de popularidad respetable y presentan evidencia suficientemente documentada para tenerla en cuenta al crear un conjunto de datos. A este conjunto, que consta de treinta y nueve archivos, se le denominó minería no pura, ya que no se conoce qué criptomoneda se obtuvo. Cabe resaltar que algunos archivos contienen también tráfico de otros *malware*.

Para el tráfico normal se desarrolló un conjunto de herramientas que automatizaron el proceso de captura, usando como fuente los sitios más visitados según Alexa: la primera herramienta comprueba el estatus de los sitios web para saber si están activos; la segunda utiliza el API de VirusTotal para determinar si la URL es legítima, en ese proceso se toma como criterio las puntuaciones de todos los motores de antivirus de VirusTotal, si uno califica la URL con una puntuación negativa, se descarta del proceso con la lista de sitios web legítimos; la tercera herramienta ejecuta Selenium para abrir el navegador durante seis minutos, redireccionando el sitio web, y a su vez ejecuta tshark para capturar tráfico.

Este proceso se realizó en un entorno semicontrolado, usando una computadora con sistema operativo Linux Ming 19.3 Tricia, donde solo se mantuvieron los procesos vitales para su funcionamiento. De este proceso se obtuvo un total de 1284 archivos pcap.

Se utilizó también un conjunto de herramientas que hacen parte del proyecto nfdump [18] para procesar los tres conjuntos de datos en archivos nfpcaps, y la herramienta desarrollada por Urcuqui, Gaviria y Ramírez [19] para transformar los archivos nfpcacps en ventanas de tiempo, con la finalidad de resumir las comunicaciones de los flujos activos. Del anterior proceso de transformación se obtuvo:

- el conjunto normal, con 2.501 registros de ventanas de tiempo;
- el conjunto de minería no pura, con 37.053 registros de ventanas de tiempo;
- el conjunto de minería pura, con 2.232 registros de ventanas de tiempo;
- Bitcash, con 485 registros;
- Bitcoin, con 247 registros;
- Ethereum, con 959 registros;
- Litecoin, con 349 registros; y
- Monero, con 192 registros

Los tres conjuntos de datos contienen las siguientes variables:

- Name: nombre de la ventana de tiempo;
- Netflows: cantidad *netflows* en la ventana de tiempo;
- First Protocol: top 1 de los protocolos usados en la ventana de tiempo;
- Second Protocol: top 2 de los protocolos usados en la ventana de tiempo;
- Third Protocol: top 3 de los protocolos usados en la ventana de tiempo;
- P1 d: percentil 25 % de todas las duraciones en la ventana de tiempo;
- P2 d: percentil 50 % de todas las duraciones en la ventana de tiempo;
- P3 d: percentil 75 % de todas las duraciones en la ventana de tiempo;
- Duration: duración total de la ventana de tiempo;
- Max d: valor máximo de todas las duraciones en la ventana de tiempo;
- Min d: valor mínimo de todas las duraciones en la ventana de tiempo;
- Packets: número total de paquetes en la ventana de tiempo;
- Avg bps: promedio de bits por segundo en la ventana de tiempo;
- Avg pps: promedio de paquetes por segundo en la ventana de tiempo;
- Avg bpp: promedio de *bytes* por paquete en la ventana de tiempo;
- Bytes: número total de *bytes* en la ventana de tiempo;

- Number sp: número total de puertos de origen usados en la ventana de tiempo;
- Number dp: número total de puertos de destino usados en la ventana de tiempo;
- First sp: top 1 de los puertos de origen en la ventana de tiempo;
- Second sp: top 2 de los puertos de origen en la ventana de tiempo;
- Third sp: top 3 de los puertos de origen en la ventana de tiempo;
- First dp: top 1 de los puertos de destino en la ventana de tiempo;
- Second dp: top 2 de los puertos de destino en la ventana de tiempo;
- Third dp: top 3 de los puertos de destino en la ventana de tiempo;
- P1 ip: percentil 25 % de todas las entradas de paquetes en la ventana de tiempo;
- P2 ip: percentil 50 % de todas las entradas de paquetes en la ventana de tiempo;
- P3 ip: percentil 75 % de todas las entradas de paquetes en la ventana de tiempo;
- P1 ib: percentil 25 % de todas las entradas de *bytes* en la ventana de tiempo;
- P2 ib: percentil 50 % de todas las entradas de *bytes* en la ventana de tiempo;
- P3 ib: percentil 75 % de todas las entradas de *bytes* en la ventana de tiempo; y
- Type: tipo de ventana de tiempo (minera / normal).

En primera instancia se optó por usar el conjunto de datos de minería pura de Muñoz et al. [12] porque se conoce con certeza el criptoactivo asociado a cada registro. A este grupo se le concatenó el conjunto de datos normales, aplicando un proceso de aleatoriedad, para reorganizar las posiciones de los registros. Este grupo se usó para el análisis, modelado y evaluación.

A nivel de hardware, se exploró la actividad del *cryptojacking* usando un conjunto de datos obtenido de la cuenta de Kaggle de Keshani Jayasinghe [20], una reconocida investigadora en esta temática. El *dataset* incluye dos archivos CSV: uno de datos anormales, con 14.461 registros, otro de datos normales, con 80.851 registros, con las siguientes variables identificadas:

- cpu iddle, tiempo en que un procesador está inactivo;
- iowait, tiempo en que los procesos de la CPU pasan sin hacer nada;
- cpu nice, tiempo de CPU en uso;
- cpu softirq, interrupciones;
- cpu total, uso total del CPU;
- diskio sda1 write bytes, escritura de *bytes*; y
- diskio sda1 read bytes: lectura de *bytes*.

En la fase de preparación de los datos se realizó un análisis exploratorio univariado y otro bivariado. Cabe resaltar que solo se llega a esta parte con las variables de hardware, pues con las variables de red se realizó un preprocesamiento de datos y técnicas de ingeniería de variables con el propósito de escoger las mejores variables para el entrenamiento de los modelos.

Hay varios métodos para escoger los mejores predictores, pero al no existir una referencia que indicara cuál se adaptaría mejor a un conjunto de datos o problema en particular, se procedió a seleccionar dos métodos: Anova F-Value y ExtraTreesClassifer: el primero porque durante la exploración de datos se evidenció que todas las posibles variables de entrada, excepto una son de naturaleza numérica; el segundo porque aleatoriza subconjuntos de datos para reducir el sobreajuste. También se escogió un tercer grupo de características, bajo el criterio de un experto.

En la fase de modelado se realizaron tres iteraciones, donde cada uno de los cuatro modelos seleccionados (k-NN, SVM, *naive* Bayes y árbol de decisión), se entrenó con los tres conjuntos de predictores seleccionados, con la finalidad de descubrir el conjunto de variables óptimo.

Al contar con tan pocos registros de tráfico normal, fue necesario hacer estructuraciones en los conjuntos de datos, puesto que tendrían que compartir los registros normales. Para evitar que el mejor modelo conociera todos los datos no mineros, se descompuso el total de registros normales en: 70 % para el conjunto de minería pura y 30 % para minería no pura.

Durante el primer experimento, se procedió a entrenar una nueva instancia del mejor modelo, para verificar su capacidad de predicción frente al cambio de los datos de entrenamiento. En el segundo experimento, se puso a prueba al mejor modelo con el conjunto de datos de minería no pura, para observar la capacidad de inferencia ante datos de entornos no controlados. Durante el

tercer experimento, se procedió a reforzar el mejor modelo con los mejores predictores de los conjuntos de minería pura y no pura, combinando porciones de registros de ambos grupos, para luego aplicar una *cross-validation* con diez pliegues que permitiera escoger la mejor instancia y nuevamente proceder a probarlo con el conjunto no puro.

Para llevar a cabo lo anterior se estructuró un nuevo conjunto de datos de entrenamiento conformado con el 70 % de los registros normales, la totalidad de los registros mineros puros y el 6 % de los registros mineros no puros (equivalente a la misma cantidad de los mineros puros). Esta última porción quedo excluida del grupo de minería no pura.

En la fase de evaluación se utilizaron las siguientes métricas: *baseline*, *recall*, matriz de confusión, especificidad, *accuracy*, F1-score, Kappa, ROC, *precision* y área bajo la curva.

Durante el entrenamiento de todos los modelos, se le dio importancia a la *accuracy*, puesto que indica cuál fracción de las predicciones de las realizadas por cada modelo fue correcta. Cabe resaltar que el conjunto que se usó para entrenamiento y prueba está cerca de estar balanceado. La matriz de confusión, por su parte, permitió utilizar estimadores de la realidad y estimadores de predicción, para clasificar cuándo un registro es procedente de actividad minera.

Durante los experimentos se requiere que el mejor modelo se enfrente a datos nunca vistos y funcione de manera óptima, pero es posible que las características estén muy ajustadas al conjunto de entrenamiento y por ello "hagan" que el modelo prediga de manera casi perfecta el porcentaje de validación, pero fracase a la hora de clasificar registros recolectados en ambientes de incertidumbre. Por lo tanto, es necesario usar el conjunto de minería no pura, el cual indicará si el modelo tiene la capacidad de detección de mineros en otros escenarios, como en un ataque con una *botnet*.

También se planteó un escenario donde se requiere reforzar el modelo con registros de ambos conjuntos de datos, pero existe la posibilidad de que se ajuste nuevamente el modelo. Se procedió a realizar una validación cruzada, donde se hacen k iteraciones y por cada una de ellas se instancia un modelo que itera sobre k particiones, donde $k-1$ se usan para entrenamiento y una para validación, con el fin de estimar el rendimiento del mejor modelo reforzado y prevenir el sobreajuste. Una vez seleccionado el modelo se pone a prueba

con los datos no puros que se excluyeron en los procesos de entrenamiento y validación.

Finalmente, durante la fase de despliegue se pone en producción el mejor modelo de *machine learning*. A esta etapa no se llegó, por las limitaciones propias de un proyecto de investigación académico.

En el proyecto se utilizó como hardware: un computador XPS, con sistema operativo Linux Mint v.19.3 Tricia, como entorno de capturas y procesamiento de datos; y un computador ASUS, con sistema operativo Windows 10, para la ejecución de los modelos, la programación y la evaluación. Como software, se usó: Python v.3.6.9, para codificación; Tshark v.2.6.10 para la captura de tráfico; Selenium v.3.141.0, para la ejecución del navegador; y scikit-learn v.0.24.0, para los algoritmos de *machine learning*.

5.4. Resultados

5.4.1. Análisis univariado

Este análisis se realizó solo con los datos de minería pura con el fin de observar aspectos en la distribución de las variables, mediante los coeficientes de asimetría y curtosis (ver Tabla 5.2).

Tabla 5.2. Coeficientes de asimetría y curtosis

	Coeficiente_asimetría	Coeficiente_courtosis
netflows	5.126296	47.864906
p1_d	1.297105	0.327065
p2_d	1.218563	0.257082
p3_d	1.143464	0.257082
duration	4.878084	32.972115
max_d	1.022449	0.259034
min_d	1.342877	0.355841
#packets	13.328879	290.702150
Avg_bps	12.972739	304.806459
Avg_pps	15.317764	452.450504
Avg_bpp	0.648386	-1.017486

Tabla 5.2. Coeficientes de asimetría y curtosis (continuación)

	Coeficiente_asimetría	Coeficiente_courtosis
#Bytes	19.786796	578.555904
#sp	3.790053	22.180617
#dp	3.892243	23.004691
first_sp	1.919529	1.778351
second_sp	0.525267	-1.636671
third_sp	0.764612	-1.245960
first_dp	1.779636	1.249238
second_dp	0.601989	-1.545151
third_dp	1.782343	1.365050
p1_ip	3.752210	13.478351
p2_ip	3.710988	13.243028
p3_ip	3.589257	12.535532
p1_ib	4.019052	22.684737
p2_ib	3.172156	11.129250
p3_ib	2.879763	8.809831

Para las variables cualitativas se procedió hacer un conteo. Estos fueron los resultados más relevantes:

- las columnas con datos faltantes son Second Protocol, Third Protocol, second sp, third sp, second dp y third dp, con tasas de datos perdidos de 52 %, 97 % 7 %, 46 %, 7 % y 45 %, respectivamente;
- el conjunto de datos está prácticamente balanceado;
- el protocolo más recurrente es TCP, seguido de UDP e ICMP6;
- todas las características tienen una asimetría hacia la derecha, lo que significa que el promedio es mayor que la mediana en diferentes proporciones;
- la única característica cercana a la simetría es Avg bpp;
- las características con forma leptocúrtica, es decir, con alta concentración de los valores cerca de su media, son: netflows, duration, packets, Avg bps, Avg pps, Bytes, sp, dp, p1 ip, p2 ip, p3 ip, p1 ib, p2 ib y p3 ib;
- las características con forma platicúrtica, esto es baja concentración de los valores en torno a su media, son: p1 d, p2 d, p3 d, max d, min d, Avg bpp, first sp, second sp, third sp, first dp, second dp y third dp; y

- no hay ninguna variable con una distribución normal, por lo tanto, no es necesario aplicar este proceso con el conjunto de minería no pura.

Este análisis no permite formular una conclusión ni identificar un patrón, ya que el problema que sé está abordando es de naturaleza dicótoma.

5.4.2. Análisis bivariado: conjunto de entrenamiento

Se procedió con el análisis de los tres conjuntos (minería pura, minería no pura y tráfico normal), tomando como referencia las variables usadas en [12]. Cabe reiterar que en la presente investigación se usa el concepto de ventanas de tiempo, por lo que las conclusiones se formulan a partir de las variables equivalentes a las proporcionadas por Netflow.

De la investigación de Muñoz, Suárez-Varela y Barlet-Ros [12], las características usadas son: Packets / second Inbound; Bits / second Inbound; Bits / Packet Inbound; Packets / second Outbound; Bits /second Outbound; Bits / Packet Outbound; Packets Inbound / Packets Outbound; y Bits Inbound / Bits Outbound. Adicionalmente, se analizaron: Avg bps, Avgpps, Avgbpp, p1 ip, p2ip, p3ip, p1 ib y p2ib. A continuación, se relacionan los hallazgos más relevantes.

El tráfico de los registros normales tiene una probabilidad más alta de aparecer que el de los mineros, además de que tienen un rango más amplio en los promedios de *bits* por segundo (Figura 5.2).

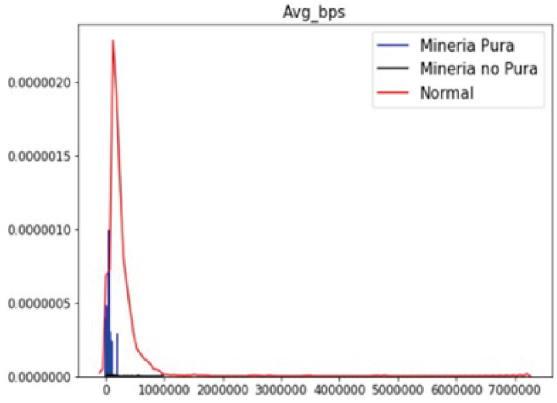

Figura 5.2. Avg bps vs tipo de tráfico

El tráfico de los registros mineros puros tiene una probabilidad mayor en los primeros promedios de paquetes, que el tráfico normal. La probabilidad de que aparezca minería no pura es casi nula, por lo menos en los flujos activos de las muestras (Figura 5.3).

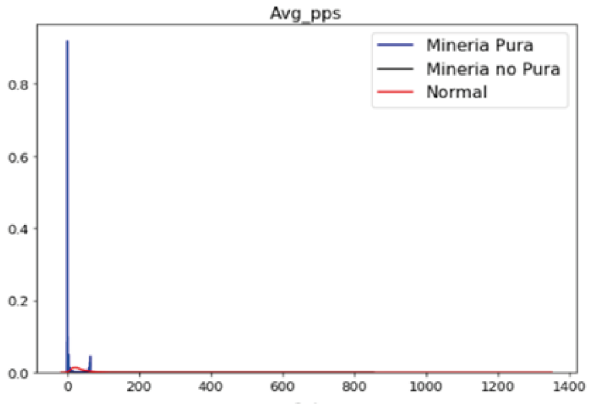

Figura 5.3. Avg pps vs tipo de tráfico

La trasmisión de *bytes*, en promedio, fue mayor en los mineros no puros; le sigue la de los mineros puros, lo cual indica que hubo una mayor trasmisión de *bytes* en procesos de minería (Figura 5.4).

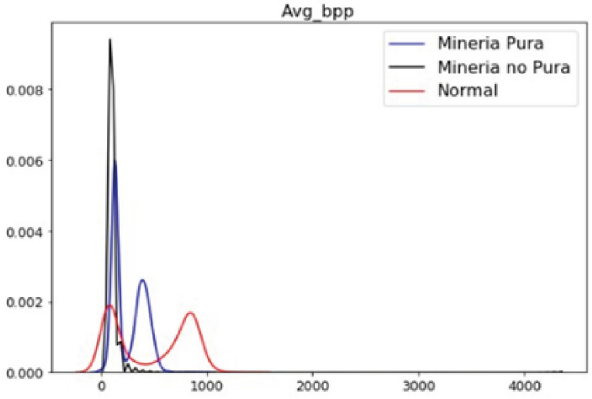

Figura 5.4. Avg bpp vs Tipo de tráfico

La probabilidad más alta se presenta en los registros normales en las primeras cantidades de paquetes; a pesar de tener poca probabilidad, los mineros puros pueden tomar una probabilidad de ocurrencia en un rango más amplio de valores en paquetes, en cambio, la presencia de minería no pura es muy baja, al igual que su ocurrencia (Figura 5.5).

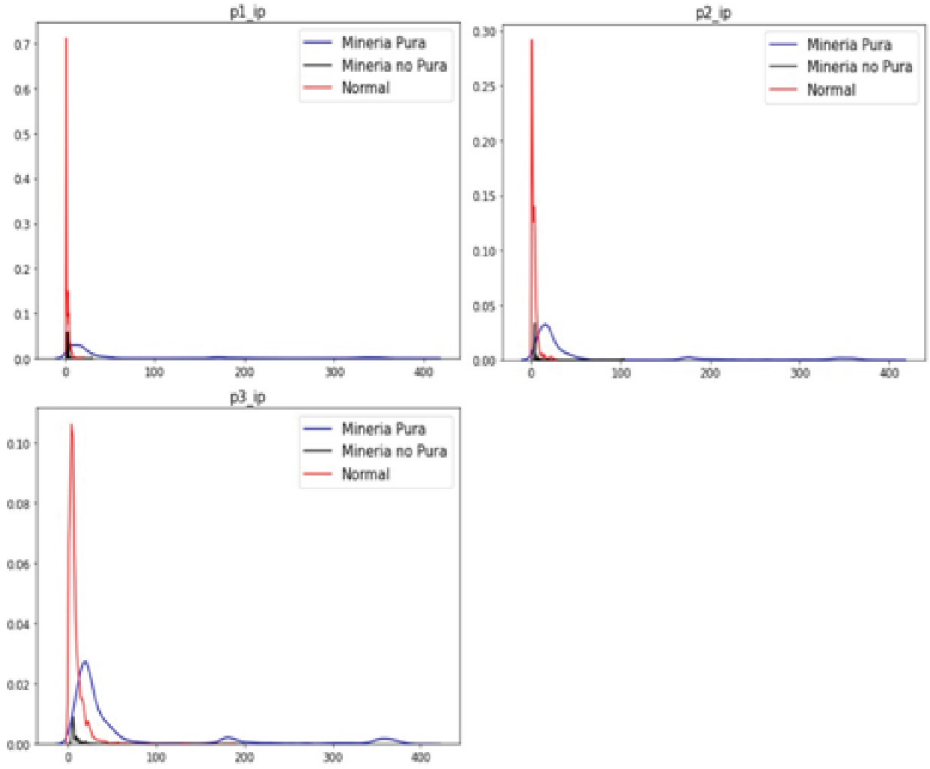

Figura 5.5. Percentiles de todas las entradas de paquetes vs tipo de tráfico

En los tres percentiles la probabilidad de ocurrencia es más alta en los registros normales durante los primeros intercambios de *bytes*. En los percentiles uno y dos (p1 ib y p2 ib) no hay presencia de mineros no puros y los puros presentan una ocurrencia mínima, aunque prolongada; en el percentil tres (p3 ib) se evidencia un incremento en la probabilidad de ocurrencia de los mineros no puros (Figura 5.6).

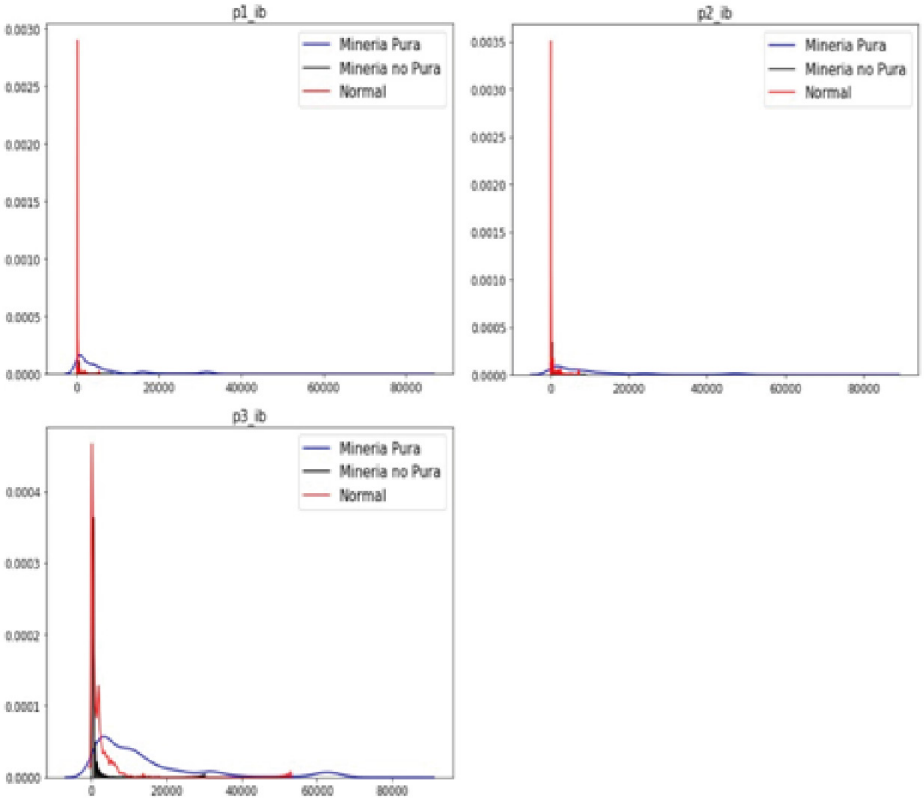

Figura 5.6. Percentiles de todas las entradas de bytes vs tipo de tráfico

5.4.3. Preprocesamiento

Durante esta fase, se imputaron los valores faltantes de las características second sp, third sp, second dp y third dp usando la mediana como criterio, por su robustez ante valores atípicos, y se tuvo que eliminar las características de *Second Protocol* y *Third Protocol* porque sus registros tenían un porcentaje de valores perdidos de 52 % y 97 %, respectivamente, ante lo cual no es factible eliminar registros ni usar otras estrategias de imputación, como el dato más frecuente o el uso de constantes, ni implementar modelos de aprendizaje automático, por su complejidad o incremento de sesgo.

Se decidió no eliminar los valores atípicos (la mayoría de variables supera el 9 %) o modificarlos, ya que el proceso de inferencia podría: verse afectado

con la introducción de un sesgo o afectar la distribución de las varianzas o el tamaño muestral; además, se debería tener una autoridad en la materia para determinar hasta qué punto se podría tolerar lo atípico. Por lo tanto, se optó por dejar los datos atípicos para identificar posibles anomalías en el tráfico.

Asimismo, se identificó que la mayoría de las variables de este conjunto presentan una alta multicolinealidad por medio de la inflación de la varianza, ya que los resultados superan el cinco, valor usado como referencia para la multicolinealidad de acuerdo con la ecuación (9), en donde *R2* es el coeficiente de determinación.

$$\frac{1}{1-R_2} \qquad (5.2)$$

Para finalizar, se aplicó *one-hot-encoding* para las variables categóricas y para la estandarización de las variables cuantitativas, ya que es más robusta que la normalización, ante valores atípicos.

5.4.4. Feature selection

En el conjunto de datos procesados, las variables de entrada son de naturaleza numérica y la variable objetivo es categórica. Para poder seleccionar los mejores predictores se escogió principalmente el método ANOVA-F para calcular la relación entre los valores de la varianza [21], en donde se ajustó la función seleccionada con el parámetro *K* de la clase SelectKBest de la librería scikit-learn. Este método determinó como mejores predictores a las variables: p1 d, p2 d, min d, second sp y second dp; el segundo método usado, *ExtraTreesClassifier*, seleccionó: p1 d, p2 d, min d, Avg bpp, first sp, second sp, first dp, second dp, p2 ib y p3 ib.

Como se puede apreciar, solo fue seleccionada una variable (Avgbpp) equivalente a las mencionadas en [12], algo bastante anormal debido a que los servidores de minería trasmiten gran cantidad de información hacia el minero, mientras que este último envía poca información. Esto se puede verificar por medio del flujo de *bytes* y paquetes, tanto de entrada como de salida [12]; Por tal razón, se seleccionaron otros cuatros predictores equivalentes a los mencionados en los trabajos anteriores para tenerlos en cuenta, con ello, las variables son las siguientes: Avgbps, Avgbpp, Avgpps, p3 ip y p3 ib.

5.4.5. Modelado

Al disponer de tres conjuntos de posibles predictores, se realizaron tres iteraciones en el entrenamiento y evaluación de los modelos, ya que como se mencionó en *Feature Selection* es muy extraño no tener variables relacionadas a bits y paquetes. Se usó la *accuracy* y F1-score para observar el rendimiento de los modelos ante las características seleccionadas.

Como se aprecia en la Tabla 5.3, los modelos tuvieron una *accuracy* muy alta, lo que significa que clasificaron de manera correcta casi todos los registros. Observando la media armónica de los mineros, se aprecia que los porcentajes son cercanos al 99 %, lo que indica que en cada caso los modelos tuvieron una tasa alta de desempeño a la hora de clasificar registros de minería, a excepción del modelo bayesiano, que tuvo un pequeño decremento. Esto mismo sucede con F1-score de los registros normales, lo que permite afirmar que los modelos son ideales, sin importar el conjunto de características.

Tabla 5.3. Comparación de los modelos frente a diferentes predictores

Métrica	Métricas de desempeño											
	Anova F				Extra Trees Classifier				Criterio de un experto			
	kNN	NB	SMV	DT	kNN	NB	SMV	DT	kNN	NB	SMV	DT
Accuracy	0.99	0.92	0.94	0.99	0.99	0.97	0.99	1.0	0.99	0.78	0.97	0.99
F1-score (mineros)	0.99	0.91	0.94	0.99	0.99	0.97	0.99	1.0	0.99	0.81	0.97	0.99
F1-score (normales)	0.99	0.93	0.94	0.99	0.99	0.97	0.99	1.0	0.99	0.74	0.97	0.99

Existen algunas explicaciones posibles ante la anomalía. Que los predictores tengan una relación casi lineal, debido a su alta correlación, puede estar relacionado con su alta multicolinealidad (Tabla 5.4), donde una variable puede estar configurada a partir de otras, lo que está causando un ruido que no permite a los métodos de selección de características identificar de manera clara la relación de otras variables con la variable objetivo, por la gran variabilidad de los predictores seleccionados (no hay mucha literatura sobre el efecto de la multicolinealidad en los modelos usados en esta investigación).

Otro fenómeno que se puede estar presentando es la existencia de relaciones espurias entre las variables, donde una variable no tomada en cuenta por

Tabla 5.4. Multicolinealidad

			Extra Trees Classifier			Criterio de un experto		
	Feature	VIF		Feature	VIF		Feature	VIF
0	p1_d	102.542838	0	p2_d	68.296691	0	Avg_bps	3.529267
1	p2_d	40.009904	1	p3_d	39.825271	1	Avg_bpp	1.540052
2	min_d	45.129893	2	min_d	19.183506	2	Avg_pps	3.125442
3	second_sp	5.505198	3	first_sp	9.477966	3	p3_ip	7.560062
4	second_dp	5.545944	4	second_sp	15.730467	4	p3_ib	7.399004
			5	first_dp	10.023918			
			6	second_dp	15.937182			
			7	First_Protocol_TCP	1.017401			

los métodos de selección de características puede hacer que otras variables se correlacionen. Aunque existen métodos matemáticos que se pueden usar para validar esta premisa, ella se puede explicar de manera objetiva con un hecho, el tiempo que duran los flujos entre los nodos de minería debería estar relacionado por el intercambio de *bytes* y paquetes.

Como se ha mencionado, la muestra fue recolectada en ambientes controlados (registros de minería pura) contemplando pocos escenarios, lo que puede restarle representatividad a la muestra, lo que a su vez haría que las variables se sobreajusten a su propio conjunto de datos [22]. Esta, entonces, podría ser la razón más plausible ante el hecho de que todos los modelos funcionen bien con conjuntos de variables distintas.

La Tabla 5.3 sugiere que los dos mejores modelos son k-NN y árbol de decisión; de ellos, se seleccionó el segundo por ser más eficiente, ya que no necesita procesar todos los datos, mientras que k-NN tiene que hacerlo en cada iteración, hecho por el cual es denominado como un algoritmo "perezoso".

5.4.6. Experimentos

Experimento 1

En este experimento se implementó un nuevo modelo de árbol de decisión, para validar su capacidad de predicción frente a los cambios en el conjunto de entrenamiento. Como se aprecia en la matriz de confusión de la Tabla 5,5. el modelo solo se equivocó tres veces (un conjunto de métricas para este

experimento se presenta en la Tabla 5.6). Asimismo, como se observa en la curva ROC (Figura 5.7), el modelo tiene un 99 % de efectividad para predecir los registros mineros de los normales (el área bajo la curva es casi perfecta).

Tabla 5.5. Matriz de confusión del mejor modelo

Árbol de decisión	Not_mine	Mine
Not_mine	543	1
Mine	2	649

Tabla 5.6. Métricas del mejor modelo

	Precision		Recall		Especificidad		F1-Score	
	Miner	No_miner	Miner	No_miner	Miner	No_miner	Miner	No_miner
Árbol de decisión	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
	Accuracy: 0.9				Kappa: 0.9			

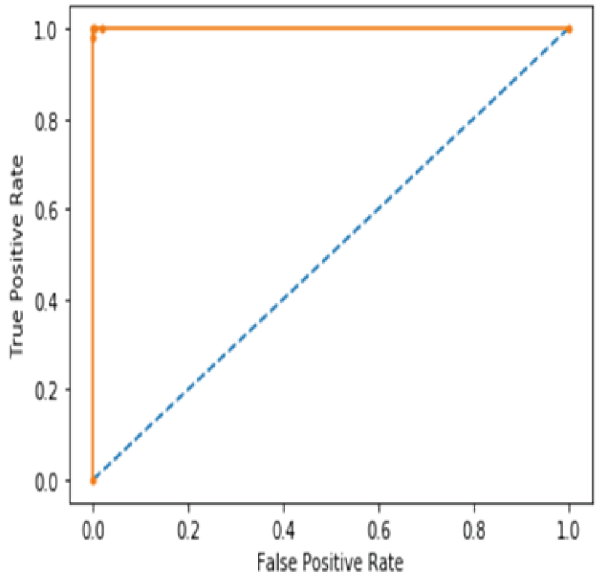

Figura 5.7. ROC del mejor modelo

Experimento 2

El mejor modelo se puso a prueba con el conjunto de datos de minería no pura, el cual contiene el 30 % de los registros normales. Se espera que el modelo falle ante este nuevo set, por la naturaleza de los datos de entreno, los cuales, como se mencionó, podrían no ser muy representativos por culpa de la manera en que fueron obtenidos.

Esta expectativa se confirma con los resultados que se presenta en la Tabla 5.7, en donde se destaca que el modelo se equivocó 34.194 veces (de un total de 37.803), a la hora de clasificar los registros, y en la Tabla 5.8, que presenta sus métricas de desempeño.

Tabla 5.7. Matriz de confusión del mejor modelo vs el conjunto de minería no pura

Árbol de decisión	Not_mine	Mine
Not_mine	749	1
Mine	34.193	2.860

Tabla 5.8. Métricas del mejor modelo vs el conjunto de minería no pura

	Precision		Recall		Especificidad		F1-Score	
	Miner	No_miner	Miner	No_miner	Miner	No_miner	Miner	No_miner
Árbol de decisión	0.9	0.02	0.07	0.99	0.99	0.07	0.14	0.04
	Accuracy: 0.09				Kappa: 0.003			

El interés ante estas métricas es reducir la especificidad y aumentar el *recall* de los mineros, ya que si la detección no es oportuna, el equipo afectado se puede deteriorar y se va a incrementar el costo de la energía. El modelo tuvo un rendimiento del 59 % con una diferencia mínima de la prueba sin capacidad discriminatoria.

Al reunir todo lo citado, es claro que el no es adecuado para predecir minería de ambientes no controlados. La Figura 5.8 corresponde a la comparación entre el ROC del mejor modelo y el del conjunto de minería no pura.

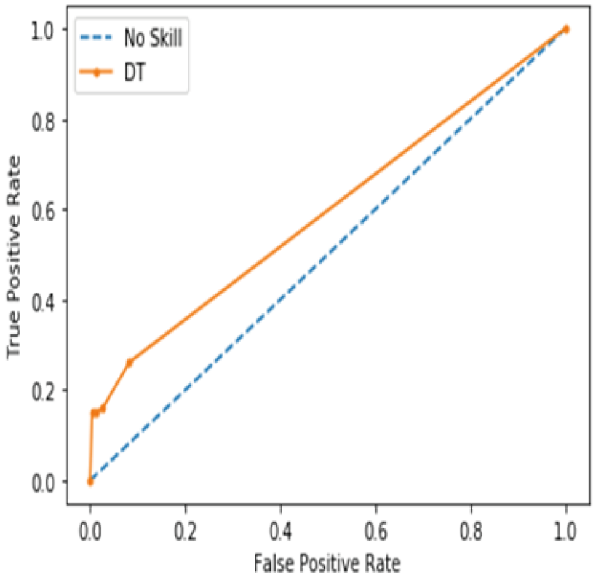

Figura 5.8. ROC del mejor modelo vs el conjunto de minería no pura

Experimento 3

Se procedió a utilizar las mejores variables de los conjuntos de minería pura y no pura para entrenar un nuevo modelo de árbol de decisión más robusto, con la hipótesis de que tendría mejores resultados a la hora de predecir. Para evitar un desbalance tan pronunciado en los datos, se realizó un sobremuestreo de los registros normales; asimismo, se combinaron las siguientes características: p3 d, min d, first sp, first dp, second sp y second dp.

Las variables de tiempo fueron seleccionadas porque el modelo se vería beneficiado al tener en cuenta cuánto dura el flujo de trasmisión de *bytes* en la comunicación de los nodos de minería. Se seleccionaron los top 1 y 2 de los puertos de origen y de destino porque existen *criptojackers* con bajas nociones de programación, que no configuran los puertos adecuadamente, por lo que en muchos basta con utilizar un *script* en un sitio web para iniciar el proceso de minado.

Es posible que un modelo falle ante nuevos datos, por lo cual se entrenó un nuevo modelo usando cross validation k fold, en él se dividen los datos del mismo tamaño k-1 y se usa uno de los grupos para validar el modelo,

maximizando el ajuste del modelo con los datos y reduciendo el *bias*. Este proceso se realizó con diez pliegues, siempre con la misma idea: usar el mejor modelo.

Ante la combinación de las mejores características, como se observa en la Tabla 5.10, el modelo robusto tuvo métricas casi perfectas. Posteriormente, el modelo se puso a prueba con los registros que nunca estuvieron involucrados en el entrenamiento, si bien este conjunto está desbalanceado, ese hecho no es relevante, ya que son datos de prueba que simulan un entorno real.

Tabla 5.9. Matriz de confusión del modelo robusto

Árbol de decisión	Not_mine	Mine
Not_mine	349	0
Mine	1	447

Tabla 5.10. Métricas del modelo robusto

	Precision		Recall		Especificidad		F1-Score	
	Miner	No_miner	Miner	No_miner	Miner	No_miner	Miner	No_miner
Árbol de decisión	1	0.99	0.99	0.1	0.1	0.99	0.99	0.99
	Accuracy: 0.99				Kappa: 0.99			

Las buenas métricas que obtiene el mejor modelo obtenido de la validación cruzada (Tabla 5.12) y la curva ROC de la Figura 5.9 indican que el modelo es muy bueno para la detección de *cryptojacking*, debido a que se ha realizado un proceso y se han obtenido buenas métricas. Claro está, su efectividad real solo se podrá demostrar en modo producción, alcance que no tiene el presente proyecto.

Tabla 5.11. Matriz de confusión del modelo robusto vs los datos de minería

Árbol de decisión	Not_mine	Mine
Not_mine	1.384	72
Mine	150	34.671

Tabla 5.12. Métricas del modelo robusto vs los datos de minería

	Precision		Recall		Especificidad		F1-Score	
	Miner	No_miner	Miner	No_miner	Miner	No_miner	Miner	No_miner
Árbol de decisión	0.99	0.90	0.99	0.95	0.95	0.99	0.99	0.92
	Accuracy: 0.99				Kappa: 0.92			

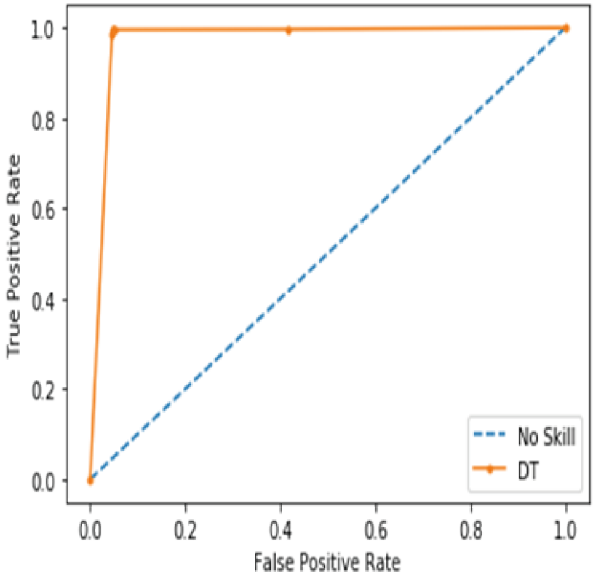

Figura 5.9. ROC del modelo robusto vs los datos de minería

5.4.7. Análisis de variables de hardware

Se analizó el comportamiento de mineros en algunas variables del conjunto de hardware: Como se aprecia en la Figura 5.10, existe una probabilidad alta de que los procesos legítimos, al terminar sus tareas, dejen de usar la CPU del equipo; en cambio, las CPU afectadas por los mineros, difícilmente van a estar inactivas durante mucho tiempo; este hecho, sumado a que algunas CPU cuando están suspendidas reducen el consumo de voltaje en tiempo de inactividad, hace alusión al desperdicio de consumo energético que representa un minero ilegal.

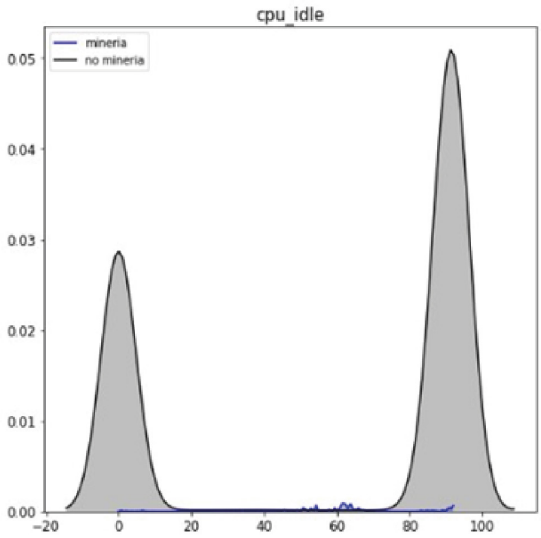

Figura 5.10. cpu idle

De acuerdo con la medida de iowait (Figura 5.11), hay más probabilidad de encontrar procesos en espera a leer y escribir en el disco, mientras que los procesos de minería no suelen tardar tanto tiempo.

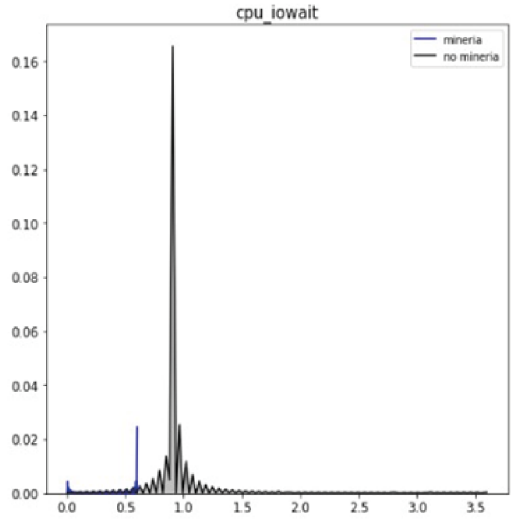

Figura 5.11. iowait

Es probable que un minero cause un cuello de botella en el sistema. De acuerdo con el registro de cpu_nice (Figura 5.12), hay una probabilidad mayor, pero poco prolongada, de que un minero estuviera usando la CPU, en lugar de un proceso legítimo, cuando se tomaron estos registros.

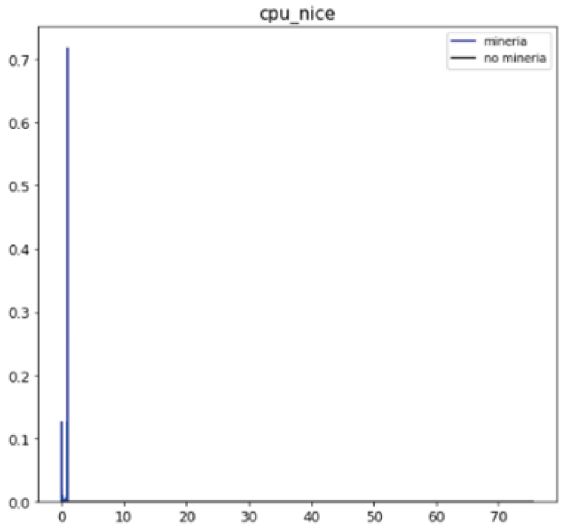

Figura 5.12. cpu nice

Es posible que la investigadora no dejara ejecutar el código minero por tanto tiempo. Si la métrica llegara ser negativa, el minero estaría consumiendo todo el tiempo en el uso de la CPU, lo cual podría estar relacionado con los cuellos de botella.

Hay más probabilidad de que los mineros estén causando interrupciones en el sistema que los procesos normales por cargas altas de trabajo. Aunque no se sabe en qué orden está sucediendo las interrupciones pesadas y livianas (Figura 5.13).

Los mineros tienen picos de probabilidades altos en comparación con los registros normales en el uso total de la CPU, lo interesante es que la actividad minera se dispara en un determinado y corto tiempo (Figura 5.14).

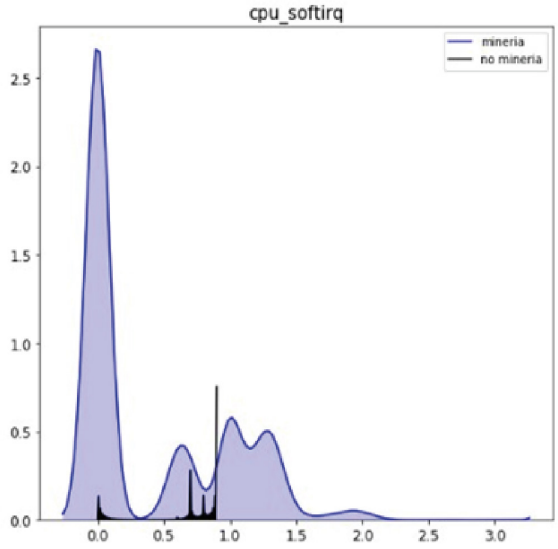

Figura 5.13. cpu softirq

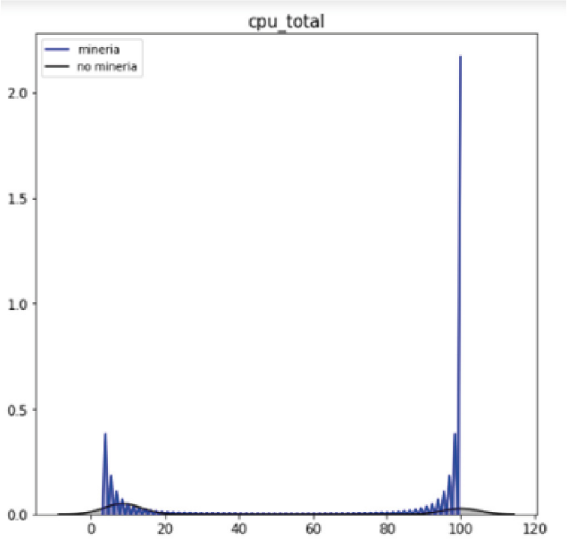

Figura 5.14. cpu total

Los mineros tienen probabilidades pequeñas y prolongadas en el tiempo en la lectura de *bytes*, pero no tienen tanta participación en la escritura de *bytes* (ver Figura 5.15). Esto es interesante y es probable que se relacione con el hecho de que las comunicaciones son asincrónicas y la lectura sea la información que el servidor de minería envíe al minero.

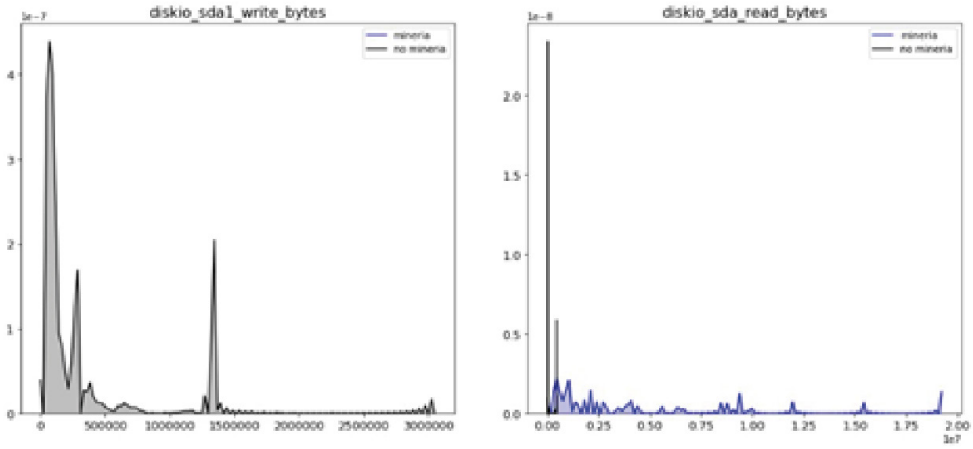

Figura 5.15. Escritura y lectura de bytes

5.5. Conclusiones y recomendaciones

Con base en los resultados del proyecto, se puede concluir:

- los mineros usan la CPU en su totalidad, como se evidencia en la Figura 5.14 y en el estado del arte;
- los mineros parecen tener un patrón común, el cual consiste en que el servidor de minería trasmite más información que el nodo minero y por ello causa más tráfico en la red, al menos en los primeros minutos;
- después de esos primeros instantes, es más complejo identificar el momento, la duración y la carga que el minero usa en la comunicación posterior, y aun más complejo saber cuándo el servidor le responde; y
- por medio de un análisis inicial de las variables de hardware es entonces posible notar que un minero pueda causar cuellos de botella en el sistema, debido al uso indebido de recursos de la CPU

Como trabajo futuro se propone:

- reforzar el modelo propuesto en este trabajo; explorar con mayor detalle la manera de contribuir en la explicación del patrón identificado, para incrementar las posibilidades de los modelos de identificar minería;
- entrenar un modelo de *machine learning* para la detección de actividad minera en hardware; y
- entrenar o hacer la propuesta de un modelo que tenga la capacidad de detección de *cryptojacking* a nivel de hardware, así como de red.

Los recursos (documentos, gráficos, códigos, etc.) obtenidos durante la ejecución del proyecto están disponibles en [23], con excepción de la presentación del proyecto realizada en la Universidad Nacional Abierta y a Distancia (UNAD), disponible en [24].

Referencias

[1] C. A. Arango-Arango, M. M. Barrera-Rego, J. F. Bernal-Ramírez, y A. Boada-Ortiz, *Criptoactivos*, Bogotá, Colombia: Banco de la República, 2018.

[2] S. Nakamoto. (2008). *Bitcoin: a peer-to-peer electronic cash system*. Disponible: https://bitcoin.org/

[3] X. Yang, Y. Chen y X. Chen, "Effective scheme against 51% attack on proof-of-work blockchain with history weighted Information," *2019 IEEE International Conference on Blockchain (Blockchain)*, 2019, pp. 261-265, https://doi.org/10.1109/Blockchain.2019.00041.

[4] S. Eskandari, A. Leoutsarakos, T. Mursch y J. Clark, "A first look at browser-based cryptojacking," en *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2018, p.p. 58-66, https://doi.org/10.1109/EuroSPW.2018.00014

[5] M. Rosenfeld, (2011), *Analysis of bitcoin pooled mining reward systems*. Disponible: https://arxiv.org/abs/1112.4980

[6] K. Jayasinghe y G. Poravi, G, "A survey of attack instances of cryptojacking targeting cloud infrastructure,", *APIT 2020: Proceedings of the 2020 2nd Asia Pacific Information Technology Conference*, January 2020, pp. 100–107. https://doi.org/10.1145/3379310.3379323

[7] Y. Li, G. Yang, W. Susilo, Y. Yu, M. H. Au y D. Liu, "Traceable Monero: anonymous cryptocurrency with enhanced accountability," en *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 2, pp. 679-691, 1 March-April 2021, https://doi.org/10.1109/TDSC.2019.2910058.

[8] S. Varlioglu, B. Gonen, M. Ozer, y M. Bastug, "Is cryptojacking dead after coinhive shutdown?," en *2020 3rd International Conference on Information and Computer Technologies (ICICT)* p.p. 385-389. https://doi.org/10.1109/ICICT50521.2020.00068

[9] M. A. Peters, B. Green, y H. Yang, "Cryptocurrencies, China's sovereign digital currency (dcep) and the us dollar system," *Educational Philosophy and Theory*, 2020. https://doi.org/10.1080/00131857.2020.1801146

[10] I. Yousaf y A. Shoaib, "Discovering interlinkages between major cryptocurrencies using high-frequency data: new evidence from covid-19 pandemic," *Financ Innov*, vol. 6, art. 45, 2020. https://doi.org/10.1186/s40854-020-00213-1

[11] D. Carlin, P. O'Kane, S. Sezer, y J. Burgess, "Detecting cryptomining using dynamic analysis," en *2018 16th annual conference on privacy, security and trust (PST)*, https://doi.org/10.1109/PST.2018.8514167

[12] J. Z. i. Muñoz, J. Suárez-Varela y P. Barlet-Ros, "Detecting cryptocurrency miners with NetFlow/IPFIX network measurements," *2019 IEEE International Symposium on Measurements & Networking (M&N)*, 2019, pp. 1-6, https://doi.org/10.1109/IWMN.2019.8804995.

[13] Cisco, (2020), *Cisco annual Internet report (2018–2023)*, white paper. Disponible: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[14] P. O'Kane, S. Sezer, K. McLaughlin y E. G. Im, "SVM training phase reduction using dataset feature filtering for malware detection," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 3, pp. 500-509, March 2013, https://doi.org/10.1109/TIFS.2013.2242890.

[15] G. Hong, Z. Yang, S. Yang, L. Zhang, Y. Nan, Z. Zhang, M. Yang, Y. Zhang, Z. Qian, y H. Duan, "How you get shot in the back: a systematical study about cryptojacking in the real world," en *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications*

Security, p.p. 1701–1713. New York, NY, ACM. https://doi.org/10.1145/3243734.3243840

[16] R. Tahir, S. Durrani, F. Ahmed, H. Saeed, F. Zaffar, y S. Ilyas, "The browsers strike back: countering cryptojacking and parasitic miners on the web," en *IEEE Infocom 2019, IEEE Conference on Computer Communications*, p.p. 703-711. https://doi.org/10.1109/INFOCOM.2019.8737360

[17] M. Saad, A. Khormali, y A. Mohaisen, "Dine and dash: static, dynamic, and economic analysis of in-browser cryptojacking," en *2019 APWG Symposium on Electronic Crime Research (ECRIME)*, https://doi.org/10.1109/eCrime47957.2019.9037576

[18] P. Haag. (2015). *Nfdump* [repositorio GitHub]. Disponible: https://github.com/phaag/nfdump

[19] C. C. Urcuqui, J. Gaviria, y A. Ramírez, *Analysis of time windows to detect botnet's behavior*, ponencia en DragonJAR Security 2020. https://doi.org/10.13140/RG.2.2.26451.81441

[20] K. Jayasinghe, *Cryptojacking attack timeseries dataset*. Disponible: https://www.kaggle.com/keshanijayasinghe/cryptojacking-attack-timeseries-dataset/code

[21] J. Brownlee, *Machine learning mastery with python: discover the fastest growing platform for professional machine learning with step-by-step tutorials and end-to-end projects*, 2020.

[22] A. D'Amour, K. Heller, D. Moldovan, et al., *Underspecification presents challenges for credibility in modern machine learning*. https://doi.org/10.48550/arXiv.2011.03395

[23] C.C Urcuqui. (2021). Minerdetector. *i2tResearch* [repositorio GitHub]. Disponible: https://github.com/i2tResearch/Ciberseguridad_web/tree/master/MinerDetector

[24] S. Bernal, y C. C. Urcuqui. *Detección de Cryptocjaking aplicando técnicas de ciencia de datos* [video]. Disponible: https://youtu.be/2VTNDG-ClXM

Capítulo 6
Análisis de ventanas de tiempo para detectar el comportamiento de botnets

Julio César Gaviria, Ánderson Ramírez, Christian Camilo Urcuqui, Andrés Navarro

6.1. Introducción

El conocimiento le ha permitido a la humanidad crear nuevas tecnologías que le han ayudado a progresar; sin embargo, un problema recurrente ha sido su capacidad para tergiversar el uso de ciertas herramientas tecnológicas. Un ejemplo de ello son las *botnets*, que surgieron como una nueva forma de comunicación, y han llegado al punto en que algunas personas malintencionadas usan sus ventajas para cometer ciberdelitos [1].

El termino *botnet* surgió con el nacimiento de EggDrop, la primera utilizada con fines legítimos [1], que administraba canales de *chat* a través del protocolo IRC (*Internet Relay Chat*). Con el paso del tiempo, algunos curiosos empezaron a utilizar las herramientas que proveía el protocolo IRC para crear *botnets* malignas, capaces de: expandirse por su cuenta a través de vulnerabilidades en el software, comprometer información valiosa o incluso realizar ataques de denegación de servicio distribuido (DDoS, *Distributed Denial of Service*).

Hoy en día siguen existiendo familias de *botnets* malignas. Algunas de ellas ya han sido deshabilitadas gracias a la colaboración de investigadores en el área de seguridad informática y de algunos gobiernos, como en el caso de la *botnet* GRUM [2]. Los cibercriminales que aún controlan *botnets* activas han desarrollado la capacidad de implementar nuevas técnicas con el fin de volverlas dinámicas en el tiempo. Lo anterior se debe a la inteligencia de los ciberdelincuentes que están a cargo de estos mecanismos, que les ha permitido generar nuevos algoritmos que le facilitan a la *botnet* migrar sus servidores de comando y control hacia otros sitios de manera autónoma. Además, los ciberdelincuentes han logrado cambiar las topologías de sus redes, haciendo aún más difícil la identificación de todo el sistema.

Los cibercriminales que controlan las *botnet* activas siguen cometiendo ciberdelitos, tales como ataques de denegación de servicio (DoS, D*enial of Service*), robo de información e incremento de *spam*. Al mismo tiempo, algunas de ellas han sido liberadas con el fin de que personas curiosas puedan pagar por su uso [1]. Es pertinente saber que puede ser difícil eliminarlas por completo, ya que día tras día se siguen incorporando nuevos *exploits* (módulos de programación), con el fin de hacerlas más completas, como en el caso de la *botnet* Echobot [3]. Pero no solo eso, según García [1], existen algunos gobiernos u organizaciones cibercriminales que financian esta clase de mecanismos. Por lo anterior, muchas *botnets* siguen expandiéndose a través del uso de *spam* o mediante actividades de ingeniería social, incrementando así el número de delitos informáticos exponencialmente.

Actualmente existen muchos sectores que emplean infraestructuras tecnológicas como parte de su organización, tales como los gobiernos y las empresas. Muchas de estas entidades pueden estar en peligro de ser comprometidas por un mecanismo de este tipo, el cual podría usar sus equipos para llevar a cabo los objetivos del ciberdelincuente.

El problema de la detección de ciberataques de tipo *botnet* se presenta por diversos motivos, el más importante es el dinamismo constante que revela el mecanismo. Este comportamiento es provocado porque día tras día se identifican nuevas vulnerabilidades en el software, que permiten al ciberdelincuente abrir caminos hacia su objetivo. Las malas prácticas de programación, los baches de seguridad en las plataformas y el mal diseño en la arquitectura del software son algunas de las causas que incrementan la posibilidad de crear nuevos caminos hacia la ciberdelincuencia.

Un reporte realizado por spamhaus.org muestra el incremento exponencial de estos mecanismos y destaca cómo entre 2017 y 2018 se incrementó en algo más del 100 % el número de nombres de dominio registrados y configurados por cibercriminales con el único propósito de hospedar una *botnet*. La cifra pasó de 50.000 a 103.503 dominios. Por su parte NSFOCUS, una red global líder en ciberseguridad, que protege empresas de ciberataques avanzados, menciona que en el 2018 se detectaron 111,472 mil instrucciones de ataque de diferentes familias de *botnets*, que fueron recibidas por un total de 451,187 objetivos de ataque, lo que representa un incremento de 66,4 % respecto de 2017 [4]. También es pertinente tener en cuenta que el financiamiento que reciben los grupos o los creadores de estos mecanismos provoca que cada vez

se mejoren e incrementen las funcionalidades de la red, haciéndola aún más compleja de detectar. Todo lo anterior provoca el incremento de actividades maliciosas, tales como *spam*, DDoS y robo de información, inclusive, puede dar paso a la creación de nuevas *botnets*, como ha sucedido ya.

Actualmente existen muchas investigaciones que han abordado el análisis y la detección de *botnets* desde distintas perspectivas, en algunas de ellas inclusive, los investigadores han podido trabajar junto con entidades gubernamentales y desactivar algunos de estos mecanismos. El problema surge cuando la *botnet* se vuelve dinámica, algo que muchas de estas investigaciones pasan por alto, debido a que se vuelven obsoletas. Debido a esto y a la agilidad de los cibercriminales, los DDoS se siguen presentando, tal como lo refleja Kaspersky lab en uno de sus reportes [5]. Este mismo reporte menciona una estadística de servidores de administración de *botnets*, en él, el primer lugar lo ocupan los Estados Unidos (44,14 %), seguidos de los países bajos (12,16 %) y el Reino Unido (9,46 %).

Lo dicho evidencia la existencia de estructuras de este tipo. Además, la alta tasa de servidores de administración de *botnets* muestra la dificultad de tener una prevención adecuada ante un posible ataque. Por tal motivo, es necesario implementar un mecanismo que permita notificar a un conjunto de usuarios la presencia de patrones de comportamiento maliciosos, propios de la fase de infección o C&C del ciclo de vida de la *botnet*, con el fin de reducir el índice de ciberdelincuencia.

Detectar y predecir un ciberataque de tipo *botnet* es difícil por la dificultad de mitigar a los cibercriminales denominados *botmasters*, debido a su inteligencia y astucia, y por el dinamismo de la estructura de la *botnet*, que le permite aprovechar vulnerabilidades presentes en el software debido al error humano. También, puede ser complejo llevar investigaciones de este tipo, debido al descontento de los agentes externos y la complejidad de los datos para su estudio. Además, existe una clara ausencia de mecanismos que permitan predecir si un conjunto de usuarios hace parte de la estructura de la *botnet* (zombies) a través del análisis de tráfico de red.

Tomando en consideración la situación descrita, el proyecto actual se desarrolló con el objetivo general de desarrollar una herramienta que permita detectar patrones de comportamiento maliciosos, propios de la fase del ciclo de vida de C&C de las *botnets*, haciendo uso de ventanas de tiempo. Asimismo, como objetivos específicos fueron definidos: obtener una muestra de pcaps a

través del análisis de tráfico de red; generar un *dataset* con variables de tráfico de red, clasificado en registros legítimos y maliciosos; evaluar un conjunto de modelos de *machine learning* que permitan predecir si un tráfico de es legítimo o si presentan evidencias de comportamientos maliciosos; y desarrollar una herramienta de software que permita notificar a un conjunto de usuarios si son parte de la estructura de la *botnet* (zombies).

6.2. Estado del arte

La revisión realizada por el proyecto incluyó dos investigaciones: *Identifying, modeling and detecting botnet behaviors in the network* [1] y *Botnet command detection using virtual honeynet* [6].

Identifying, modeling and detecting botnet behaviors in the network

García [1] recopiló una serie de variables relacionadas con *botnets* presentes en investigaciones anteriores y propuso categorizarlas dependiendo de cuatro dimensiones globales: fuentes de detección (la fuente principal de la información usada para la detección; características de detección; técnicas de detección (con énfasis en las más efectivas en la detección de *botnets*; y algoritmos de detección. Con base en estas cuatro dimensiones presentó el cuadro comparativo de la Tabla 6.1.

Tabla 6.1. Cuadro comparativo de investigaciones sobre botnets [1]

Survey	Fuentes	Características	Técnicas	Algoritmos
A survey of botnet technology and defenses [7]	Paquetes de red, DNS logs, red oscura u flujos de tráfico.	No incluido	Comportamiento (ataque y cooperativo) y firma	No incluido
Botnet : Classification , Attacks, Detection, Tracing and preventive measures [8]	Honeynets y paquetes de red.	No incluido	Basado en firmas y anomalías	No incluido
A Taxonomy of Botnet Detection Techniques [6]	Honeynets e Intrusion Detection Systems (IDS).	No incluido	Firmas y anomalías (host y basadas en redes).	No incluido

Tabla 6.1. Cuadro comparativo de investigaciones sobre botnets (continuación)

Survey	Fuentes	Características	Técnicas	Algoritmos
Network Anomaly Detection System: The state of Art of Network Behavior Analysis [9]	No incluido	No incluido	Modelos de anomalías, basados en modelos, reglas y estadísticas; y modelos de especificación, basados en protocolos, estados y transacciones.	Data mining, neuronal networks pattern matching, expert systems, Bayesians, covariance, matrices, chi-squared
A Wide Scale Survey on Botnet [10]	Honeynets y tráfico de red.	No incluido	Basado en comportamiento, DNS y minería de datos.	No incluido

Como se observa, muchas de las investigaciones evaluadas por García [1] utilizan técnicas de mitigación, tales como *honeynets* y *honeypots*, que no se usan en el actual proyecto; además, quienes utilizaron técnicas de *machine learning*, no mencionan sus fuentes de detección, si fue, por ejemplo, análisis de tráfico de red. García identifica algunos problemas en ellas:

- su escasa reproducibilidad, característica define si una investigación aporta datos significativos, suficientes para que investigaciones posteriores puedan rehacer sus pasos;
- el elevado número de *host* cuando el diseño de la investigación depende de esta característica, muchas investigaciones necesitan un gran número de *host* y paquetes para obtener buenos resultados, lo que tiene un efecto negativo, ya que es difícil en algunas ocasiones acceder a la misma cantidad de *host* utilizados;
- las características que surgen cuando los investigadores preprocesan los datos antes de la extracción de características, pues tienden a sobrefiltrarlos y generan los algoritmos con base en ellos; y
- la mayoría de las investigaciones no computan el set completo de métricas o no definen los experimentos correctamente, lo que imposibilita la comparación entre investigaciones.

Considerando lo dicho, García propone trabajar: con análisis de tráfico de red, porque es el mejor camino para proteger miles de computadores al mismo tiempo; y con base en el comportamiento, porque parece el camino que mejor representa las acciones complejas de las *botnet*.

En términos de metodología, García:

- creó una instalación de captura de *malware* para obtener un tráfico de *malware* bueno, etiquetado y real;
- desarrolló una herramienta denominada SimDetect para detectar similitudes en el tráfico *botnet*;
- desarrolló BClus, un método que automáticamente realiza *clusters* y detecta grupos similares en cuanto a su comportamiento sobre las *botnets*; y,
- con una mejor comprensión de los comportamientos de las *botnets*, creó CCDetector, que permite detectar los patrones de comportamiento de los canales de C&C de las *botnets*, usando un novel *state based model* del tráfico de red.

Además, BClus y CCDetector fueron comparados con otros métodos, verificando sus resultados con técnicas del estado del arte.

La gran ventaja de esta investigación es que ayuda a entender los comportamientos de las *botnets* analizando tráfico real. Además, los algoritmos y los pcaps generados en ella, al ser de código abierto, permiten reproducir la investigación. El único problema es que, aunque permite detectar patrones de comportamiento: no enfatiza en la extracción de ciertas variables de tráfico de red que puedan ser de suma importancia para el análisis y la detección en tiempo real; no utiliza un módulo de notificación para avisar al usuario de la probabilidad de ser un objetivo de la *botnet*; y no usa modelos de *machine learning* para predecir una acción maliciosa.

Botnet command detection using virtual honeynet

Bhatia, Sehgal y Kumar [11] muestran una perspectiva un poco diferente de las investigaciones revisadas por García [1], afirman que existen primordialmente dos enfoques para la detección y el seguimiento de *botnets*: uno basado en la creación de redes trampa, otro, en el monitoreo y análisis pasivo del tráfico de la red. Los investigadores desarrollaron un algoritmo basado en el comportamiento de la red en tiempo de ejecución de los *bots* y a la secuencia de comandos utilizada en la conversación que estos tienen en la fase de C&C. La única desventaja es que, a pesar de que crearon el prototipo, desarrollar una infraestructura escalable y robusta para alcanzar sus objetivos es un problema [11].

Esta investigación es muy proactiva, ya que permite visualizar a gran escala una serie de comandos que cualquier *botnet* podría utilizar en la fase de C&C.

La diferencia con el proyecto actual es que aun cuando este realiza el análisis de tráfico de red con el fin de utilizar una serie de variables extraídas de los datos, Bhatian et al. utilizan un prototipo completo basado en *honeynets* (que en el proyecto actual no se usa), con el fin de hacer correlaciones entre comportamiento y nombres de comandos utilizados en la fase de C&C. Además, Bhatian et al. no utilizan un módulo de notificación al usuario para avisarle la probabilidad de ser un objetivo de la *botnet* ni técnicas de *machine learning* que permitan predecir ese comportamiento.

6.3. La investigación

Previo a la presentación de la metodología, es necesario analizar los dos principales componentes del proyecto. El primero está directamente relacionado con la ciencia de datos, por la necesidad de generar un *dataset* a partir de los pcaps obtenidos, ese conjunto de datos se debe limpiar realizando una exploración de los datos con el fin de detectar datos atípicos, anómalos, escalas y otros factores propios del EDA; una vez procesados los datos, el siguiente paso es realizar el entrenamiento y evaluación de los modelos de *machine learning*. El segundo contempla el desarrollo de un software que permita notificar al conjunto de usuarios si hacen parte de la estructura de la *botnet* (zombies); la herramienta será desarrollada bajo el esquema incremental (elicitación de requerimientos, diseño y desarrollo del prototipo de la solución), teniendo en cuenta que debe tener un módulo de notificación asociado a los modelos anteriormente entrenados y un posible modulo que permita la recopilación de tráfico de red. Para su desarrollo, el proyecto seleccionó dos conceptos: la metodología CRISP-DM y el ciclo de vida incremental.

Para la fase de entendimiento del negocio, se realizó el estudio de las investigaciones presentes en el estado del arte. Para la segunda fase, el entendimiento de los datos, antes de comenzar la investigación se indagó acerca de la naturaleza de los datos que se usarían en el desarrollo del aplicativo. Para ello, se consultó la página de Stratosphere Lab [12], para entender la forma cómo obtuvieron sus registros. Los datos legítimos de los que ellos disponían fueron registros de peticiones HTTP a varios sitios web registrados en una página denominada Alexa; mientras que los datos maliciosos evidenciaban capturas de *malware* en las que quedaban registrados comportamientos propios del ciclo de vida de C&C de las *botnets* que estaban estudiando.

Una vez transformados los pcaps a ventanas de tiempo, se generó un aproximado de 258.697 registros extraídos por medio de un algoritmo. Una ventana de tiempo es un periodo (generalmente de cinco minutos), en donde se condensa un conjunto de comunicaciones evidenciadas a través de la red, donde estas conexiones se representan en forma de *netflows* (resumen de la comunicación desde un nodo hacia otro).

Las ventanas de tiempo generadas se dividieron en: 519 legítimas y 258.178 maliciosas. Para intentar corregir el desbalance presente en la cantidad de datos, en una primera instancia se aplicó la técnica de *oversampling*, intentando generar una mayor cantidad de datos legítimos; el resultado fue la generación de 649 nuevas ventanas de tiempo legítimas. Como ese resultado estaba muy lejos de poder igualar la cantidad de datos maliciosos, se decidió aplicar la técnica de *undersampling*, para lo cual se tomó una muestra de los datos maliciosos.

La Tabla 6.2 describe las variables presentes en los *datasets*, algunas de ellas fueron generadas directamente con la herramienta de NFDUMP; las demás fueron propuestas por la presente investigación, debido a que era necesario darle un enfoque particular a esta propuesta.

Tabla 6.2. Variables presentes en los datasets

Variable	Descripción
1. Name	Nombre de la ventana de tiempo
2. Netflows	Cantidad de netflows en la ventana de tiempo
3. First_Protocol	Top 1 de los protocolos usados en la ventana de tiempo
4. Second_Protocol	Top 2 de los protocolos usados en la ventana de tiempo
5. Third_Protocol	Top 3 de los protocolos usados en la ventana de tiempo
6. P1_d	Percentil 25 de todas las duraciones en la ventana de tiempo
7. P2_d	Percentil 50 de todas las duraciones en la ventana de tiempo
8. P3_d	Percentil 75 de todas las duraciones en la ventana de tiempo
9. Duration	Duración total de la ventana de tiempo
10. Max_d	Valor máximo de todas las duraciones en la ventana de tiempo
11. Min_d	Valor mínimo de todas las duraciones en la ventana de tiempo
12. Packets	Número total de paquetes en la ventana de tiempo
13. Avg_bps	Promedio de bits por segundo en la ventana de tiempo
14. Avg_pps	Promedio de paquetes por segundo en la ventana de tiempo
15. Avg_bpp	Promedio de bytes por paquete en la ventana de tiempo

Tabla 6.2. Variables presentes en los datasets (continuación)

Variable	Descripción
16. Bytes	Número total de bytes en la ventana de tiempo
17. Number_sp	Número total de puertos de origen usados en la ventana de tiempo
18. Number_dp	Número total de puertos de destino usados en la ventana de tiempo
19. First_sp	Top 1 de los puertos de origen en la ventana de tiempo
20. Second_sp	Top 2 de los puertos de origen en la ventana de tiempo
21. Third_sp	Top 3 de los puertos de origen en la ventana de tiempo
22. First_dp	Top 1 de los puertos de destino en la ventana de tiempo
23. Second_dp	Top 2 de los puertos de destino en la ventana de tiempo
24. Third_dp	Top 3 de los puertos de destino en la ventana de tiempo
25. P1_ip	Percentil 25 de todas las entradas de paquetes en la ventana de tiempo
26. P2_ip	Percentil 50 de todas las entradas de paquetes en la ventana de tiempo
27. P3_ip	Percentil 75 de todas las entradas de paquetes en la ventana de tiempo
28. P1_ib	Percentil 25 de todas las entradas de bytes en la ventana de tiempo
29. P2_ib	Percentil 50 de todas las entradas de bytes en la ventana de tiempo
30. P3_ib	Percentil 75 de todas las entradas de bytes en la ventana de tiempo
31. Type	Tipo de ventana de tiempo (maliciosa/legítima)

En la fase de preparación de los datos, se realizó su exploración y limpieza, así: se eliminaron los valores faltantes de las columnas; se cambió el tipo de algunas columnas (Object-Int, Float-Int); se eliminaron los *outliers*; se seleccionaron las variables más representativas para el conjunto de datos mediante la técnica de *feature importance*, por cada experimento realizado; y se utilizó un escalamiento (normalización) de los datos para facilitar su manejo.

Por su parte, en la fase de modelado se entrenaron los siguientes algoritmos clasificatorios: *naive* Bayes gaussiano, árboles de decisión, k-NN, regresión logística y *random forest*, y se realizó un *holdout* para obtener un 70 % de los datos dirigidos al entrenamiento de los modelos y 30 % para las pruebas. En cada uno de los modelos se aplicó una técnica denominada GridSearchCV, la cual realiza una búsqueda exhaustiva de los mejores parámetros especificados para un modelo en específico.

En la fase de evaluación, se tuvieron en cuenta las métricas: *precision*, *recall*, *F1 score*, *accuracy* y coeficiente Kappa, las cuales fueron almacenadas en un archivo CSV, para comparar los modelos presentes en cada experimento. Al final de

cada prueba realizada a cada modelo, se desplegó una matriz de confusión con el fin de evidenciar la distribución de las predicciones categorizadas en VP, VN, FP y FN.

Ciclo de vida incremental

El modelo de ciclo de vida incremental permite tener un desarrollo basado en incrementos, lo que representa un mayor control sobre el flujo del proyecto estructurado junto con el cronograma planteado, logrando así alcanzar un crecimiento progresivo de la funcionalidad global del proyecto.

El proyecto se desarrolló en cuatro fases. En la primera se descargaron los pcaps legítimos y maliciosos del laboratorio Stratosphere y se almacenaron en un archivo CSV las URL de descarga de cada uno de los pcaps, con su nombre original y el nuevo nombre que se le dio al archivo (ver ejemplo en la Figura 6.1).

	Ref	Download	Pcap Name
0	1	https://mcfp.felk.cvut.cz/publicDatasets/CTU-M...	2018-05-03_win12
1	2	https://mcfp.felk.cvut.cz/publicDatasets/CTU-M...	162.222.213.28
2	3	https://mcfp.felk.cvut.cz/publicDatasets/CTU-M...	2018-04-04_win16
3	4	https://mcfp.felk.cvut.cz/publicDatasets/CTU-M...	2018-04-03_win12
4	5	https://mcfp.felk.cvut.cz/publicDatasets/CTU-M...	2018-04-03_win11
...	...	...	...
75	76	https://mcfp.felk.cvut.cz/publicDatasets/CTU-M...	2017-06-24_win8
76	77	https://mcfp.felk.cvut.cz/publicDatasets/CTU-M...	2017-06-24_win7
77	78	https://mcfp.felk.cvut.cz/publicDatasets/CTU-M...	2017-06-24_win6
78	79	https://mcfp.felk.cvut.cz/publicDatasets/CTU-M...	2017-06-24_win5
79	80	https://mcfp.felk.cvut.cz/publicDatasets/CTU-M...	2017-06-24_win4

80 rows × 3 columns

Figura 6.1. Ejemplo de CSV generado en fase 1

En la fase 2 se realizó el montaje del laboratorio de investigación, para lo que se utilizó una distribución Linux (Ubuntu) y las herramientas: NFPCAPD y NFDUMP, como se aprecia en la Figura 6.2.

El laboratorio permite las siguientes funcionalidades: transformar un archivo pcap a un conjunto de archivos NFPCAP; renombrar el conjunto de archivos NFPCAPS a una secuencia de números consecutivos; transformar el conjunto

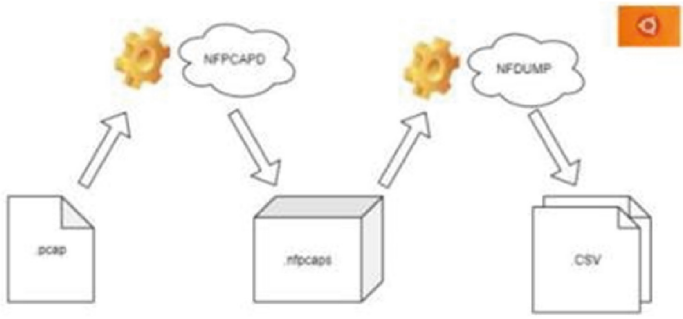

Figura 6.2. Montaje del laboratorio de investigación

de archivos NFPCAPS en archivos CSV; y comprimir los CSV en un solo archivo a partir de un algoritmo generador que permite generar las variables descritas en el entendimiento de los datos de CRISP-DM. Por su parte, la fase 3 se dedicó a la analítica de datos.

En la fase cuatro se implementó una aplicación web que contemplaba los requerimientos funcionales y un diagrama de *deployment* creado a partir de ellos. De acuerdo con los requerimientos funcionales, la aplicación debe permitir:

- tener un módulo de minería de datos con las siguientes funcionalidades:
  - permitir correr un proceso de NFDUMP con el fin de extraer datos de la red local,
  - convertir el archivo PCAP generado cada determinada cantidad de paquetes en un conjunto de archivos NFPCAPS,
  - convertir el conjunto de archivos NFPCAP a archivos CSV,
  - transformar los archivos CSV a un solo archivo CSV que presente las variables mencionadas en la fase de entendimiento de los datos de la metodología CRISP-DM, y
  - eliminar todos los archivos generados que ya hayan sido analizados por el módulo de ciencia de datos;
- tener un módulo de ciencia de datos capaz de generar una predicción, cada determinada cantidad de paquetes, con los datos generados por el módulo de minería;
- enviar una notificación al correo electrónico ingresado, en caso de que la predicción realizada por el módulo de ciencia de datos sea maliciosa;

- persistir en una base de datos no relacional: fecha y hora, e-mail, predicción y mensaje; y
- por medio de servicios REST, consultar el historial de predicciones, iniciar el servicio, detener el servicio, ingresar el correo electrónico de la persona que recibirá las notificaciones y generar un reporte del historial de predicciones

El diagrama de *deployment*, que presenta las interacciones entre los módulos de programación, se puede descargar de los archivos del proyecto en Github [13].

En cuanto a las tecnologías empleadas: para el desarrollo de la fase relacionada con la analítica de datos, se utilizó la herramienta denominada Jupyter Notebook, la cual permitió crear un conjunto de *notebooks* para registrar el proceso realizado durante las fases del proyecto; para el desarrollo del *front* del aplicativo web, se utilizó Vue Js como *framework*. Algunas de las dependencias usadas en conjunto con el *framework* mencionada fueron: Axios, Vuetify, Vuex y ApexChart; y para el desarrollo del *back* del aplicativo web, se utilizó Flask, un *framework* que permite crear aplicaciones web integrando servicios REST por medio de lenguaje Python.

6.4. Resultados

Experimento 1

La distribución de datos utilizada en este experimento fue: 519 ventanas de tiempo legítimas y 800 ventanas de tiempo maliciosas, ambas de Stratosphere Lab [12]. Una vez realizada la limpieza de los datos, se aplicó la técnica *feature importance* para extraer las variables más representativas, ellas son:

1. First_sp
2. Avg_bps
3. P1_ib
4. Duration
5. Number_dp
6. Bytes
7. Number_sp
8. First_Protocol

9. P2_ib
10. First_dp
11. P3_ib
12. Netflows
13. P3_d
14. Second_Protocol
15. Type

Luego se separaron los datos de entrenamiento y prueba utilizando la técnica de *holdout* (70/30) y se procedió con la evaluación de los modelos de ML. Con el fin de seleccionar los mejores, se creó un archivo donde se pueden apreciar, en orden descendente, la *accuracy* de los modelos. Los resultados se presentan en la Tabla 6.3 (entrenamiento) y en la Tabla 6.4 (prueba).

Los dos mejores modelos de este experimento fueron *random forest*, con una *accuracy* de 99,74 % y un KAPPA del 99.45 %, con una configuración de criterion = "Gini", max_depth = "5", n_stimators = "64"; y k-NN, con

Tabla 6.3. Métricas del experimento 1: entrenamiento

Índice	Modelo	Accuracy	CV
1	k-NN	99,78 %	10
2	*Random forest*	99,78 %	10
3	Árboles de decisión	99,67 %	10
4	Regresión logística	98,04 %	10
5	*Naive* Bayes gaussiano	93,93 %	10

Tabla 6.4. Métricas del experimento 1: prueba

Índice	Modelo	Accuracy	Kappa	CV
1	*Random forest*	99,74 %	99,45 %	10
2	k-NN	99,49 %	98,91 %	10
3	Árboles de decisión	98,98 %	97,83 %	10
4	*Naive* Bayes gaussiano	95,45 %	90,00 %	10
5	Regresión logística	90,90 %	79,55 %	10

una *accuracy* del 99.49 % y un KAPPA de 98.91 %, con una configuración de *metric* = "Manhattan", K = 5 y *weights* = "*distance*". Las configuraciones fueron extraídas utilizando la técnica de GridSearchCV.

Una vez obtenidos los mejores modelos, se procedió a probarlos con la muestra legítima destinada a realizar el intento de *oversampling*. Cabe recordar que se optó por utilizar *undersampling* debido a la extensa cantidad de datos maliciosos obtenidos.

Como se evidencia en la Figura 6.3, hubo una gran cantidad de falsos positivos predichos con el modelo k-NN: de 649 datos etiquetados como legítimos, el modelo predijo que tan solo 24 lo eran. Las métricas para este modelo se presentan en la Tabla 6.5.

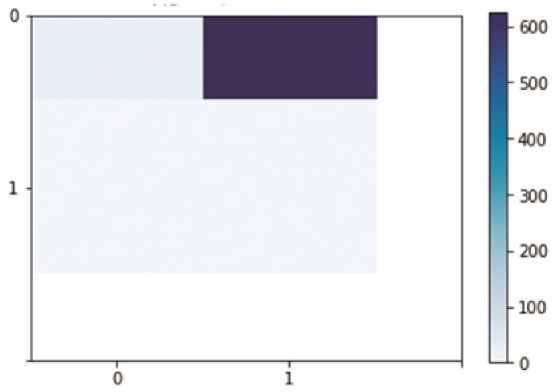

Figura 6.3. Experimento 1: matriz de confusión k-NN

Tabla 6.5. Experimento 1: métricas k-NN

Índice	Precision	Recall	F1-Score	Support
0	1.00	0.04	0.07	649
1	0.00	0.00	0.00	0
Accuracy			0.04	649
Macro avg	0.50	0.02	0.04	649
Weigted avg	1.00	0.04	0.07	649

Las predicciones con *random forest classifier*, en cambio, no fueron muy desacertadas: de 649 datos legítimos, el modelo predijo que 641 eran legítimos, es decir que solo generó 8 falsos positivos. Como se puede observar en la Figura 6.4, hubo una alta concentración de verdaderos positivos. Las métricas para este modelo se presentan en la Tabla 6.6.

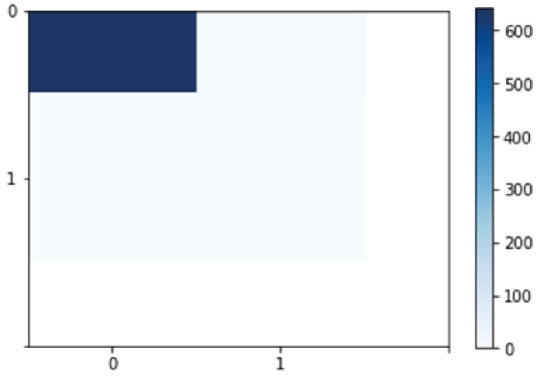

Figura 6.4. Experimento 1: matriz de confusión random forest

Tabla 6.6. Experimento 1: métricas random forest

Índice	Precision	Recall	F1-Score	Support
0	1.00	0.99	0.99	649
1	0.00	0.00	0.00	0
Accuracy			0.99	649
Macro avg	0.50	0.49	0.50	649
Weigted avg	1.00	0.99	0.99	649

El contraste es evidente: *random forest* consiguió una *accuracy* muy elevada (98,76 %), mientras que k-NN tuvo una muy baja (4 %). Se decidió entonces indagar por qué, para hacerlo se compararon los registros obtenidos en el laboratorio del proyecto, con los obtenidos por medio de Stratosphere Lab en cantidades iguales (519 por cada uno). Se compararon las variables filtradas por medio *feature importance*, las observaciones se resumen a continuación.

- La Figura 6.5 corresponde a la variable First_Protocol, el protocolo más usado, en ella se observa la existencia de protocolos que comparten ciertas similitudes. El protocolo UDP fue uno de los más utilizados, tanto en los datos de Stratosphere como en los del proyecto, se presenta en grandes cantidades; sin embargo, mientras el protocolo UDP tiene más registros en los datos de Stratosphere, TCP tiene más registros en los datos generados por el proyecto.

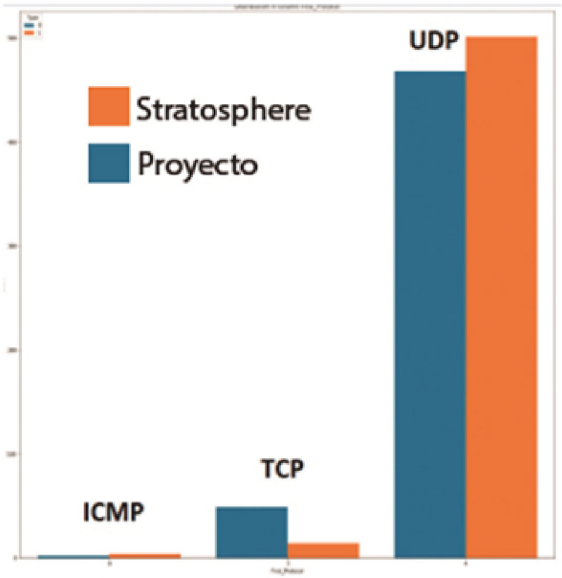

Figura 6.5. Comparación de datasets: variable First_Protocol

- La Figura 6.6 corresponde a la variable Second_Protocol, el segundo protocolo más evidenciado en las ventanas de tiempo, en ella las diferencias son más claras: existe una alta concentración de datos en el protocolo TCP, sin embargo, como se dijo, los datos del proyecto presentan más registros que Stratosphere; lo mismo sucede en los protocolos UDP e ICMP, sin embargo, Stratosphere presenta más registros de datos en el protocolo *none*, no evidenciado en los registros del proyecto. *None* es un protocolo no identificado por la herramienta de NFDUMP y es categorizado como un protocolo desconocido.

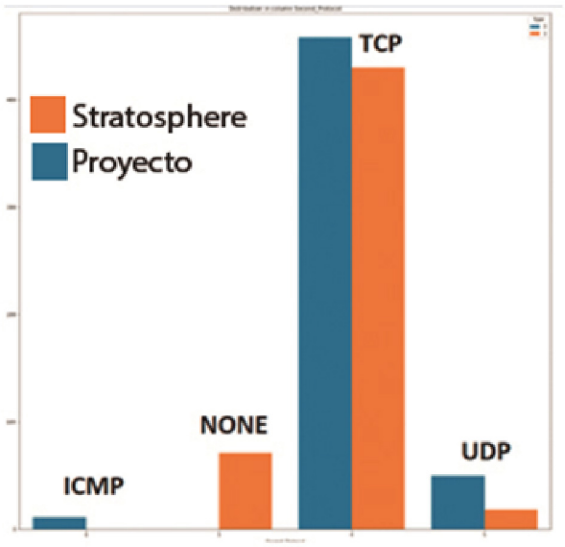

Figura 6.6. Comparación de datasets: variable Second_Protocol

- Las variables p3_ib, number_sp y number_dp, que representan el percentil 3 (75 %) de *bytes* de ingreso, número de puertos de origen y puertos de destino involucrados respectivamente, presentan diferencias muy claras: en p3_ib (Figura 6.7), el rango de los datos de Stratosphere oscila entre 0 y casi 3.500 *bytes* de entrada, mientras que en los datos generados en el proyecto, oscila entre 0 y casi 5.000 *bytes* de entrada; y en las variables number_sp (Figura 6.8) y number_dp (Figura 6.9), el rango de los datos

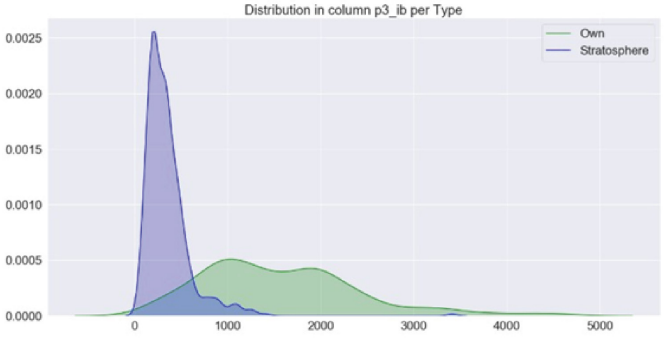

Figura 6.7. Comparación de datasets: variable p3_ib

generados por Stratosphere Lab oscila entre 0 y casi 7.000 puertos de origen, mientras que en los datos generados en el proyecto (verde), oscila entre 0 y casi 1.000 puertos de origen.

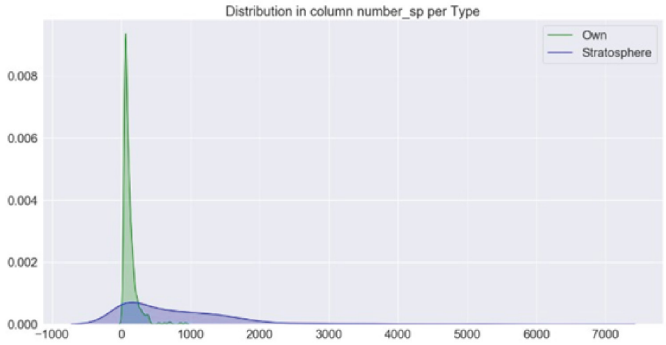

Figura 6.8. Comparación de datasets: variable number_sp

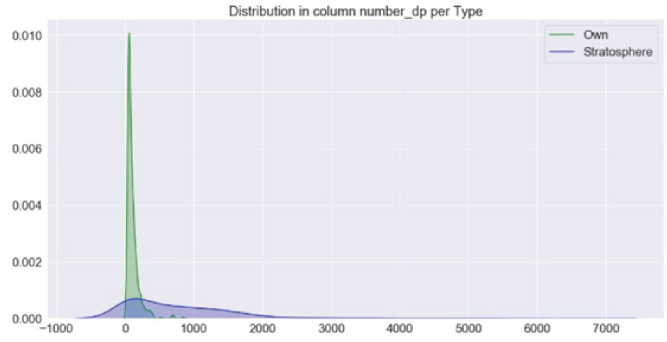

Figura 6.9. Comparación de datasets: variable number_dp

En conclusión, existen diferencias entre ambos *datasets* legítimos, por lo que en un siguiente experimento se incluyen registros de ambos laboratorios para así complementar los modelos generados.

Experimento 2

La distribución de datos utilizada en este experimento fue: 519 datos legítimos y 1.200 datos maliciosos de Stratosphere Lab [12]; y 649 datos legítimos del

laboratorio del proyecto. Una vez realizada la limpieza de los datos, se aplicó la técnica *feature importance* para extraer las variables más representativas, ellas son:

1. Avg_bps
2. Avg_pps
3. Bytes
4. P2_ib
5. Number_sp
6. First_Protocol
7. Number_dp
8. Duration
9. First_sp
10. Netflows
11. P3_ib
12. P3_d
13. Type

Luego se separaron los datos de entrenamiento y prueba utilizando la técnica de *holdout* (70/30) y se procedió con la evaluación de los modelos de ML. Con el fin de seleccionar los mejores, se creó un archivo donde se pueden apreciar, en orden descendente la *accuracy* de los modelos. Los resultados se presentan en la Tabla 6.3 (entrenamiento) y en la Tabla 6.4 (prueba). Como se puede apreciar, algunas variables que fueron representativas para el experimento 1, no lo son para el experimento 2, por lo que el número de variables contempladas ahora es menor.

Tal como sucedió en el experimento 1, se separaron los datos de entrenamiento y prueba utilizando la técnica de *holdout* y se procedió con la evaluación de los modelos de *machine learning* por medio de GridSearchCV. Con el fin de seleccionar los mejores, se creó un archivo donde se pueden apreciar, en orden descendente la *accuracy* de los modelos. Los resultados se presentan en la Tabla 6.7 (entrenamiento) y en la Tabla 6.8 (prueba).

Como se puede observar, *random forest* y k-NN continúan como los mejores modelos de clasificación, con una *accuracy* de 99,85 % y un Kappa de 99,71

Tabla 6.7. Métricas del experimento 2: entrenamiento

Índice	Modelo	Accuracy	CV
1	*Random forest*	99,69 %	10
2	k-NN	99,51 %	10
3	Árboles de decisión	99,45 %	10
4	Regresión logística	98,85 %	10
5	*Naive* Bayes gaussiano	88,14 %	10

Tabla 6.8. Métricas del experimento 2: prueba

Índice	Modelo	Accuracy	Kappa	CV
1	k-NN	99,85 %	99,71 %	10
2	*Random forest*	99,85 %	98,71 %	10
3	Regresión logística	98,29 %	98,58 %	10
4	Árboles de decisión	98,73 %	97.45 %	10
5	*Naive* Bayes gaussiano	89,42 %	78,68 %	10

%. Los parámetros arrojados por GridSearchCV son: n_neighbors = 1, *metric* = "Manhattan" y *weights* = "Uniform", para k-NN; y n_estimator = 64, max_features = "auto", max_depth = 6 y criterion = "Gini", para *random forest classifier*.

Con base en los resultados obtenidos en el experimento, se decidió analizar con mayor detalle las variables obtenidas por medio del *feature importance*. A continuación, analizaremos los patrones de comportamiento evidenciados en cada una de ellas.

- En las variables Avg-bps (*Average bits per second*) (Figura 6.10), Avg_pps (*Average packets per second*) (Figura 6.11) y Bytes (Figura 6.12), se puede evidenciar que los datos legítimos poseen un rango mucho mayor que el de los maliciosos, una comportamiento que se puede explicar porque las comunicaciones producidas por servidores de comando y control tienden a enviar periódicamente comandos de sincronización, pero no desean mantener conexiones que perduren en el tiempo, a diferencia de las comunicaciones normales que sí requieren mantener un flujo constante de datos.

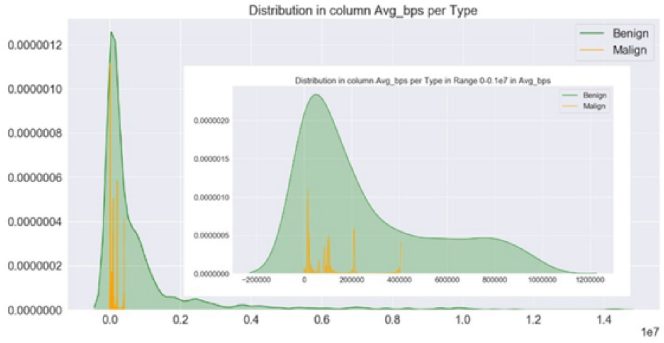

Figura 6.10. Comparación de datasets: variable Avg_bps

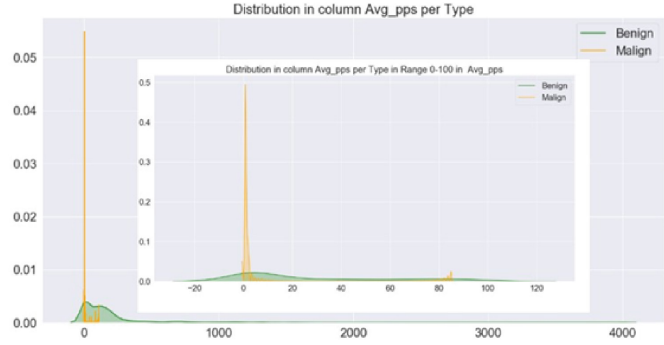

Figura 6.11. Comparación de datasets: variable Avg_pps

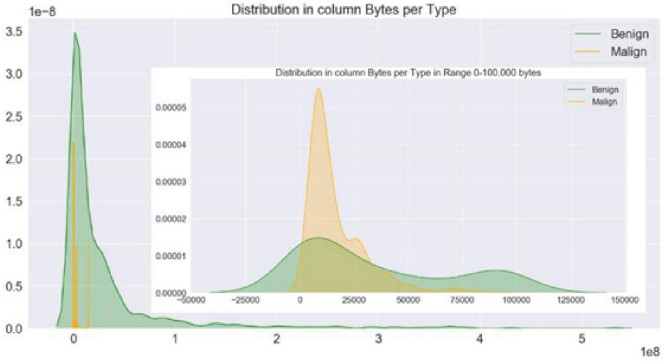

Figura 6.12. Comparación de datasets: variable Bytes

- Sumado a lo anterior, variables como duración (Figura 6.13) y número de *netflows* (Figura 6.14), poseen el mismo comportamiento de las variables mencionadas.

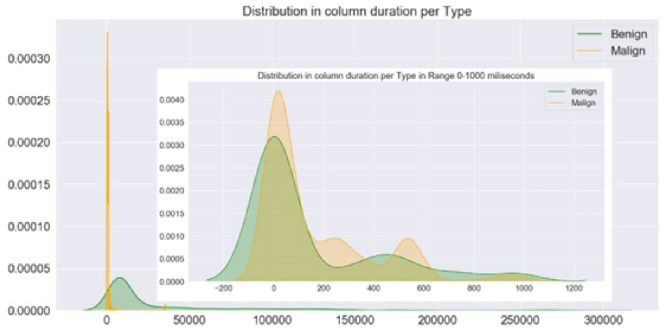

Figura 6.13. Comparación de datasets: variable Duration

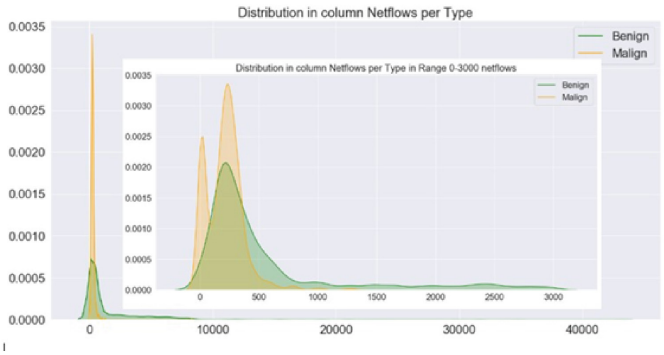

Figura 6.14. Comparación de datasets: variable Netflows

- En las variables que representan el número de puertos totales utilizados en las ventanas de tiempo (origen y destino), se observa que los flujos normales de datos utilizan un rango mucho más amplio de puertos, definidos en el tipo de comunicaciones y protocolos que desean utilizar, mientras que las comunicaciones maliciosas utilizan un número de puertos mucho más bajo, debido a que los cibercriminales de entrada conocen cuáles son los puertos más vulnerables, por lo que sus *exploits* están diseñados para trabajar sobre ellos (ver Figura 6.15 y Figura 6.16).

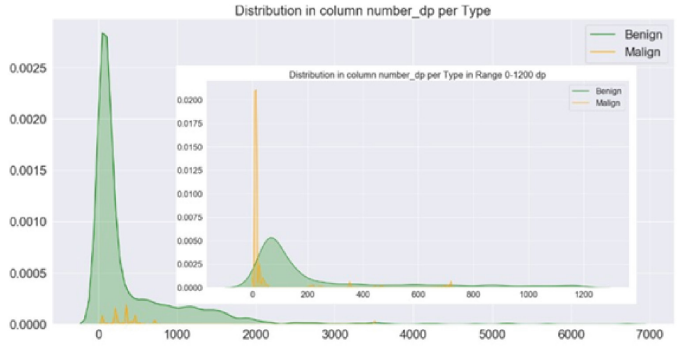

Figura 6.15. Comparación de datasets: variable number_dp

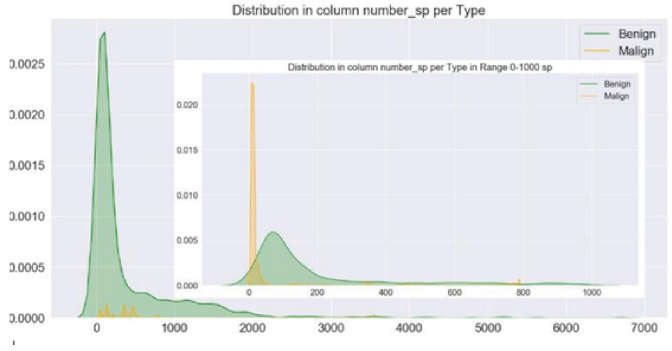

Figura 6.16. Comparación de datasets: variable number_sp

- En la variable First_Protocol (Figura 6.17), se evidencia que los datos legítimos utilizan protocolos como UDP, TCP e ICMP (este último en pocas cantidades) para la transmisión de datos, y que las comunicaciones maliciosas pueden transmitir su flujo de datos por medio de estos mismos protocolos, Sin embargo, la diferencia radica en que estos últimos también usan el protocolo ICMP6 y un protocolo no reconocido por NFDUMP.
- En las variables: p1_ib, percentil 1 de todos los *bytes* de entrada presentes en la ventana de tiempo (Figura 6.18); y p3_ib, percentil 3 de todos los *bytes* de entrada presentes en la ventana de tiempo (Figura 6.19), se evidencia que las comunicaciones maliciosas presentan un rango mucho mayor que los flujos normales de datos, lo que probablemente se debe a

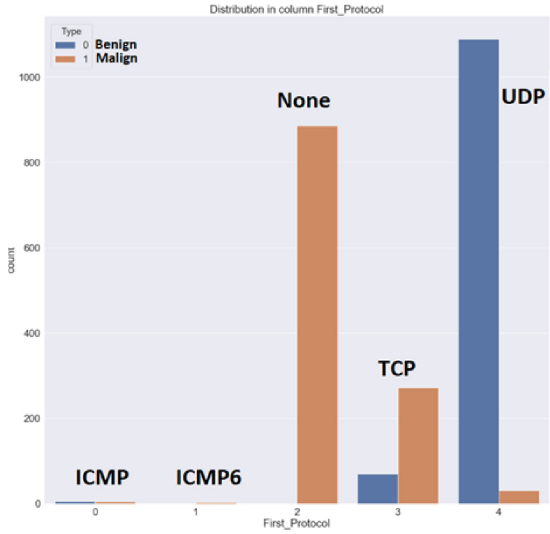

Figura 6.17. Comparación de datasets: variable First_Protocol

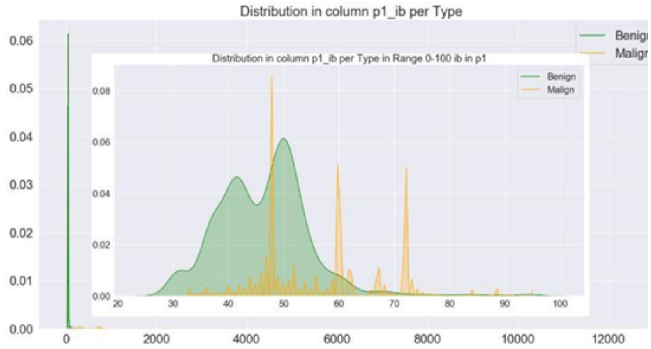

Figura 6.18. Comparación de datasets: variable p1_ib

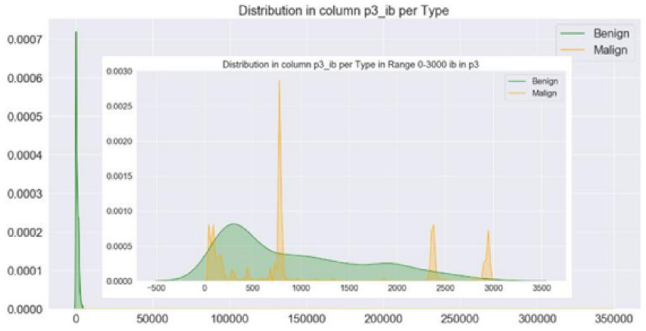

Figura 6.19. Comparación de datasets: variable p3_ib

que los servidores de C&C periódicamente pueden obligar a los "zombies" a realizar escaneos a las redes subyacentes, minar datos en el caso que fueran utilizados para minado de criptomonedas y transmitir información entre los nodos con el fin de sincronizarse y descargar nuevos archivos binarios, entre otras tareas.

- En la variable First_sp (puerto de origen más utilizado, Figura 6.20), se evidencia que los datos legítimos utilizan frecuentemente los puertos 53 (DNS), 443 (HTTPS), 80 (HTTP) y otros relacionados con protocolos TCP/UDP, mientras que los datos maliciosos usan primordialmente los puertos 0, 443 (HTTPS) y 8080 (HTTP). El puerto 0 es una forma abreviada de referirse a paquetes sin un encabezado L4, como TCP/UDP [14]. Ejemplos comunes de este tipo de tráfico son ICMP y el tráfico IP fragmentado como respuesta DNS que excede el tamaño máximo de 512 *bytes*.

  El puerto 0 es realmente efectivo para los ataques de agotamiento de ancho de banda DDoS [14], si la victima intenta bloquearlo, el equipo de reenvió de red puede rechazar la ACL, como referencia a un puerto no legítimo; se podría pensar que los servidores de C&C lo utilizan por esa misma razón. Asimismo, algunas aplicaciones pueden utilizar el puerto 0 para realizar un bind() de la conexión, lo que les permite obtener un puerto automáticamente designado por el sistema operativo que se encuentre disponible [15].

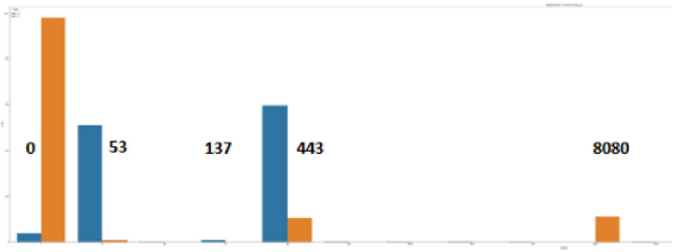

Figura 6.20. Comparación de datasets: variable First_sp

Experimento 3

En este experimento se realizaron tres instancias de prueba de una serie de *datasets* inyectada directamente en el aplicativo: en el primer caso (ver resultados

en la Tabla 6.9), se generaron muestras con 5.000, 10.000, 20.000 y 500.000 paquetes de datos legítimos; para el segundo, se extrajo una muestra del *dataset* del laboratorio Stratosphere [12], garantizando que nunca pasaron por el proceso de entrenamiento de los modelos (ver resultados en la Tabla 6.10); para el tercero, se trabajó con un *dataset* proveniente de la Universidad de New Brunswick [16] (ver resultados en la Tabla 6.11).

Tabla 6.9. Experimento 3: resultados con muestras legítimas

Índice	Paquetes (#)	Árboles de decisión	Random forest	Regresión logística
1	5.000	Bien	Bien	Bien
2	10.000	Bien	Bien	Bien
3	20.000	Bien	Mal	Bien
4	500.000	Bien	Bien	Bien

Tabla 6.10.Experimento 3: resultados con muestras maliciosas obtenidas del dataset de Stratosphere Lab

Índice	Fuente	Árboles de decisión	Random forest	Regresión logística
1	https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-204-1/	Bien	Bien	Bien
2	https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-195-1/	Bien	Bien	Bien
3	https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-192-1/	Bien	Bien	Bien
4	https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-179-1/	Bien	Bien	Bien
5	https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-169-1/	Bien	Bien	Bien

Tabla 6.11. Experimento 3: resultados con muestras maliciosas obtenidas del dataset de la Universidad de New Brunswick

Índice	Fuente	Árboles de decisión	Random forest	Regresión logística
1	http://205.174.165.80/CICDataset/ISCX-Bot-2014/Dataset/	Bien	Bien	Bien

Aplicativo Web

Una contribución importante de este proyecto es el desarrollo del aplicativo web, lo que le permite a los investigadores reemplazar los modelos de *machine learning* vistos en este proyecto, con nuevos modelos entrenados con datos actualizados, es decir, facilita reutilizar las herramientas, evitando así su obsolescencia. Mediante el aplicativo web, los usuarios pueden: iniciar sesión con su correo electrónico, el cual se utilizado con el fin de notificarles cuándo un escaneo evidencia comportamiento anómalo; iniciar y detener el escaneo personalizado; y visualizar el historial de predicciones en una lista y un gráfico (*pie*)[13].

6.5. Conclusiones y recomendaciones

El uso de las 259.346 ventanas de tiempo, cada una con una duración de cinco minutos, permitió reducir el almacenamiento y procesamiento de los datos (a diferencia de trabajar con *netflows*), lo cual permitió entrenar modelos de *machine learning* con una tasa de detección que oscila entre 90.4 % y 99.8 %.

El experimento 2 permitió llegar a unos modelos más robustos, a través de su entrenamiento y mediante el uso de 1.168 ventanas de tiempo legítimas y 1.200 ventanas de tiempo maliciosas, se logró una tasa de detección que oscila entre 89.4 % y 99.8 %. Sumado a lo anterior, se concluyó que los mejores modelos fueron: k-NN y *random forest*, que lograron una precisión de 99.8 % y un coeficiente Kappa de 99.7 %.

Mediante el uso de las trece variables presenciadas en el experimento 2, las cuales comprometían características como: promedio de bits por segundo, número total de *bytes*, duración de la ventana de tiempo, número de *netflows*, protocolo más usado, puerto origen más usado y número total de puertos origen y destino usados, fue posible evidenciar que las conexiones maliciosas: conocen los puertos más vulnerables; no perduran en el tiempo; envían tráfico sin encabezado a través del puerto 0, como técnica para realizar ataques de denegación de servicios distribuida (DDoS). Todo lo anterior permite tener diferencias muy claras entre los patrones de comportamientos maliciosos y normales.

A continuación, se listan las recomendaciones para trabajo futuro.

- Desarrollar herramientas que permitan la transformación de archivos PCAP a ventanas de tiempo en los sistemas operativos Windows y Mac OS, pues actualmente el aplicativo web únicamente funciona para

sistemas con distribución Linux. Si bien esta limitación está justificada (el laboratorio utiliza la herramienta NFPCAPD, presente en NFDUMP), el desarrollo sugerido permitiría abarcar una mayor cantidad de equipos empresariales y personales.

- Plantear y realizar experimentos como el realizado en este proyecto, pero utilizando aprendizaje no supervisado, así se podría universalizar los hallazgos y encontrar similitudes y diferencias entre los patrones de comportamiento evidenciados por el proyecto. Esta recomendación tiene sentido, toda vez que en la presente investigación se usó únicamente aprendizaje supervisado para las predicciones.
- Analizar una mayor cantidad de datos y obtener así nuevos y mejores resultados en la detección de patrones de comportamiento en el aplicativo.
- Usar ventanas de tiempo más amplias, para mejorar el entrenamiento de los modelos de *machine learning*.
- Continuar realizando investigaciones de este tipo, debido a que los ciberdelincuentes no se detienen y cada día están buscando nuevas vulnerabilidades en los sistemas operativos, los cuales podrían usar a su favor para generar nuevos *exploits*.

Los recursos (documentos, gráficos, códigos, etc.) obtenidos durante la ejecución del proyecto están disponibles en [17], con excepción de la presentación del proyecto realizada durante la Dragon JAR Conference, disponible en [18].

Referencias

[1] S. García, (2014, nov.), "Identifying, modeling and detecting botnet behaviors in the network," tesis de doctorado, Buenos Aires, Argentina: Universidad Nacional del Centro de la Provincia de Buenos Aires. Disponible: https://doi.org/10.13140/2.1.3488.8006

[2] A. Mushtaq. (2012, jul.). *Grum botnet: no longer safe havens*. Disponible: https://www.fireeye.com/

[3] J. Jiménez. (2019, ago.). *Nueva variante Echobot*. Disponible: https://www.redeszone.net/

[4] D. Jain. (2019, jun. 18). NSFocus shares botnet trends in new 2018. *Insights Report*. Disponible: https://nsfocusglobal.com/nsfocus-shares-botnet-trends-new-2018-insights-report/

[5] P. Minark. (2016). *How to analyze and understand your network*. Disponible: https://s3.eu-central-1.amazonaws.com/cms-api-beta.mgw.cz/flowmon/edefcfaf-5dc3-499a-8c97-66f5ebbebd94.pdf.

[6] H. R. Zeidanloo, M. Jorjor, Z. Shooshtari, P. V. Amoli, M. Safari y M. Zamani, "A taxonomy of botnet detection techniques," en *2010 3rd International Conference on Computer Science and Information Technology*, 2010, pp. 158-162, https://doi.org/10.1109/ICCSIT.2010.5563555.

[7] M. Bailey, E. Cooke, F. Jahanian, Y. Xu y M. Karir, "A survey of botnet technology and defenses," en *2009 Cybersecurity Applications & Technology Conference for Homeland Security*, 2009, pp. 299-304, https://doi.org/10.1109/CATCH.2009.40.

[8] J. Liu, Y. Xiao, K. Ghaboosi, H. Deng y J. Zhang, "Botnet: Classification, attacks, detection, tracing and preventive measures," en *EURASIP Journal on Wireless Communications and Networking*, 2009, dic. https://doi.org/10.1155/2009/692654

[9] S. Y. Lim y A. Jones, "Network anomaly detection system: The state of art of network behaviour analysis," en *2008 International Conference on Convergence and Hybrid Information Technology*, pp. 459-465, https://doi.org/10.1109/ICHIT.2008.249.

[10] A. K. Tyagi y G Aghila, "A wide scale survey on botnet," International *Journal of Computer Applications*, vol.34, No.9, pp. 10-23, 2011.

[11] J. Bhatia, R. Sehgal y S. Kumar, "Botnet command detection using virtual honeynet," *International Journal of Network Security & Its Applications*, vol. 3 No. 5, p. 177, 2011.

[12] *Stratosphere Research Laboratory* [portal]. Disponible: https://www.stratosphereips.org/

[13] J. C. Gaviria y A. Ramírez. Ciberseguridad. *i2tResearch* [GitHub]. Disponible: https://github.com/i2tResearch/

[14] T. Jones. (2013, ago. 27). *DDoS Attacks on Port 0 – Does it mean what you think it does?* Disponible: https://blog.endace.com/2013/08/27/ddos-attacks-on-port-0-does-it-mean-what-you-think-it-does/#:~:text=This%20special%20meaning%20of%20Port,making%20it%20impossible%20to%20block

[15] *Port 0 details*. Disponible: https://www.speedguide.net/port.php?port=0

[16] E. Biglar-Beigi, H. Hadian-Jazi, N. Stakhanova y A. A. Ghorbani, "Towards effective feature selection in machine learning-based botnet detection approaches," en *2014 IEEE Conference on Communications and Network Security*, 2014, pp. 247-255, https://doi.org/10.1109/CNS.2014.6997492

[17] C.C Urcuqui. Botnets. *i2tResearch* [repositorio GitHub]. Disponible: https://github.com/*i2tResearch*/Ciberseguridad_web/tree/master/Botnets

[18] J. Gaviria, A. Ramírez, y C. Urcuqui. *Analysis of time windows to detect botnet's behaviour* [video]. https://youtu.be/vJ9AzfudQK8

Capítulo 7
Secure learning para detección de Android malware

Jhoan Steven Delgado, Christian Camilo Urcuqui, Andrés Navarro, Javier Díaz Cely

7.1. Introducción

Android es el sistema operativo móvil más usado del mundo, su cuota de mercado para teléfonos inteligentes se estima entre 75 y 80 %. Es un sistema de código abierto (*open source*) que permite instalar aplicaciones APK (*Android Application Package*) tanto desde la tienda oficial (Play Store) como desde fuentes no oficiales. Su amplia cuota de mercado hace que sea muy apetecido por los cibercriminales, quienes pueden llegar a una gran cantidad de las personas [1]. De acuerdo con Tam et al. [2], la mayoría de los cibercriminales crea *malware* para obtener ganancias ilegales y una minoría (por ejemplo Anonymus) para realizar activismo político

Urcuqui et al. [1] describen los métodos para analizar el *malware* en Android y enfocándose en *machine learning* obtienen algunas características clave en la capa de red y en la capa de aplicación, las cuales son fundamentales para determinar si una APK es *malware* o no; asimismo, entrenan seis modelos de *machine learning* teniendo en cuenta estas características y concluyen que al evaluarlas sí es posible hacer la detección de Android *malware* utilizando técnicas de *machine learning*. Argumentan, sin embargo, que a pesar de que los clasificadores de *machine learning* son muy buenos, existe el riesgo de que puedan ser fácilmente engañados, haciéndoles pensar que una APK *malware* es goodware, suceso que pertenece al campo de investigación llamado *adversarial machine learning*. Barreno et al. [3] discuten que *machine learning* es una herramienta muy poderosa, que ha sido utilizada en muchas aplicaciones, incluyendo servicios web, agentes de servicio *online*, monitoreo de *cluster*, donde se evidencian patrones dinámicos de datos cambiantes. Asimismo, en una investigación posterior [4] proponen un marco de trabajo para la seguridad del aprendizaje en *machine learning*, ocho tipos de ataque y algunos mecanismos de defensa.

Por su parte Goodfellow et al. [5] indican que se ha descubierto que varios modelos de *machine learning*, incluyendo las redes neuronales, son vulnerables a muestras adversarias, es decir, que estos modelos pueden realizar mal la clasificación de las muestras obtenidas de la distribución. Mencionan que incluso, en la mayoría de los casos, diferentes modelos entrenados con subconjuntos del *dataset*, clasifican mal la misma muestra adversaria, lo que permite pensar que las muestras adversarias exponen puntos ciegos fundamentales de los algoritmos de entrenamiento de *machine learning*.

Solucionar estos problema reduciría la tasa de error de clasificación de los algoritmos de *machine learning* y aportaría confianza, pues una clasificación errónea hace que los usuarios desconfíen del modelo o aplicación y dejen de usarlo (en el mejor caso) o, si no lo notan, sigan usando el modelo (en el peor caso), lo que podría generar una catástrofe, debido a que se estaría clasificando algo malicioso como si fuese legítimo [4].

Las vulnerabilidades en los algoritmos de *machine learning* para la clasificación de Android *malware* [6] existen por varias razones: algunos de sus modelos son muy lineales (poco flexibles); se tiene como premisa que los *dataset* de entrenamiento y de evaluación pertenecen a la misma distribución estadística, por lo que los atacantes adversarios, en este caso apps maliciosas modificadas, pueden engañarlos [7]; o porque los modelos de *machine learning* se vuelven muy complejos (*overfitting*).

El objetivo general del proyecto es implementar *secure learning* a un modelo clasificatorio para detección de Android. Para lograrlo se definieron como objetivos específicos: entrenar y evaluar seis modelos de *machine learning* clasificatorios para la detección de Android *malware* a partir del tráfico de red; implementar un método de programación para realizar un *exploit* al mejor modelo de *machine learning* previamente entrenado; y proponer un conjunto de recomendaciones de *secure learning* para el mejor modelo de *machine learning* previamente entrenado.

7.2. Estado del arte

Se realizó la revisión de tres proyectos con propuestas similares a las del presente. Para efectos de comparación (ver Tabla 7.1), se utilizaron como criterios: el uso de *Reinforcement Learning* (RL) como técnica para la generación de datos adversarios; el uso de *adversarial training* para mejorar la robustez del modelo;

la propuesta de un *framework* para realizar un ataque adversario; la utilización de variables de tráfico de red para discriminar entre Android apps legítimas y maliciosas; y la inclusión de recomendaciones de *secure learning*.

Tabla 7.1. Comparativo de proyectos

Proyecto	RL	Adversarial Training	Attack framework	Network features	Secure learning
Yes, machine learning can be more secure: a case study on Android malware detection	No	No	No	No	No
Poster: towards adversarial detection of mobile malware	No	No	No	No	No
Android HIV: a study of repackaging malware for evading machine-learning detection	No	No	Si	No	No
Actual	Si	Si	Si	Si	Si

Yes, machine learning can be more secure: a case study on Android malware detection

Demontis et al. [6] exponen que aun cuando ML ha representado una gran ayuda para detectar y contrarrestar las crecientes y cambiantes amenazas en Android, tiene vulnerabilidades y es propenso a recibir ataques destinados a hacer que los modelos fallen al hacer una clasificación. Su trabajo está basado en un *framework* de ataque (análisis estático) propuesto en un trabajo previo para categorizar los potenciales escenarios de ataques en contra de los modelos de ML; luego, implementan un conjunto de ataques evasivos contra "Drebin", un modelo de ML útil para detectar Android *malware*, para evaluar de forma rigurosa su seguridad; finalmente, proponen un paradigma simple y escalable de *secure learning* que mitiga los ataques evasivos, aunque afecta un poco la tasa de detección sin los ataques.

Poster: towards adversarial detection of mobile malware

Chen, Xue y Xu [8] exponen que los clasificadores de ML convencionales fallan ante ciertos ataques, tomando en consideración los denominados *poisoning attacks*, en los cuales el atacante tiene control del *dataset* y puede inyectar

datos para confundir al modelo durante su entrenamiento, generando así una vulnerabilidad.

Los investigadores proponen un modelo adversario con tres tipos de ataques, para luego presentar un sistema iterativo de aprendizaje auto-adaptativo llamado KuafuDet (Figura 7.1), el cual cuenta con: una fase *offline*, en donde se seleccionan y se extraen las variables más significativas del conjunto de entrenamiento para entrenar el modelo; y una fase *online*, donde se realiza la clasificación con el modelo entrenado en la primera fase. Además, cuenta con un detector de camuflaje para detectar posibles inyecciones. Es el primer trabajo en detectar Android *malware* en un ambiente adversario, logra reducir la tasa de falsos positivos y mejorar la *accuracy* en al menos 15 %.

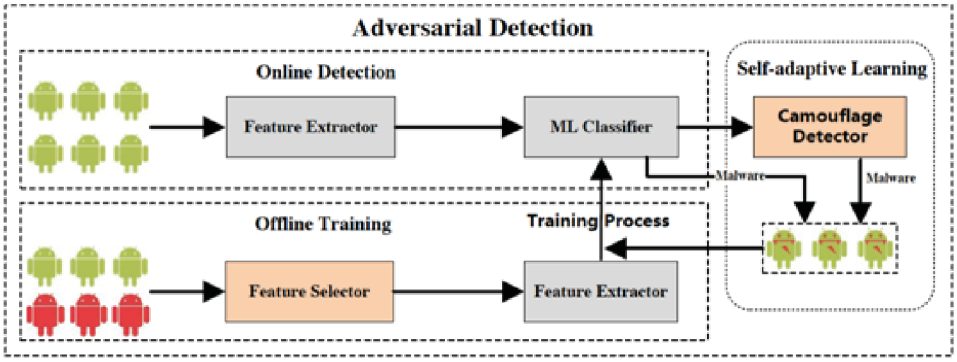

Figura 7.1. KuafuDet framework [8]

Android HIV: a study of repackaging malware for evading machine-learning detection

Chen et al [9] discuten el importante papel que cumplen los modelos de ML para realizar la detección de Android *malware*, pero indica también la poca robustez de estos modelos clasificatorios con respecto a los datos adversarios generados a través de pequeñas perturbaciones a los datos normales (los datos adversarios pueden engañar a modelos cuyas variables de entrada sean sintácticas, es decir, los permisos de la App, los llamados específicos del API, etc.). En el caso de los modelos con variables de entrada semánticas (por ejemplo, bytecode de Dalvik), los ataques con datos adversarios se vuelven más complejos, debido a

que realizar perturbaciones en el *bytecode* no es trivial, incluso una perturbación en el código podría afectar directamente la funcionalidad original de la App.

Los investigadores proponen un método de ataque para realizar perturbaciones óptimas en los apk generando datos adversarios. En contraste con los trabajos recientes, su método logra generar datos adversarios para algunos modelos con variables de entrada semánticas (por ejemplo, control-flow-graph y Dalvik). Al evaluar las manipulaciones hechas con los *datasets* de Drebin y MaMadroid, las tasas de detección disminuyeron de 97 y 96 %, respectivamente, a 1 %.

7.3. La investigación

Para este proyecto de investigación se decidió usar la metodología CRISP-DM (*Cross-Industry Estándar Process for Data Mining*) porque su estructura beneficia la resolución de problemas relacionados con las ciencia de datos, el aprendizaje de máquina y la minería de datos.

En la fase de entendimiento del negocio se analizaron las investigaciones reportadas en el estado del arte y el trabajo de Urcuqui et al. [1]. En la fase de entendimiento de los datos, se trabajó con el *dataset* de características de red utilizado por Urcuqui et al. [1], cuya cardinalidad es de 7.845 registros, para luego realizar una exploración y descripción de los datos.

El *dataset* cuenta con diez variables: (R1) paquetes TCP; (R2) paquetes distintos TCP; (R3) IP externas; (R4) volumen de *bytes*; (R5) paquetes UDP; (R6) paquetes de la aplicación fuente; (R7) paquetes de la aplicación remota; (R8) *bytes* de la aplicación origen; (R9) *bytes* de la aplicación remota; y (R10) consultas DNS, número de consultas DNS. Incluye además una variable adicional, llamada *type*, que puede tomar valores de *malicious* o *benign*.

En la fase de la preparación de los datos, se procedió a realizar su limpieza, eliminando los atípicos y las columnas donde tuviese "0" o "NaN" en más de la mitad de los datos. Finalmente se realizó el escalamiento (normalización) de los datos, para facilitar su manejo, y un análisis descriptivo de ellos.

En la fase de modelado, se entrenaron seis modelos (*naive* Bayes, *random forest*, k-NN, SVM, regresión logística y árboles de decisión, para lo que se realizó un *hold-out* con el 75 % del *dataset* de entrenamiento, el resto se usó para evaluar la capacidad de generalización de cada modelo.

En la fase de evaluación, se tuvo como un criterio de selección que la *accuracy* fuese mucho mayor que la *baseline*. Se obtuvieron y se tuvieron en cuenta durante este proceso otras métricas, a saber: matriz de confusión, precisión, *recall*, F1-score, *accuracy* y Kappa. Con base en estos criterios, se seleccionó el mejor modelo para dicho problema y se serializó en un archivo con extensión ".sav".

En la fase de despliegue, el modelo de ML seleccionado debe llevarse a producción, algo que no se realizó porque el propósito del proyecto es mejorar la robustez del modelo antes de llegar a esta fase.

Para realizar el *exploit* al modelo de defensa, se utilizó un enfoque *white-box*, en donde se tiene acceso al *dataset* con el cual el modelo fue entrenado, a la interacción con el modelo y a las variables. Como método de programación se utilizó *reinforcement learning*. El modelo de defensa y las variables del *dataset* de entrenamiento, representan el medio y las acciones en el algoritmo de RL. Al momento del ataque, se generó un total de 500 datos adversarios siguiendo el procedimiento que se describe a continuación.

- Se seleccionan las variables más significativas a atacar del modelo de defensa, teniendo en cuenta su *feature importance*, para disminuir la complejidad computacional al momento del ataque. *Feature importances*, método que hace parte de la librería Scikit-learn para Python, evalúa la importancia de cada variable del modelo al momento en que él realiza la clasificación [10].
- Se ajustan los parámetros correspondientes del algoritmo de RL, tales como: greedy_factor = 0.9, learning_rate = 0.1 y discount_factor = 0.9, y se crea un método (choose_action) con el cual se selecciona la acción (variable a atacar), según los valores en la Q-table.
- Se carga el mejor modelo de defensa (desde un archivo serializado ".sav"), y el *dataset* con cual el modelo fue entrenado.
- Se crea un *dataset* auxiliar con la misma estructura que el *dataset* de entrenamiento del modelo de defensa, con una muestra aleatoria de 500 datos maliciosos (*malicious*) tomados del *dataset* de entrenamiento del modelo de defensa, eliminando la columna de clasificación (*type*) del *dataset* auxiliar.
- Se modifica el algoritmo de RL para realizar perturbaciones, es decir, reemplazos en los valores, a las variables del *dataset* auxiliar con los 500

datos maliciosos. Estas perturbaciones tienen un rango específico por cada variable, en donde el mínimo es el dato mínimo del *dataset* de entrenamiento y el máximo, es el dato máximo del *dataset* de entrenamiento de cada variable, respectivamente. Las perturbaciones se aplican reemplazando de forma aleatoria el valor original de las variables del *dataset* auxiliar, dichas variables (acciones), como se mencionó, se escogen de acuerdo con los valores de la Q-table (la aleatoriedad está limitada al respectivo rango de cada variable)).

- Se crea un método (*get_env_feedback*) para asignar el respectivo *reward* para cada acción (variable a atacar) al momento de realizar una perturbación: positivo (+2), cuando el modelo de defensa previamente cargado, clasifica un registro del *dataset* auxiliar como *benign*; y Negativo (0) cuando es clasificado como la verdadera clase, es decir *malicious*.
- Se crea un método que imprime la cantidad de pasos que le tomó al algoritmo de RL para que el modelo clasificara un registro del *dataset* auxiliar como *benign* (es decir, convertir el registro en un dato adversario).
- Se ejecuta el algoritmo, se guardan los valores de la Q-table y los 500 datos adversarios generados en un archivo ".sav" y un archivo ".csv", respectivamente.

Adversarial training, por su parte, incorpora un *dataset* híbrido, que incluye los datos normales de entrenamiento con la adición de datos adversarios [11]. Para realizar el *adversarial training*, se tuvo en cuenta el trabajo de Wang et al. [11], quienes exponen que este método fue uno de los primeros usados para contrarrestar los ataques adversarios en contra de los modelos de ML. Este método puede mejorar la robustez de un modelo de ML con respecto a los ataques adversarios.

El procedimiento para realizar *adversarial training* (Figuras 7.2 a 7.4) es el siguiente:

- se carga tanto el *dataset* de entrenamiento y el *dataset* de los 500 datos adversarios;
- se adiciona la columna de clasificación (*type*) en el *dataset* de los datos adversarios con la etiqueta *malicious*;
- se realiza un *hold-out* con los 500 datos adversarios, dejando el 75 % para entrenamiento y el otro 25 % para evaluación;
- se agregan los datos adversarios (el 75 %) al *dataset* de entrenamiento;

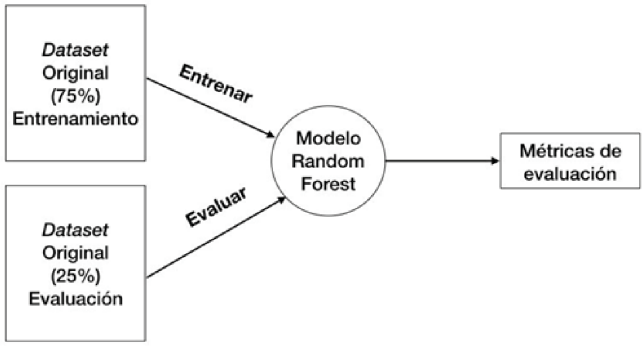

Figura 7.2. Entrenamiento y evaluación del modelo de defensa

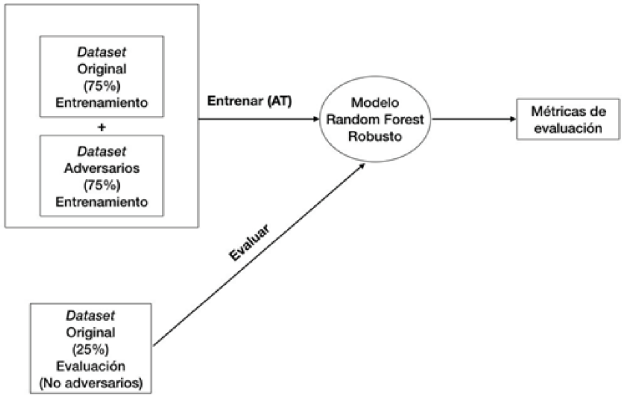

Figura 7.3. Adversarial training y verificación con el conjunto de evaluación del dataset original

Figura 7.4. Evaluación con solo datos adversarios

- se entrena un modelo de ML (*random forest*) con la agregación de los datos adversarios;
- se evalúa con el conjunto de evaluación del modelo de defensa para verificar que el nuevo modelo siga clasificando de manera correcta y no tenga un *bias* muy alto; y

- se evalúa utilizando el conjunto de evaluación (el del 25 %) de los datos adversarios con la técnica *cross-validation*, con *k=5*, para verificar su robustez respecto de nuevas muestras adversarias.

7.4. Resultados

El proyecto, en su etapa de experimentación, utilizó Python 3.7.1 y Jupyter Notebook, como lenguajes de programación, y la librería de *machine learning* scikit-learn 0.20.1.

Durante la fase de preparación de los datos: se realizó la limpieza de 7.845 registros, en el proceso se eliminaron trece datos atípicos, de los 7.832 restantes, 4.691 eran del tipo *benign* y 3.141 *malicious*; y se normalizaron todas las variables independientes utilizando el método RobustScaler de la librería scikit-learn.

En las capturas de tráfico de red se evidencia que el promedio de la cantidad de paquetes TCP enviados es de 1,72 *benign* y 0,55 *malicious*. Asimismo, se evidencia que la variable remote_app_packets tenía en promedio 2,35 de la clase *benign* y 0,86 *malicious*.

Posteriormente, en la fase de modelado, se reentrenaron y evaluaron seis modelos de *machine learning* clasificadores de *malware* Android, con las diez características de red ya mencionadas como variables predictivas, a partir de los datos utilizados por Urcuqui et al. [1]. El desempeño de los seis modelos se resume en la Tabla 7.2.

Como se observa en la Tabla 7.2, los mejores resultados fueron obtenidos con *random forest*. Se procedió entonces con el ajuste de algunos parámetros,

Tabla 7.2. Desempeño individual de los modelos

Algoritmo	Precision		Recall		F1-Score		Accuracy	Kappa
	benign	malicious	benign	malicious	benign	malicious		
Naive Bayes	0.81	0.41	0.12	0.96	0.20	0.58	0.44	0.06
Random forest	0.93	0.90	0.94	0.88	0.93	0.89	0.91	0.82
k-NN: K=4	0.89	0.89	0.93	0.83	0.91		0.89	0.77
SVM	0.72	0.79	0.92	0.45	0.81	0.57	0.73	0.39
Regresión logística	0.72	0.68	0.86	0.47	0.78	0.56	0.70	0.34
Árboles de decisión	0.91	0.86	0.91	0.85	0.91	0.86	0.88	0.76

así: n_estimators = 250; max_depth = 50; y random_state = 45. Cabe resaltar que se logró un mejor desempeño que el reportado por Urcuqui et al. [1]: la accuracy pasó de 0.89 a 0.91 y el Kappa de 0.75 a 0.82.

De este mejor modelo se obtuvieron las variables más importantes consideradas al realizar la clasificación (ver Tabla 7.3); la matriz de confusión (Figura 7.5); y la curva ROC (Figura 7.6).

En la matriz de confusión, instrumento que permite visualizar los errores de clasificación del modelo, se observan 1.117 registros legítimos y 69 maliciosos clasificados correctamente (ver diagonal principal); y 73 registros legítimos y 89 maliciosos mal clasificados (ver la otra diagonal). Por su parte, en la curva ROC, una métrica donde a partir de graficar las razones de los falsos positivos y los falsos negativos, se puede determinar qué tan bien clasifica un modelo, se observa un 0,91, una excelente medida, si se tiene en cuenta que un modelo es mejor en la medida en que ese valor se acerque a 1 [12].

Tabla 7.3. Feature importance (random forest)

	Feature	Importancia
1	source app bytes	0.221418
2	remote app bytes	0.147509
3	tcp packets	0.142017
4	remote app packets	0.134524
5	vulume bytes	0.127661
6	source app packets	0.086725
7	dns query times	0.050267
8	dist port tcp	0.045439
9	external ips	0.043016
10	udp packets	0.001425

		Predicción	
		Legítimo	Malicioso
Realidad	Legítimo	1.117	73
	Malicioso	89	679
	Accuracy = 0.9173; misclass = 0.0827		

Figura 7.5 . Matriz de confusión del modelo de defensa

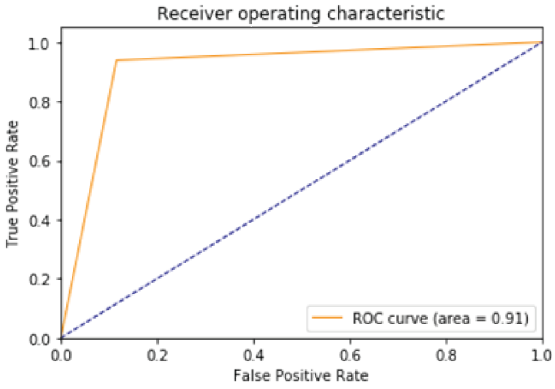

Figura 7.6. Curva ROC del modelo de defensa

Como se mencionó, *feature importances* permite identificar las variables importantes para que *random forest* pueda tomar decisiones al momento de clasificar. *Feature importances* asigna un peso a cada variable (la suma del peso de cada una debe ser igual a 1). En la Tabla 7.3 se evidencia que source_app_bytes es la variable de mayor importancia y udp_packets, la de menor relevancia. De este grupo de diez, el proyecto tomó las seis primeras, para reducir costos computacionales.

Para realizar un *exploit* al modelo previamente entrenado, se procedió con la generación de un *dataset* auxiliar conformado por quinientas instancias, basado en perturbaciones de las seis variables seleccionadas de la Tabla 7.3. Se encontró que luego de generar veinte datos adversarios, solo fueron necesarios cinco pasos para realizar una perturbación exitosa, valor que se mantuvo estable hasta llegar a la generación de los quinientos datos adversarios. En los datos generados previamente se requirieron entre seis y quince pasos, lo que evidencia la mejor eficiencia y la mayor eficacia de la Q-table al momento de elegir una acción, a medida que se iban generando nuevos datos adversarios.

Después de realizar el *adversarial training*, sus métricas (Tabla 7.4) se compararon con las del conjunto de evaluación del modelo de defensa (Tabla 7.2), para verificar que continuara clasificando de manera correcta. Efectivamente, como se puede observar, las métricas de ambas tablas son similares, lo que indica que el modelo mantiene su buen nivel de desempeño en clasificación; incluso, se observa una ligera mejora en: la *accuracy* (0,9183 vs 0,9173); y en la matriz de confusión (Figura 7.7), con cuatro clases más, clasificadas correctamente.

		Predicción	
		Legítimo	Malicioso
Realidad	Legítimo	1.120	70
	Malicioso	90	678
	Accuracy = 0.9183; misclass = 0.0817		

Figura 7.7. Matriz de confusión luego del adversarial training

Tabla 7.4. Desempeño de random forest luego del adversarial training

Algoritmo	Precision		Recall		F1-Score		Accuracy	Kappa
	benign	malicious	benign	malicious	benign	malicious		
Random forest robusto	0.93	0.91	0.94	0.88	0.93	0.89	0.91	0.82

Luego de verificar que aún era un buen modelo, se procedió con su evaluación frente al conjunto de evaluación de los datos adversarios (25 %), mediante la técnica de *cross-validation* con un $k = 5$.

Como se puede observar en la Tabla 7.5, pasó de no reconocer los datos adversarios a reconocer aproximadamente el 71 % de ellos en promedio. Si se tiene en cuenta que este 25 % de los datos adversarios no se encontraba en el *dataset* híbrido de entrenamiento, se puede concluir que este modelo es ahora mucho más robusto frente a ataques adversarios.

Tabla 7.5. Cross-validation para la evaluación respecto de los datos adversarios

Cross validation	Accuracy
k = 1	0,77
k = 2	0,73
k = 3	0,75
k = 4	0,71
k = 5	0,59
k	0,71

7.5. Conclusiones y recomendaciones

En el proyecto se implementó el concepto de *secure learning* a un modelo que permite predecir la clasificación de las APK Android, como legítimas o maliciosas, a partir de las variables de tráfico red propuestas por Urcuqui et al. [1].

Luego del *adversarial training*, el modelo de ML basado en *random forest* mejoró ligeramente sus métricas y pasó de no reconocer los datos adversarios, a reconocer tres cuartas partes de ellos (77 %), lo que claramente da cuenta de un modelo mucho más robusto ante ataques adversarios.

Del proceso realizado se concluyó además que es posible realizar un *exploit* a un modelo clasificatorio de Android *malware* entrenado con características de tráfico de red utilizando RL. A través del ajuste de algunos parámetros, tales como: n_estimators, max_depth y random_state, se mejoró la *accuracy* del modelo de defensa.

En el desarrollo del proyecto, se trabajaron dos objetivos de la ciberseguridad: integridad, porque mejorar la robustez del modelo reduce la cantidad de falsos positivos y falsos negativos; y disponibilidad, porque si el modelo es un poco más seguro no estará fuera de servicio por realizar clasificaciones erróneas. Sin embargo, cuando se lleve a producción (*deployment*), se debe tener en cuenta también un tercer objetivo de la ciberseguridad: la confidencialidad. Esto implica muchas más cosas (como el sistema y el medio donde interactúa), no solamente el modelo en cuestión. En consecuencia, para que un modelo de ML sea menos proclive a ataques adversarios, se deben tener en cuenta las siguientes recomendaciones:

- agregar una nueva actividad de defensa proactiva en la fase de evaluación de la metodología CRISP-DM, donde el diseñador del modelo desarrollé estrategias de defensa ante posibles ataques adversarios, antes de su despliegue, para reducir así su vulnerabilidad;
- evitar que el *dataset* de entrenamiento, junto con la interacción del modelo de ML, estén disponibles al público en general, para evitar ataques de *white-box* y *poisoning*;
- entrenar los modelos de ML con algunos datos adversarios (que se pueden generar con el método desarrollado en el proyecto actual), para así mejorar su robustez y confidencialidad; y

- al llevar un modelo a producción (*deployment*), verificar que sean seguros, no solo el sistema, sino también el medio en que el modelo interactúa.

Como trabajo futuro se sugiere:

- mejorar el algoritmo de *reinforcement learning* para realizar perturbaciones más precisas utilizando *Deep Reinforcement Learning* (DRL) o *Generative Adversarial Networks* (GAN), como proponen Goodfellow et al. [13];
- proponer un método de programación para realizar un *exploit* de un modelo de ML con un enfoque *black-box*; y
- llevar a producción el modelo robusto (*random forest*) del proyecto actual, desplegándolo en un servidor del grupo de investigación i2T de la Universidad Icesi, para que clasifique Android *malware* a través del tráfico de red, incorporado en el aplicativo móvil Sniff.

Los recursos (documentos, gráficos, códigos, etc.) obtenidos durante la ejecución del proyecto están disponibles en [14], con excepción de las presentaciones del proyecto realizadas durante el Dragon JAR Conference 2020, PyCon Colombia 2021 y BSides 2021, disponibles en [15], [16] y [17], respectivamente.

Referencias

[1] C. Urcuqui, J. Delgado, A. Perez, A. Navarro, y J. Diaz, "Features to detect Android malware," *2018 IEEE Colombian Conference on Communications and Computing* (COLCOM), 2018.

[2] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, y L. Cavallaro, "The evolution of Android malware and Android analysis techniques," *ACM Computing Surveys*, vol. 49, no. 4, pp. 1–41, 2017.

[3] M. Barreno, A. D. Joseph, y J. D. Tygar, "Can machine learning be secure?," en *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, pp. 16–25, 2006.

[4] M. Barreno, B. Nelson, A. D. Joseph, y J. D. Tygar, "The security of machine learning," *Machine Learning*, vol. 81, no. 2, pp. 121–148, 2010.

[5] I. J. Goodfellow, J. Shlens, y C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv*:1412.6572, 2014.

[6] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, y F. Roli, "Yes, machine learning can be more secure! A case study on Android malware detection," *IEEE Transactions on Dependable and Secure Computing*, vol. 16. No. 4. pp. 711–724, 2017.

[7] I. Goodfellow, P. McDaniel, y N. Papernot, "Making machine learning robust against adversarial inputs," *Communications of the ACM*, vol. 61, No. 7, pp. 56–66, 2018.

[8] S. Chen, M. Xue, y L. Xu, "Towards adversarial detection of mobile malware [poster]," en *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, pp. 415-416, 2016.

[9] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, y K. Ren, "Android HIV: A Study of Repackaging Malware for Evading Machine-Learning Detection," *arXiv*:1808.04218 [cs.CR], 2018

[10] *Feature importances with forests of trees*. Disponible: https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html#:~:text=Feature%20importances%20are%20provided%20by,impurity%20decrease%20within%20each%20tree.&text=Impurity%2Dbased%20feature%20importances%20can,features%20(many%20unique%20values).

[11] X. Wang, J. Li, X. Kuang, Y.-a. Tan, y J. Li, "The security of machine learning in an adversarial setting: a survey," *Journal of Parallel and Distributed Computing*, vol. 130, pp. 12–23, 2019.

[12] S. Narkhede. (2018, jun. 26). *Understanding AUC - ROC Curve*. Disponible: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

[13] I. J. Goodfellow, J. Pouget, M. Mirza, B. Xu, D. Warde, S. Ozair, A. Courville, y Y. Bengio, "Generative adversarial networks," *arXiv*:1406:2661[stat.ML], 2014.

[14] C. Urcuqui. (2019). SL para detección de Android Malware (PDG). *i2tResearch* [repositorio GitHub]. Disponible: https://github.com/i2tResearch/Ciberseguridad_web/tree/master/Android/SL%20para%20detecci%C3%B3n%20de%20Android%20Malware%20(PDG)

[15] J. Delgado, y C. C. Urcuqui. (2020). *Hackeando el aprendizaje de máquina (machine learning)* [video]. Disponible: https://youtu.be/9XongQYLc3o

[16] J. Delgado, y C. C. Urcuqui. (2021). *Hacking machine learning* [video]. Disponible: https://youtu.be/vvGp6n69rIg

[17] J. Delgado, y C. C Urcuqui. (2021). *Hacking Machine Learning and how it leads us to make them less compromised* [video]. Disponible: https://youtu.be/6w3WDl7TxT4

Capítulo 8
Secure learning y deep reinforcement learning para la detección de Android malware

David Alejandro Huertas, Brayan José Vargas, Christian Camilo Urcuqui

8.1. Introducción

En 2020 Android alcanzó una cuota de 41 % en el mercado de sistemas operativos, lo que representa el primer lugar entre los sistemas operativos con mayor número de usuarios activos [1]; esta ventaja lo pone en la mira de la industria, por ello, cada vez es más común encontrar todo tipo de aplicaciones en su tienda oficial, la que ya supera los 2.9 millones de aplicaciones [2]. Este número no incluye las aplicaciones que no están disponibles en la Play Store, pero que se pueden encontrar navegando en Internet. Ese amplio mercado también es atractivo para los cibercriminales, quienes con variadas técnicas, entre ellas la creación de aplicaciones maliciosas (*malware*), buscan obtener desde los datos de la actividad de los usuarios, hasta el control total de sus dispositivos [3].

A raíz de la creciente amenaza del *malware*, se han desarrollado sistemas con modelos de *machine learning* capaces de identificar *malware*; sin embargo, aunque esto reduce el número de ataques exitosos que logran infectar los dispositivos, se han descubierto técnicas capaces de engañarlos y vulnerarlos. Eso ilustra la gran importancia de implementar medidas de seguridad en los sistemas de ML que mejoren su resistencia frente a este tipo de ataques.

El sesgo de los modelos de clasificación de *malware* Android como efecto de los ataques adversarios, se presenta cuando estos son vulnerados y se altera alguno de sus componentes: los datos de entrenamiento, la representación de los datos o el algoritmo de aprendizaje [4]. Esta alteración hace que el sistema presente una alta tasa de falsos positivos y falsos negativos, lo que a su vez puede repercutir en los usuarios finales, quienes al instalar una aplicación aparentemente segura: instalan un *malware* o una aplicación que es *malware*

pero no se detecta como tal; o dejan de instalar una aplicación legítima que se cataloga injustamente como *malware*.

Un anterior avance sobre la solución a este problema se desarrolló mediante un enfoque de aprendizaje seguro (*Secure Learning*, SL), en el cual se abordó el *adversarial training* desde un punto de vista en el que un cibercriminal tenía conocimiento sobre los datos de aprendizaje, la representación de las características y el algoritmo del modelo (*white-box*) [5]. A través de SL, y utilizando *Reinforcement Learning* (RL), se disminuyeron las vulnerabilidades del modelo de clasificación.

También se propuso para trabajos futuros un enfoque basado en el desconocimiento de uno a dos componentes (*gray-box*) o incluso una aproximación en la se desconozcan los tres componentes (*black-box*) y a través de *Deep Reinforcement Learning* (DRL) y *Generative Adversarial Networks* (GAN) se logren disminuir las vulnerabilidades del modelo clasificatorio [6]. Lo anterior es una motivación porque permite, desde una perspectiva más cercana al cibercriminal común, enfocar el SL para proteger modelos en los que se desconocen, parcial o totalmente, sus componentes.

Es importante considerar este problema, pues desde un enfoque más realista, no es común que los atacantes tengan acceso a toda una lista de información esencial de un sistema o de un modelo de aprendizaje. Aunque con el anterior proyecto [5], descrito en el capítulo anterior, se logró conseguir un modelo de clasificación mucho más robusto, esto no significa que sea suficiente o definitivo, ya que no contempla los escenarios en donde el atacante tiene como única alternativa realizar un ataque de *black-box* o *gray-box*. Así, como exponen Barreno et al. [7], es vital que se considere la robustez además de la precisión del modelo, para así garantizar un proceso de aprendizaje y clasificación mucho más confiable y seguro contra un rango más amplio de posibles ataques.

Actualmente existen vulnerabilidades a ataques con enfoque adversario en algoritmos de *machine learning* para la clasificación de *malware* Android. Las soluciones desarrolladas por proyectos anteriores se basan en entornos *white-box*, por lo que un amplio margen (*gray* y *black box*), queda sin cobertura, lo que representa un riesgo de seguridad para la industria y para los usuarios de dispositivos móviles con sistema operativo Android.

El objetivo general de este proyecto es reducir la vulnerabilidad a ataques con enfoque adversario en algoritmos de *machine learning* para la clasificación

de *malware* Android utilizando SL y RL. Para alcanzarlo se establecieron como objetivos específicos: proponer un modelo de ataque adversario con enfoque *gray-box*; evaluar un modelo de DRL para realizar perturbaciones más precisas en el proceso de aprendizaje del modelo; y proponer enfoques de desarrollo que cuenten con buenas medidas de seguridad.

8.2. Estado del arte

El actual proyecto propone un esquema de *secure learning* aplicado a un modelo clasificador de *malware* Android en un ambiente de cajas gris y negra utilizando aprendizaje profundo reforzado, teniendo en mente estas cuatro características, se revisaron cinco proyectos con propuestas similares. La Tabla 8.1 permite su comparación con base en estas variables. Los proyectos citados, se describen a continuación.

Tabla 8.1. Comparativo de proyectos

Proyecto	DRL	Gray box	Secure learning	Malware Classifier
Robust Android malware detection system against adversarial attacks using q-learning	No	Si	Si	Si
Deep reinforcement learning for black-box testing of Android apps	Si	No	No	No
Evading anti-malware engines with deep reinforcement learning	Si	Si	No	Si
Adversarial-example attacks toward Android malware detection system	No	Si	No	Si
Black box analysis of Android malware detectors	No	No	No	Si
Actual	Si	Si	Si	Si

Robust Android malware detection system against adversarial attacks using q-learning

Rathore, Sahay, Nikam y Sewak [8] señalan que aun cuando los modelos de ML y DL en los que se basan los sistemas de detección de *malware* en Android tienen un gran desempeño, suelen ser muy vulnerables a ataques de tipo adversario. En su investigación desarrollaron ocho modelos de detección de *malware* en Android basados en ML y DRL para poner a prueba su robustez

contra ataques de tipo adversario. Entre los ataques propuestos, uno fue del tipo *white-box*, un ataque de política única que busca modificar el vector de características extraído de una app maliciosa, de tal forma que llegue a ser identificada como legítima por el clasificador de tipo *gray-box*; y otro fue un ataque de política múltiple que consiste en la recolección de un conjunto de políticas a partir de Q-tablas para luego usarlas paralelamente en ataques de adversario. Se trata de la primera investigación de ataques de tipo adversario con un enfoque *gray-box* en el área de la detección de *malware*. Finalmente, se propone una estrategia defensiva contra este tipo de ataques que reduce significativamente la tasa de engaños promedio contra ataques de política única y múltiple, mejorando así la robustez general del sistema de detección de *malware* Android.

Deep reinforcement learning for black-box testing of Android apps

Romdhana, Merlo, Ceccato y Tonella [9] exponen que uno de los grandes retos en el desarrollo de aplicaciones móviles, debido a su creciente complejidad, es el diseño e implementación de pruebas de funcionamiento para ellas, y señalan que la implementación de una efectiva fase de pruebas es de suma importancia para minimizar la aparición de fallas durante la ejecución. Sin embargo, explican, los distintos procedimientos y exploraciones llevadas a cabo por pruebas automáticas no llegan a ser suficientes para cubrir la extensa cantidad y complejidad de los posibles estados de la aplicación. Los investigadores proponen la ejecución de pruebas con enfoque *black-box* basadas en DRL usando la herramienta ARES. Sus resultados muestran un alcance y revelación de fallas mayor en comparación con procedimientos convencionales de pruebas automáticas basadas en la exploración aleatoria, que si bien cubren gran parte del código y la detección de *bugs*, se pueden quedar cortas al tratar con transiciones más complejas.

Evading anti-malware engines with deep reinforcement learning

El objetivo de Fang, Wang, Li, Wu, Zhou y Huang [10] en este proyecto fue demostrar que los sistemas de clasificación de Android *malware* basados en aprendizaje supervisado son vulnerables, específicamente a técnicas adversarias. Para el entrenamiento de esta técnica se utilizó DRL, específicamente DQEAF (*Deep Q-network to Evade Antimalware engines Framework*), quien interactuó con los datos de entrenamiento y a través de ellos generó *malware* que el sistema de clasificación pasó por alto. Estos datos también fueron testeados en Virustotal,

evidenciando una mejora en la habilidad para evadir antivirus. Aunque este proyecto demostró la existencia de este tipo de vulnerabilidades en muchos sistemas de *machine learning*, su alcance no partió de un enfoque de caja negra o gris, sino que se le permitió al DQEAF conocer a profundidad el algoritmo de clasificación, es decir, se basó en un enfoque de caja blanca. Por otra parte, como el propósito del proyecto fue la demostración de la vulnerabilidad, no se realizaron aplicaciones de *secure learning* dirigidas a robustecer el modelo.

Adversarial-example attacks toward Android malware detection system

Li, Zhou, Yuan, Li y Leung [11] mencionan que así como los sistemas existentes de detección de *malware* en Android pueden ser burlados por ataques de tipo adversario basados en redes adversarias generativas (GAN), también se pueden defender fácilmente con la implementación de un cortafuego a un detector de *malware* Android conectado remotamente en la nube (ver Figura 8.1).

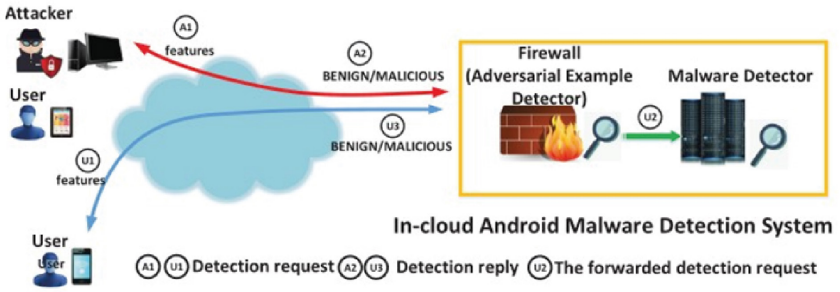

Figura 8.1. Sistema propuesto por Li et al. [11]

Los investigadores proponen un nuevo método de ataque, llamado E-MalGAN, el que intenta enviar una entrada maliciosa con la capacidad de eludir las mencionadas formas de detección de *malware*, a partir de su propuesta de variante de GAN, llamada *bi-objective* GAN, la cual es capaz de generar ejemplos de *malware* adversario con el objetivo de engañar tanto al *firewall* como al detector de *malware*. Sus resultados muestran que el 95 % de los ejemplos de *malware* adversario producidos por E-MalGAN evadieron exitosamente los dos detectores con que trabajaron en su investigación.

Black box analysis of Android malware detectors

Nellaivadivelu, Di-Troia y Stamp [12] reconocen el rápido crecimiento de Android en el mercado como el motor que impulsa el aumento de los ataques a estos sistemas por medio de *malware*. Mencionan que uno de los retos para crear formas de protección ante este tipo de ataques radica en las técnicas de ofuscación de código, las cuales permiten camuflar las verdaderas intenciones de un código malicioso sin que se vea afectado el rendimiento o la funcionalidad de la app. Los investigadores proponen un análisis con enfoque *black-box* a la aplicación de ofuscación de código tanto a determinadas y conocidas características de *malware* Android como a su efecto al enfrentarse con diferentes detectores de *malware* Android, para luego aplicar ingeniería inversa en estos algoritmos y determinar en qué característica se basa un detector según sus capacidades.

8.3. La investigación

Este proyecto se llevó a cabo bajo la metodología CRISP-DM [13] y fue seleccionada porque se ajusta al manejo de los datos de este proyecto y al desarrollo y entrenamiento de modelos de *machine learning*. Posteriormente, se incluyó un conjunto de actividades de *secure learning* para mejorar la robustez del modelo construido en la primera aproximación.

En cuanto al entendimiento de los datos, con base en el *dataset* escalado y los datos adversarios resultantes del proyecto de Delgado [5], fue posible analizar los valores y comprender cómo estaban estructurados a través de una descripción de los datos. En el primer *dataset* original se incluyen 7.832 muestras del comportamiento en red de aplicaciones Android: 3.141 etiquetadas "*malicious*" y 4.691 "*benign*".

El segundo *dataset* contiene los datos adversarios generados y es de gran importancia para este proyecto porque se deben incluir para generar la mayor robustez posible en el modelo final. Ambos *datasets* cuentan con once variables, diez de ellas corresponden al comportamiento de una aplicación Android en red y la restante a la etiqueta dada, la cual informa si la muestra es *malware* o no.

Las variables que componen el *dataset* son: *bytes* enviados por la aplicación; *bytes* recibidos por la aplicación; paquetes TCP; paquetes recibidos por la

aplicación; paquetes enviados por la aplicación; volumen de *bytes*; número de consultas DNS; paquetes distintos TCP; paquetes UDP; IP externas y tipo (legítimo o malicioso).

Cabe mencionar que hubo dos *datasets* de tipo csv relativamente recientes (de 2020) que se encontraron en la web pero no se consideraron. Uno de ellos solo era una recopilación de datos tomados en 2018, que ya se tenían, y el otro, aunque sí contenía datos actualizados, no contaba con las variables que el clasificador en cuestión analiza para determinar si una muestra es un *malware* o no, y es importante que las variables sean las mismas porque para robustecer y mejorar el clasificador Android, se necesita coherencia entre los datos ya estudiados y los nuevos.

Durante la fase de preparación de los datos se confirmó su integridad verificando la no inclusión de valores no permitidos según su columna. Una vez completada la evaluación de calidad, se realizó la mezcla de los dos *datasets* anteriores y se verificó nuevamente su integridad.

En la fase de modelado se entrenaron nuevamente seis modelos de *machine learning* con el objetivo de actualizarlos a versiones más recientes y confirmar al modelo *random forest* como el mejor clasificador (ver resultados en la Tabla 8.2). *Random forest* fue después usado para desarrollar el modelo de *Deep Reinforcement Learning* basado en *Deep Q Learning* para generar los datos adversarios.

Tabla 8.2. Desempeño de los modelos candidatos a clasificador actualizados

Algoritmo	Precision		Recall		F1-Score		Accuracy
	benign	malicious	benign	malicious	benign	malicious	
Naive Bayes	0.81	0.41	0.12	0.96	0.20	0.58	0.4468
Random forest	0.93	0.90	0.94	0.88	0.93	0.89	0.9172
k-NN: K=4	0.89	0.89	0.93	0.83	0.91	0.86	0.8922
SVM	0.62	0.90	1	0.06	0.76	0.11	0.6271
Regresión logística	0.72	0.68	0.86	0.47	0.78	0.56	0.7063
Árboles de decisión	0.90	0.85	0.90	0.84	0.90	0.84	0.8773

En la fase de evaluación, con el objetivo de verificar la disminución de los falsos negativos, se seleccionaron las métricas de *precision*, *F1-score*, *recall* y pérdida, por eso mismo se privilegió el *recall*. Estos resultados, una vez graficados (en

una matriz de confusión, por ejemplo, para el caso de la precisión), permiten verificar el correcto comportamiento del modelo a evaluar.

La fase de despliegue no se realizó por no estar incluida dentro del alcance de este proyecto, pues su propósito es generar un modelo más seguro y resistente a ciberataques, justo antes de su despliegue.

Secure learning

El ataque al modelo de defensa se realizó con enfoque *gray-box* con el supuesto de un atacante que solo tiene acceso a consultar el modelo clasificatorio, es decir, a entregarle una muestra de cierta cantidad de variables que desconoce y recibir una respuesta de si esta muestra entra en la categoría de *malware* o no. El atacante conoce la información que recibe el modelo clasificatorio, que en este caso corresponde a datos del comportamiento dinámico de una aplicación, específicamente del comportamiento de la comunicación en red.

Con base en la llamada *cyber kill chain* [14] el ataque se divide en varios pasos, el primero fue realizar una exploración del modelo de defensa; partiendo del desconocimiento del número de variables que el modelo clasificatorio recibe y el tipo de dato de cada una de estas, se consultó de manera aleatoria e interactiva cada una de las variables; el segundo fue generar muestras de *malware* que luego serían el insumo que se convertiría en datos adversarios, para lo que se inició con un recorrido aleatorio en busca de muestras que el modelo de defensa considerara *malware*; el tercero fue la generación de datos adversarios, para ello se implementó el algoritmo de *deep Q-learning* y luego se modificó convirtiéndolo en una red generativa adversaria, en donde el discriminador es el modelo de defensa.

Luego de realizar numerosas instancias, el resultado fue un *dataset* con el mismo número de muestras que el *dataset* de *malware* generado, en el cual todas las entradas han sufrido perturbaciones mínimas, de tal forma que ahora el modelo clasificatorio, a pesar de que contiene muestras muy similares a las de tipo *malware*, se detectan como si fueran de tipo legítimo.

Para el entrenamiento adversario, se preparó el *dataset* final incluyendo los datos generados por Delgado [5], los datos originales y los datos adversarios generados en este proyecto. De cada uno de los *dataset* se tomó el 75 % para entrenamiento y 25 % para pruebas. La actividad siguiente fue el entrenamiento del modelo clasificatorio, para lo cual se generó un modelo de *random forest* y se

entrenó con el *dataset* final. Luego, se evaluó utilizando la porción de pruebas de los *datasets*, para así comparar las métricas y determinar la robustez alcanzada.

8.4. Resultados

En su proyecto, Delgado [5] entrenó seis modelos de aprendizaje supervisado con el objetivo de determinar el clasificador que obtuviera mejores resultados. El modelo seleccionado fue *random forest* con una *accuracy* de 0.9172, valor que se mantuvo durante la actualización hecha durante el proyecto actual. Pero esto no sucedió en todos los modelos, en algunos se obtuvieron resultados diferentes al pasar a la versión de scikit-learn 0.20.1 a 1.0.1: el árbol de decisión mejoró su *accuracy* de 0.8845 a 0.8799; SVM lo disminuyó de 0.7359 a 0.6271; y la regresión logística lo aumentó de 0.7053 a 0.7063.

Los resultados de la primera fase del ataque al modelo de defensa tuvieron como base la exploración, de ella se obtuvo que eran diez las variables que recibía el modelo clasificatorio. Este dato se consiguió mediante la captura de errores, es decir que después de correr un algoritmo de fuerza bruta y verificar las excepciones, se obtuvo la combinación de diez variables de carácter numérico. Como recomendación, se propone minimizar la información que se entrega al momento de capturar los errores, ya que en este caso, aunque se ignoró, al entregarle al modelo un número errado de variables, se lanza un error que informa el tipo y la cantidad de variables que requiere el modelo, lo que facilita significativamente la fase de reconocimiento del atacante.

El segundo paso fue la construcción del generador de *malware* con base en el conocimiento obtenido en el primer paso. Para esto también se utilizó un algoritmo de fuerza bruta que de manera aleatoria género muestras de *malware*; para cada una de las variables, al desconocerse los rangos que podrían tomar, se escogió como máximo 100 y como mínimo –100; la elección del rango se realizó teniendo en cuenta que en este intervalo se podían hallar muestras de *malware* más rápidamente que en otros rangos descartados como –10, 10 y –50, 50. A partir de esto, se generaron en total 1.500 datos de *malware*, los que se almacenaron en un archivo csv que se entrega en el siguiente paso.

A continuación, se implementó el algoritmo de DQN (Figura 8.2), el cual es una modificación del *framework* básico de RL utilizando *Deep Q-learning*, en donde la parte profunda se realiza a través de un agente de red neuronal. En este caso, la salida tiene un tamaño de 20, es decir que existen veinte acciones que

el agente puede realizar sobre el estado siguiente a partir del estado anterior. Las acciones consisten en aumentar o disminuir cada una de las diez variables independientemente en diez unidades.

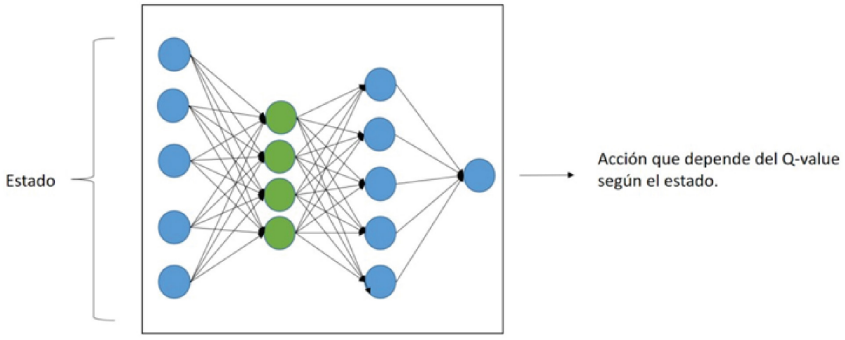

Figura 8.2. Agente DQN

En cuanto a los hiperparámetros, se utilizó un *learning rate* de 0.001, un gamma de 0.9 y un épsilon de 1, con el fin de iniciar con una exploración totalmente aleatoria e ir realizando explotación a medida que se actualizan los pesos de la red neuronal. El entrenamiento se ajustó a mil épocas, en cada una de las cuales se realizaban *N* modificaciones con el fin de lograr finalmente el resultado esperado.

Gráficamente se puede analizar qué tanto alteró finalmente el modelo DRL al *malware* generado, al compararlo con los datos adversarios, es decir los datos que a pesar de ser inicialmente *malware* y tener mínimas modificaciones, ahora el modelo clasificatorio detecta como muestras legítimas. La media de los cambios realizados en cada variable fue de 5,96 unidades. En la Figura 8.3 se puede apreciar en azul la media de los datos que el modelo consideraba *malware* y en rojo los datos modificados que ahora el modelo detecta como legítimos.

Con esta información es posible para un atacante determinar rangos para cada una de las variables, con el supuesto de que el atacante conoce que el discriminador realiza una evaluación a partir del comportamiento en red de la aplicación.

Para el entrenamiento adversario, el primer paso fue verificar la robustez del modelo clasificatorio anterior sobre los datos adversarios generados a partir

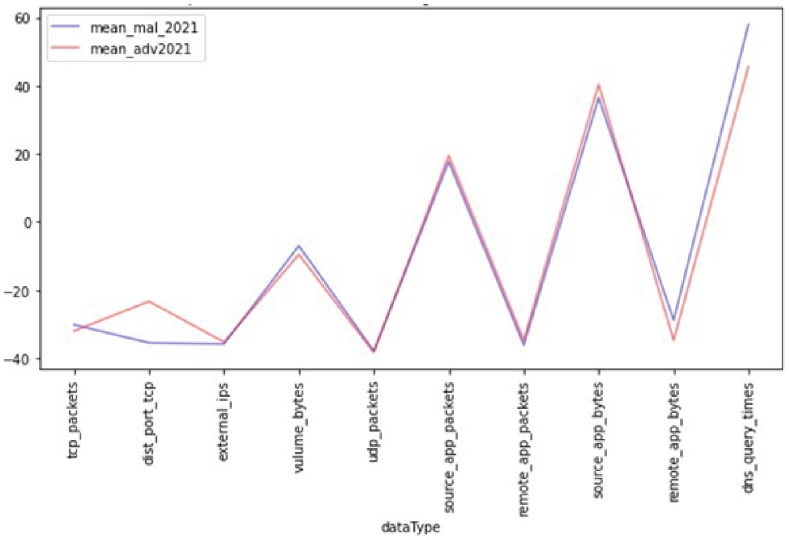

Figura 8.3. Malware generado vs datos adversarios

de caja gris. Para esto se utilizó el total de los datos, es decir, la mezcla de los valores originales, los datos adversarios del proyecto anterior [5] y los datos generados en este proyecto.

En la Figura 8.4 se puede apreciar, a través de la *accuracy* del 0.8614, el nivel de vulnerabilidad de este modelo clasificatorio que falla en detectar 1.095 muestras maliciosas; el contraste con la Figura 8.5 ofrece la evidencia de una mejora significativa, de un modelo clasificatorio notablemente más robusto que alcanza una *accuracy* de 0.9743.

		Predicción	
		Legítimo	Malicioso
Realidad	Legítimo	4.562 VN	129 FP
	Malicioso	1.095 FN	3.044 VP
	Accuracy=0.8614; misclass=0.1386		

Figura 8.4. Matriz de confusión del modelo vulnerable

		Predicción	
		Legítimo	Malicioso
Realidad	Legítimo	4.564 VN	127 FP
	Malicioso	100 FN	4.039 VP
	Accuracy=0.9743; misclass=0.0257		

Figura 8.5. Matriz de confusión del modelo robusto

8.5. Conclusiones y recomendaciones

En este proyecto se presentó un modelo que a través de DRL logra mejorar significativamente la robustez del clasificador de *malware*. Inicialmente no se reconocieron 1.095 muestras de datos adversarios, pero después del proceso de *adversarial training* tan solo 100 datos se mantuvieron mal clasificados. Por otro lado, respecto a la *accuracy* de los datos adversarios, se encontró que el algoritmo de DRL alcanzó una modificación mínima de 5,96628 respecto de los valores de entrada, lo que permite observar que los datos adversarios presentan una gran similitud con los datos de *malware* generados.

El uso de DRL en combinación con una estructura de red generativa adversaria es de gran utilidad al momento de realizar un proceso de entrenamiento adversario, porque permite reducir los tiempos de generación de estos datos. Los tiempos con el método sugerido oscilan entre 4:50 minutos y 5:10 minutos, mientras que, tomando como ejemplo la exploración aleatoria del modelo clasificatorio, obtener 500 muestras de *malware* toma alrededor de 30 minutos.

Para el desarrollo seguro de modelos de ML, en primer lugar se recomienda realizar un modelo de base teniendo en cuenta los datos originales; posteriormente, una buena medida es mejorar la robustez mediante la inclusión de datos adversarios generados a través de RL, utilizando o no los datos originales. Una vez generados esos datos adversarios, se mezclan, se entrena el modelo y se verifica la mejoría respecto del modelo inicial. Finalmente, al momento del despliegue, es de gran importancia dificultar los intentos de reconocimiento que los atacantes pueden intentar realizar, para esto, se recomiendan acciones tales como: limitar el número de consultas, adicionar

tiempos de espera y evitar dar información muy específica al momento de capturar errores. Aplicar estas recomendaciones puede significar una gran diferencia frente a ataques a modelos de ML.

Teniendo en cuenta el alcance de este proyecto y los objetivos conseguidos, como trabajo futuro se propone:

- llevar a cabo el proceso de modificación de un *malware* real con el fin de demostrar los riesgos para un modelo de clasificación poco seguro;
- proponer un método de ataque adversario y posterior entrenamiento desde un enfoque *black-box*; y
- continuar con el análisis de los rangos para determinar el intervalo en el que un atacante podría modificar un *malware*.

Los recursos (documentos, gráficos, códigos, etc.) obtenidos durante la ejecución del proyecto están disponibles en [15]

Referencias

[1] Operating system market share worldwide. (2022, mayo). *Statcounter*. Disponible: https://gs.statcounter.com/os-market-share.

[2] *Google Play Store: number of apps 2020*. Disponible: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/

[3] C. Urcuquí, M. García, J. Osorio, y A. Navarro, *Ciberseguridad: un enfoque desde la ciencia de datos*, Cali, Colombia: Universidad Icesi, 2018.

[4] S. Chaieb. (2021, feb. 3). *Machine learning systems: security*. Disponible: https://sahbichaieb.com/mlsystems-security/

[5] J. Delgado, "Secure learning para detección de Android malware," tesis, Universidad Icesi, Cali: Colombia, 2019.

[6] B. Dickson. (2020, nov. 18). *Robust AI: protecting neural networks against adversarial attacks*. Disponible: https://www.experfy.com/blog/ai-ml/robust-ai-protecting-neural-networks-against-adversarial-attacks/

[7] M. Barreno, B. Nelson, A. D. Joseph, y J. D. Tygar, "The security of machine learning," *Machine Learning*, vol. 81, no. 2, pp. 121–148, 2010.

[8] H. Rathore, S. K. Sahay, P. Nikam, y M. Sewak, "Robust Android

malware detection system against adversarial attacks using Q-Learning," *Information Systems Frontiers*, 2020 Nov 15.

[9] A. Romdhana, A. Merlo, M. Ceccato, y P. Tonella, "Deep reinforcement learning for black-box testing of android apps," *arXiv preprint arXiv*:2101.02636, 2021.

[10] Z. Fang, J. Wang, B. Li, S. Wu, Y. Zhou, y H. Huang, « Evading anti-malware engines with deep reinforcement learning," *IEEE Access*, vol. 28, No. 7, pp. 48867-48879, 2019 Mar 28.

[11] H. Li, S. Zhou, W. Yuan, J. Li, y H. Leung, "Adversarial-example attacks toward android malware detection system," *IEEE Systems Journal*. Vol. 14, No. 1, pp. 653-656, 2019 Abr. 11.

[12] G. Nellaivadivelu, F. Di-Troia, y M. Stamp, "Black box analysis of android malware detectors," *Array*, vol. 6, 2020, https://doi.org/10.1016/j.array.2020.100022

[13] *CRISP-DM*. Disponible: http:/crisp-dm.eu/

[14] *Cyber kill chain*. Disponible: https://www.sans.org/blog/applying-security-awareness-to-the-cyber-kill-chain/

[15] C.C. Urcuqui. (2021). SL and DRL for android malware detection (PDG). *i2tResearch* [repositorio GitHub]. Disponible: https://github.com/i2tResearch/Ciberseguridad_web/tree/master/Android/SL%20and%20DRL%20for%20android%20malware%20detection(PDG)

Capítulo 9
Método de detección de deepfake mediante técnicas de machine learning

Bayron Campaz, Juan David Díaz, Santiago Gutiérrez, Christian Camilo Urcuqui

9.1. Introducción

En las últimas décadas la Inteligencia Artificial (IA) ha avanzado a un ritmo acelerado debido principalmente al mejoramiento de la capacidad de procesamiento y a las aplicaciones de esta tecnología en diversas áreas del conocimiento. *Machine learning* es una rama de la IA que a su vez posee un área denominada *Deep Learning* (DL), la cual ha permitido grandes avances en el procesamiento de contenido visual. Uno de los más recientes usos de los algoritmos de DL es la manipulación de imágenes y vídeos, por ejemplo, insertando la imagen del rostro de una persona en la imagen del rostro de otra. La manipulación en este tipo de contenido que usa estos algoritmos se denomina *deepfake*.

Estas tecnologías son cada vez más accesibles, y por ello "la creciente facilidad de crear contenido de audio y vídeo falso crea amplias oportunidades para el chantaje, la intimidación y el sabotaje" [1] y puede tener repercusiones a nivel político y social.

Las redes neuronales, que se pueden usar para procesar imágenes, son la herramienta que le han permitido al *deepfake* producir resultados realistas con la capacidad de engañar a las personas, lo que incluye el intercambio de rostros y la generación de nuevos rostros instantáneamente (luego de un entrenamiento previo). Las redes neuronales son uno de los avances más prolíficos en IA, dado que permite simular el cerebro humano de manera que una máquina pueda lograr un nivel de aprendizaje con un alto grado de detalle.

La idea del *deepfake* comienza en 2012 durante una competencia de ImageNet, en donde el ganador usa por primera vez un algoritmo de red neuronal profunda; en 2014 Ian Goodfellow crea las redes generativas antagónicas

(*Generative Adversarial Networks*, GAN), que consisten en sistemas de inteligencia artificial capaces de crear imágenes completamente nuevas; en noviembre de 2017 un usuario de Reddit llamado "Deepfake" pública vídeos pornográficos con rostros de celebridades y a lo largo de 2018, el contenido aparece en cientos de artículos de prensa como: The New York Times, Washington Post, The Guardian, The Economist, The Times y The BBC. Asimismo, en 2018 BuzzFeed publica un vídeo falso de Barack Obama insultando a Donald Trump, con el fin de crear conciencia sobre cómo los contenidos sintéticos generados por la IA podrían usarse para distorsionar y manipular la realidad [2].

Estas situaciones, en las cuales se han involucrado elementos como vídeos de personas influyentes, no deben pasarse por alto, más aún cuando esta actividad no solo continúa sino que se amplifica y se hace más cercana y de mayor acceso. El *deepfake* puede representar un gran problema a nivel social dado que tiene un potencial de provocación elevado, basta con imaginar el efecto de un vídeo del primer mandatario de un país expresando un discurso con contenido bélico o de odio hacia un grupo social, es claro que "en un mundo ya preparado para la violencia, tales grabaciones tendrán un gran potencial de incitación" [1].

Deepfake podría ser usado también en la contienda política para modificar las intenciones de voto o para afectar la imagen de una figura pública, "estos avances amenazan con desdibujar aún más la línea entre la verdad y la ficción en política" [3]. El uso de la tecnología para alterar contenido hace cada vez más difícil diferenciar entre lo verdadero y lo falso.

Deepfake se distingue de otras técnicas de manipulación de vídeo por su potencial para obtener resultados fotorrealistas impactantes [4]. Las facilidades que existen para acceder a esta técnica y usarla permiten que casi cualquier persona pueda emplearla, por lo que es necesario desarrollar métodos que permitan distinguir entre el contenido auténtico y el contenido alterado con *deepfake*.

No existe un modelo o una tecnología única para hacer *deepfake*, por esto es importante desarrollar métodos que permitan detectar el contenido alterado para los distintos modelos y tecnología, pues al hacerlo se gana la posibilidad de disminuir la posibilidad de engaño y manipulación de la opinión pública.

El mundo actual esta interconectado y cuenta con fácil acceso a una gran cantidad de información. Esta información es propensa a ser alterada o falsificada, más aún con el desarrollo de técnicas como el *deepfake*, que permite falsificar contenido de audio, imágenes y vídeo.

La gran dificultad para discernir la suplantación mediante el uso de *deepfake* en conjuntos de datos de vídeos e imágenes de última generación se debe principalmente a: los grandes avances de la inteligencia artificial, la facilidad de acceso a la tecnología, la facilidad de uso de herramientas que implementan esta técnica y al sesgo que tienen los métodos de detección actuales hacia imágenes de baja calidad. La manipulación de estos archivos puede generar la afectación negativa en la imagen de las víctimas, la manipulación de la información y un cambio en las percepciones y decisiones de las personas, incluso la incitación a la violencia.

El objetivo general del proyecto es evaluar un método de detección de *deepfake* que emplee técnicas de *machine learning*, que permita distinguir entre contenido visual alterado y contenido visual auténtico. Sus objetivos específicos son: analizar el funcionamiento y las características de los métodos generadores y detectores de *deepfake*; recopilar un conjunto de datos reales y alterados; implementar un método de detección de *deepfake*; y evaluar el método propuesto en comparación con otros.

Actualmente, existen métodos y herramientas para la generación de imágenes y vídeos no reales basados en inteligencia artificial, entre estos podemos encontrar desde herramientas de código abierto, tales como Face2Face, que permiten hacer *deepfake* "en vivo", hasta aplicaciones de escritorio, como Faceswap, que permiten hacer un intercambio de caras entre dos vídeos. Los métodos y herramientas más conocidas se describen a continuación.

Faceswap es un programa para escritorio que usa una red neuronal para intercambiar un rostro original por un rostro de intercambio. Para hacer esto cuenta con un codificador compartido, que se entrena con dos conjuntos de rostros, A y B, donde A es el conjunto A de las caras originales y B el de las caras para intercambio. Compartiendo el codificador se logra que este genere un algoritmo único para ambos conjuntos de rostros.

Asimismo, cuenta con decodificadores diferentes, el decodificador A se encarga de reconstruir los rostros de conjunto A y el decodificador B, los rostros del conjunto B. Cuando se quiere hacer el intercambio de rostro (luego de entrenar al modelo con suficientes imágenes de cada conjunto), el decodificador A reconstruirá los rostros del conjunto B y el decodificador B reconstruirá los rostros del conjunto A, lo que dará como resultado una cara intercambiada (ver Figura 9.1).

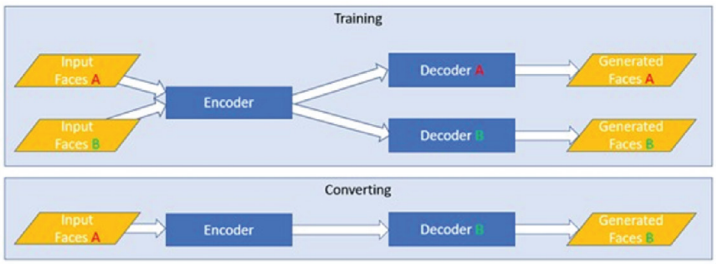

Figura 9.1. Esquema de funcionamiento de Faceswap

Faceswap permite usar diferentes configuraciones de parámetros y modelos. El que fue descrito es el modelo original, cuya entrada y salida son imágenes de rostros de 64 x 64 píxeles, sin embargo, existen modelos más sofisticados, que permiten entradas y producen salidas en diversas resoluciones.

Face2Face es una herramienta de código abierto que opera a través de dos entradas, en primer lugar, el sistema procesa una transmisión de vídeo en vivo, a través de una cámara web del sujeto a reemplazar, en segundo lugar, un vídeo en el que esté presente el rostro del objetivo (vídeo del sujeto que se quiere suplantar) utilizado. Las imágenes de los rostros se sintetizan usando un modelo multilinear de rostros y un modelo de transformación rígida.

Este método permite encontrar parámetros de los rostros que posteriormente se someterán a la aplicación de un método de minimización de energía variacional, lo que permitirá optimizar los parámetros; para minimizar esta energía, se usa un solucionador de mínimos cuadrados iterativos reponderados iterativamente (*Iteratively Reweighted Least Squares*, IRLS), lo que permite obtener la identidad facial, entre otros datos relevantes de los actores fuente y objetivo. En ejecución las animaciones se reconstruyen usando un seguimiento a cuadro con una formulación energética similar a la anterior. Para la recreación se usa una deformación rápida que opera en el estadístico usado anteriormente.

Fakeapp es una aplicación para dispositivos móviles, usa una foto del rostro de una persona, a la que posteriormente se le realiza la edición que se seleccione. La versión gratuita ofrece diferentes ediciones, como aplicar una sonrisa básica, añadir una barba básica, realizar una mezcla de fotos de rostro, aplicar un aumento de edad, cambiar el color del cabello y agregar accesorios; la versión Pro cuenta con herramientas de edición más potentes. Al ser una aplicación comercial, no es de código abierto.

Faceswap-GAN es un método basado en el uso de redes generativas antagónicas (GAN), específicamente de SAGAN (*Self-Attention Generative Adversarial Networks*), un tipo de red que introduce un mecanismo de auto atención en las GAN convolucionales.

El módulo de autoatención es complementario a las convoluciones y ayuda a modelar dependencias de múltiples niveles a largo plazo en las regiones de la imagen. Armado con atención propia, el generador puede dibujar imágenes en las que los detalles finos en cada ubicación se coordinan cuidadosamente con los detalles finos en las partes distantes de la imagen. Su arquitectura está compuesta por un codificador, un decodificador, una red generativa y una red discriminadora.

Las características más importantes de este método son: la pérdida perceptual de cara VGG que mejora la dirección de los globos oculares y permite así al *deepfake* ser más realista y consistente con la cara de entrada; una resolución de salida configurable (64x64, 128x128 y 256x256); la alineación de rostros usando MTCNN, para detecciones más estables y una alineación facial confiable, y un filtro de Kalman en conversión de vídeo.

Style-Based Generator Architecture GAN es un método basado en una red generativa antagónica con una arquitectura cimentada en una transferencia de estilos que se basa en representar el contenido de una imagen en el estilo de otra. Esta arquitectura conduce a: la separación automática, sin supervisión, de los atributos de alto nivel (por ejemplo, posee la identidad cuando se entrena en rostros humanos); y la variación estocástica en las imágenes generadas (por ejemplo, pecas, cabello).

El generador parte de una entrada constante aprendida y ajusta el estilo de la imagen en cada capa de convolución en función del código latente, por lo tanto, controla directamente la fuerza de las características de la imagen a diferentes escalas. Combinado con el ruido inyectado directamente en la red, este cambio arquitectónico conduce a la separación automática y sin supervisión de los atributos de alto nivel de la variación estocástica en las imágenes generadas, lo que permite una mezcla específica a escala intuitiva, así como operaciones de interpolación.

Esta herramienta no solo sirve para generar *deepfakes*, sino que también permite la transferencia de estilo entre diversos tipos de imágenes, sin importar que estas contengan objetos, animales o cualquier tipo de representación

visual. El modelo bajo el cual se basa este método permite el entrenamiento con cualquier tipo de resolución, sin embargo, usa como ejemplo un modelo preentrenado para imágenes de 1.024 x 1.024 píxeles.

De la revisión anterior, se puede vislumbrar que existen diferentes métodos de generación de *deepfake* que emplean diversas técnicas y tecnologías, y reciben y producen diferentes tamaños y tipos de contenido visual (vídeos o imágenes).

Cabe destacar que las redes neuronales, sean clásicas, profundas o generativas adversarias, son ampliamente usadas en los generadores de *deepfake*. A continuación se presenta una comparación entre los métodos teniendo en cuenta: el tipo de entrada (Tabla 9.1), la resolución del dato de entrada (Tabla 9.2), la resolución del dato de salida (Tabla 9.3), el tipo de modelo (Tabla 9.4), el uso (Tabla 9.5) y el acceso al código fuente (Tabla 9.6).

Tabla 9.1. Comparación entre métodos generadores: tipo de entrada

Método	Detalle
Faceswap	Conjunto de imágenes en formato PNG y/o JPG
Face2Face	Transmisión de vídeo en vivo a través de una cámara web y vídeo monocular
FakeAPP	Vídeo fuente y objetivo en cualquier formato de vídeo
Faceswap-GAN	Vídeo fuente y vídeo objetivo en formato MP4
Style-based generator architecture GAN	Conjunto de imágenes en formato PNG
FaceAPP	Imagen en formato JPG o PNG

Tabla 9.2. Comparación entre métodos generadores: resolución del dato de entrada

Método	Detalle
Faceswap	Cualquiera
Face2Face	1280 x 720 (vídeo monocular); 640 x 480 (vídeo en vivo)
FakeAPP	Cualquiera
Faceswap-GAN	Cualquiera
Style-based generator architecture GAN	1024 x 1024 píxeles
FaceAPP	Cualquiera

Tabla 9.3. Comparación entre métodos generadores: resolución del dato de salida

Método	Detalle
Faceswap	La misma del video o imagen que se ingresó
Face2Face	1280 x 720 pixeles
FakeAPP	La misma del video objetivo
Faceswap-GAN	64 x 64; 128 x 128; 256 x 256 pixeles
Style-based generator architecture GAN	1024 x 1024 píxeles
FaceAPP	La misma del video o imagen que se ingresó

Tabla 9.4. Comparación entre métodos generadores: tipo de modelo

Método	Detalle
Faceswap	Red neuronal
Face2Face	Modelo multilinear de rostros y modelo de transformación rígida.
FakeAPP	No disponible
Faceswap-GAN	GAN + auto encoder
Style-based generator architecture GAN	GAN + normalización de instancia adaptativa (AdaIN)
FaceAPP	No disponible

Tabla 9.5. Comparación entre métodos generadores: uso

Método	Detalle
Faceswap	Está integrado en una aplicación de escritorio
Face2Face	No aplica
FakeAPP	Es una aplicación de escritorio.
Faceswap-GAN	El código fuente está organizado en notebooks, lo que permite realizar deepfakes a través del navegador mediante Google Colab.
Style-based generator architecture GAN	El código fuente del modelo cuenta con instancias previamente entrenadas; se puede usar para transferencia de estilos entre dos imágenes.
FaceAPP	No aplica

Tabla 9.6. Comparación entre métodos generadores: acceso al código fuente

Método	Detalle
Faceswap	Open source
Face2Face	Privado
FakeAPP	Privado
Faceswap-GAN	Open source
Style-based generator architecture GAN	Open source
FaceAPP	Privado

Error level analysis

El análisis de nivel de error (ELA) es un método forense de análisis de artefactos comprimidos que aprovecha los esquemas de compresión con pérdida de imágenes manipuladas para identificar su falsificación. El nivel de calidad original de una imagen es una característica única en sí misma, por lo tanto, cualquier proceso de alteración deja sus huellas.

Error level analysis usa una imagen comprimida por un esquema con pérdidas, la recomprime con una tasa de error conocida y luego calcula la diferencia absoluta entre la imagen analizada y la recomprimida. Esta diferencia entre las imágenes es el nivel de error asociado con los píxeles originales, el cual, visto como una cantidad de cambio, está directamente asociado con la pérdida de compresión.

- si la cantidad de cambio es pequeña (Figura 9.2a), el píxel ha alcanzado sus mínimos locales de error a la tasa de error determinada; y
- si hay una gran cantidad de cambio (Figura 9.2b), entonces los píxeles no están en sus mínimos locales y es probable que se hayan insertado artificialmente a la imagen [5].

9.2 Estado del arte

A continuación se describen cinco proyectos con un enfoque similar al planteado en este proyecto. Una comparación entre ellos y el proyecto actual se presenta en la Tabla 9.7.

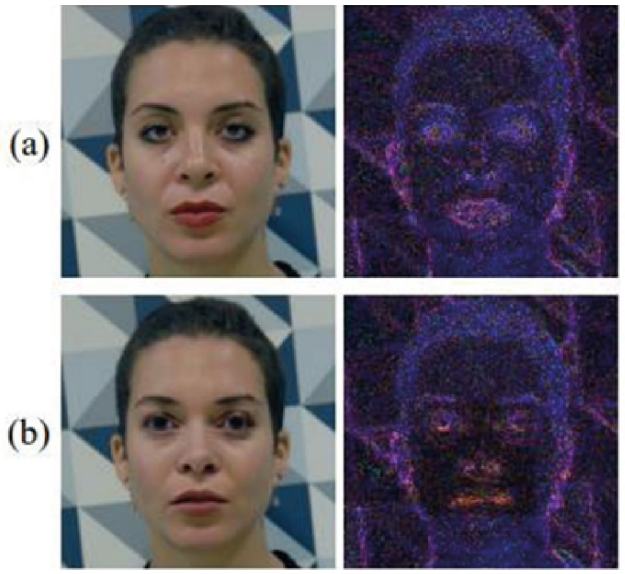

Figura 9.2. Método forense ELA: (a) imagen real; (b) imagen alterada

Two-stream neural networks for tampered face detection

Zhou, Han, Morariu y Davis [6] proponen una red de dos flujos para la detección de alteraciones faciales. En su investigación, entrenan una red mediante GoogLeNet para detectar artefactos de manipulación en una secuencia de clasificación de caras, con el fin de aprovechar las características que capturan los residuos de ruido local y las características de la cámara como una segunda secuencia; además, utilizan dos aplicaciones de intercambio de caras para crear un nuevo conjunto de datos que consta de imágenes manipuladas. Este método no puede detectar caras manipuladas muy pequeñas porque la secuencia de clasificación de caras necesita cambiar el tamaño de la cara de entrada a 299 × 299, y en el muestreo de caras pequeñas se pierde información visual crucial para la detección de alteraciones.

Exposing deepfake videos by detecting face warping artifacts

Li y Lyu [7] describen un método basado en *deep learning* capaz de distinguir los vídeos falsos generados por IA, de los vídeos reales. Este método se basa en que los algoritmos de *deepfake* deben realizar unas transformaciones que dejan elementos distintivos en los vídeos resultantes y muestran que estos pueden ser

Tabla 9.7. Comparativo de proyectos

Criterio	Two-Stream…	Exposing… FWA	Mesonet	Exposing… HeadPose	Multitask…	Actual
Tecnología usada	Red neuronal profunda GoogLeNet	CNN (ResNet50)	CNN	Modelos matemáticos para estimar la postura de la cabeza en 3D a partir de imágenes de la cara	CNN	CNN (Xception) + Imagenet
Imágenes usadas en el entrenamiento	705 alteradas y 1.400 reales.	15.185 alteradas y 15.185 reales.	5.111 alteradas y 7.250 reales.	10.847 alteradas y 10.847 reales.	218.179 alteradas y 218.179 reales.	4.073 alteradas y 2.597 reales.
Datos de entrenamiento	Generados con SwapMe.	Recopilados en varias plataformas de vídeo.	Recopilados en varias plataformas de vídeo.	Conjunto de datos UADFV	Conjunto de datos FF-DF	Conjunto de datos FF-DF, DFD, DFDC, Celeb-DF
¿Conjunto de datos público?	No	Si	No	Si	Si	Si
¿AUC mayor a 80 % sobre el conjunto de datos de primera generación?	No	Si	No	No	No	Si
¿AUC mayor a 80% sobre el conjunto de datos de segunda generación?	No	No	No	No	No	Si
¿Acceso a código fuente? (open source)	No	Si	Si	Si	Si	Si

capturados por redes neuronales convolucionales. En comparación con otros métodos, tiene la ventaja de no necesitar imágenes generadas por *deepfake* para el entrenamiento.

Mesonet: A compact facial video forgery detection network

El método propuesto por Afchar et al. [8] busca detectar de manera automática y eficiente la manipulación de la cara en vídeos. Sigue un enfoque de aprendizaje profundo y presenta dos redes, ambas con un bajo número de capas, para enfocarse en las propiedades mesoscópicas de las imágenes. Se evalúan esas redes tanto en un conjunto de datos existente como en un conjunto de datos construido por los investigadores a partir de vídeos en línea. Las pruebas demuestran una tasa de detección muy exitosa con más del 98 % para *deepfake* y 95 % para Face2Face en imágenes de 256 x 256 píxeles.

Exposing deep fakes using inconsistent head poses

Este método, propuesto por Yang, Li y Lyu [9], busca detectar imágenes o vídeos de caras falsas generadas por *deepfake*. Se basa en la observación de que el contenido alterado se genera al empalmar la región de la cara sintetizada en la imagen original y que al hacerlo se introducen errores que pueden revelarse cuando se estiman las posturas de la cabeza en 3D, a partir de las imágenes de la cara. Los investigadores realizan experimentos para demostrar este fenómeno y usando características basadas en este indicio, evalúan un clasificador SVM usando un conjunto de imágenes de caras reales y falsificadas.

Multi-task learning for detecting and segmenting manipulated facial images and videos

Nguyen, Fang, Yamagishi y Echizen [10] diseñan una CNN que utiliza el enfoque de aprendizaje de tareas múltiples para detectar simultáneamente imágenes y vídeos manipulados y ubicar las regiones manipuladas. La información obtenida al realizar una tarea se comparte con la otra tarea y, por lo tanto, mejora el rendimiento de ambas. Se utiliza un enfoque de aprendizaje semisupervisado; la red incluye un codificador y un decodificador en forma de Y; la activación de las características codificadas se utiliza para la clasificación binaria; la salida de una rama del decodificador se usa para segmentar las regiones manipuladas, y la otra para reconstruir la entrada, lo que ayuda a mejorar el rendimiento general.

9.3. La investigación

Este proyecto se llevó a cabo bajo *Cross-Industry Standard Process for Data Mining* (CRISP-DM), una metodología ampliamente probada y utilizada en proyectos que requieren procesamiento y análisis de datos, la misma que fue descrita en el capítulo correspondiente al marco conceptual.

En la primera fase de la metodología se hizo una comprensión del problema mediante el estudio de los métodos generadores y detectores de *deepfake* actuales (ver Métodos…). Para entender cómo se puede generar *deepfake*, usando Faceswap se generaron tres pares de vídeos con rostros intercambiados.

Para generar el primer par, se grabaron dos vídeos que contenían los rostros de los dos sujetos (sujeto A y sujeto B) a los que se querían intercambiar los rostros, el primero con una duración de 3:31 minutos, el segundo, 5:41 minutos. Usando la funcionalidad "*Extract*" de Faceswap se obtuvieron todas las imágenes en las que el algoritmo detectaba un rostro, y usando la funcionalidad "*Sort*" estas se ordenaron con base en su contenido, con el propósito de remover las imágenes que no tuvieran un rostro (por fallas en la detección del algoritmo) o que tuvieran rostros borrosos. Una vez obtenidos los rostros que se usarían en el intercambio, utilizando la funcionalidad "*Train*", que permite entrenar el modelo que aprenderá a intercambiar los rostros, se seleccionó el modelo Dfl-H128 y un Batch_size=64. El modelo realizó 232.162 iteraciones y tuvo una pérdida en el rostro A de 0.019 y en el rostro B de 0.022. Finalmente, usando la herramienta "*Convert*", se generaron las imágenes que componen el vídeo del sujeto B con el rostro del sujeto A; con la configuración predeterminada y la funcionalidad "Effmpg" se unieron las imágenes del paso anterior para producir el vídeo resultado. Lo mismo para el vídeo del sujeto A con el rostro del sujeto B, seleccionando la opción "*Swap Model*" en la funcionalidad "*Convert*".

Para los *deepfakes* del segundo par de vídeos se usaron grabaciones con duraciones de 6:35 y 4:07 minutos. El procedimiento fue el mismo. El modelo realizó 78.379 iteraciones y tuvo una pérdida en el rostro A de 0.025 y en el rostro B de 0.024. Para los *deepfakes* del tercer par de vídeos (Figura 9.3) se usaron grabaciones con una duración 56 segundos y 1:14 minutos. El procedimiento fue el mismo que para el primer par de vídeos, sin embargo, el modelo escogido fue el Dfaker. El modelo realizó 95.601 iteraciones y tuvo una pérdida en el rostro A de 0.025 y en el rostro B de 0.025.

Figura 9.3. Tercer par de vídeos usados para generación de deepfake (rostros originales a la izquierda, rostros intercambiados a la derecha)

En la fase de comprensión de los datos se realizó la caracterización de cinco conjuntos de datos más relevantes desde el punto de vista académico en el estudio de los *deepfakes*: se describió cada uno de estos conjuntos teniendo en cuenta elementos como: la cantidad, la resolución y la variabilidad de los vídeos que los componen, así como otros atributos. A continuación, se presenta cada uno de los conjuntos de datos con su respectiva descripción.

Deepfake-TIMIT incluye 640 vídeos *deepfake* generados con Faceswap-GAN, está basado en el conjunto de datos Vid-TIMIT. Los vídeos se dividen en dos subconjuntos de igual tamaño: DF-TIMIT-LQ y DF-TIMIT-HQ, con caras sintetizadas de 64 × 64 y 128 × 128 píxeles, respectivamente. Seleccionaron

manualmente 16 pares de personas de aspecto similar de la base de datos DF-TIMIT. Para cada 32 sujetos, se entrenaron dos modelos diferentes, conocidos como modelos de baja calidad (*Low Quality*, LQ), con 64 × 64 de entrada y salida, y de alta calidad (HQ), con tamaño de 128 × 128. Dado que hay diez vídeos por persona en la base de datos DF-TIMIT, generaron 320 vídeos correspondientes a cada versión, lo que resulta en un total de 640 vídeos con caras intercambiadas. No se realizó ninguna manipulación en el canal de audio. Este conjunto de datos fue lanzado en diciembre de 2018 [11].

FaceForensics++ consta de mil secuencias de vídeo originales que han sido manipuladas con cuatro métodos automatizados de manipulación facial: *Deepfakes*, Face2Face, Faceswap y NeuralTextures. Los datos provienen de 977 vídeos de YouTube, todos con una cara frontal rastreable, lo que permite que los métodos automáticos de manipulación generen falsificaciones realistas. La distribución de los datos respecto al género, la resolución y el cubrimiento de pixeles en las caras se presenta en la Figura 9.4, de izquierda a derecha, respectivamente. En la figura: VGA denota 480p, HD 720p y FHD 1080p; el eje y de la curva de la derecha corresponde al número de secuencias con una altura de píxel del cuadro delimitador dado (eje x). Este conjunto de datos se lanzó en enero de 2019 [12].

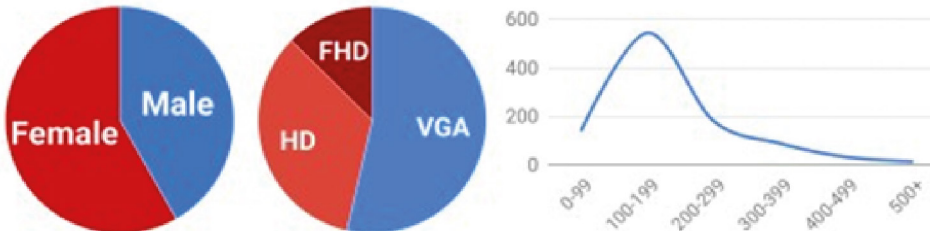

Figura 9.4. Composición del conjunto de datos FF-DF

Google/Jigsaw Deepfake Detection Dataset tiene 3.068 vídeos *deepfake* basados en 363 vídeos originales de 28 individuos de varias edades, géneros y grupos étnicos; no se revela mayor detalle del algoritmo de generación usado, pero si se establece que es una mejora a un algoritmo de generación de *deepfake* básico. Este conjunto de datos fue lanzado en septiembre de 2019 [13].

DeepFake Detection Challenge (DFDC) incluye vídeos con condiciones de iluminación y ángulos del rostro variados. Los creadores de este conjunto de datos manifiestan que los participantes pudieron grabar sus vídeos con el fondo que desearan, lo que produjo fondos visualmente diversos. La aproximación de la distribución general de género y raza en este conjunto de datos es, respectivamente, 74 % femenino; y 68 % caucásicos, 20 % afroamericanos, 9 % del este de Asia y 3 % del sur de Asia. Este conjunto de datos consta de 66 individuos y fue creado usando dos diferentes algoritmos de generación, aunque no se especifican cuáles. DFDC fue lanzado en octubre de 2019 y consta de 4.464 clips de entrenamiento y 780 de prueba [14].

Celeb-DF consta de 590 vídeos reales y 5,639 vídeos *deepfake* (correspondientes a más de dos millones de cuadros de vídeo). La duración promedio de cada vídeo es de aproximadamente 13 segundos, con una velocidad de cuadro estándar de 30 cuadros por segundo. Los vídeos reales fueron tomados de YouTube (videos públicos] y corresponden a entrevistas realizadas a 59 celebridades, con una distribución diversa entre: género (56.8 % hombres); edad (30,5 % entre 50 y 60 años, 26,6 % 40 años, 28,0 % 30 años y 6,4 % menos de 30 años); y grupo étnico (5,1 % asiáticos, 6,8 % afroamericanos y 88,1 % caucásicos. Además, los vídeos reales exhiben una gran variedad de cambios en aspectos tales como los tamaños de cara de los sujetos (en píxeles), orientaciones, condiciones de iluminación y fondos. Los vídeos de *deepfake* se generan intercambiando caras para cada par de los 59 sujetos. Los vídeos finales están en formato MPEG4.0 y no cuentan con sonido. Celeb-DF fue lanzado en noviembre de 2019 [15].

Al terminar la fase de comprensión de los datos, se decidió hacer énfasis en la detección de *deepfakes* en el conjunto de vídeos de Celeb-DF, dado que es el conjunto de datos que más dificulta la detección de contenido en los métodos detectores estudiados.

En la fase de preparación de los datos, para los experimentos que se llevaron a cabo en las fases de modelamiento y evaluación, se realizó una selección de los conjuntos de imágenes-vídeos y un preprocesamiento de los datos, siguiendo el procedimiento descrito a continuación.

1. Se tomaron 1.052 vídeos falsos, modificados con *deepfake*, y 890 vídeos reales de Celeb-DF.
2. Para los experimentos 1 y 4, se extrajo un total de 7.704 imágenes completas (*frames*) falsas y 7380 imágenes completas reales, del *dataset* citado en el primer punto.

3. Para los experimentos 2 y 5, se extrajo un total de 7.704 rostros falsos y 7.380 rostros reales usando la librería OpenCV;
4. Para los experimentos 3 y 6, se aplicó el método ELA con un ratio de error del 90 % a los rostros extraídos en el numeral 3.
5. Se tomaron 320 vídeos reales de los conjuntos de datos Celeb-DF, DFDC, FF-DF/*raw*, FF-DF/C23, FF-DF/C40, DFC/*raw*, DFC/C23 y DFC/C40, de donde C23 y C40 corresponden a los grados de compresión de los vídeos; también se tomaron 320 vídeos modificados con *deepfake* de los conjuntos de datos mencionados, junto con otros 320 vídeos de DF-TIMIT-LQ y 320 de DF-TIMIT-HQ. Se extrajeron imágenes (*frames*) de estos vídeos y se obtuvieron 4.073 imágenes falsas y 2.597 imágenes reales.
6. Para el experimento 7 se extrajo el rostro a cada una de las imágenes extraídas en el numeral 5, usando la librería OpenCV.
7. Para el experimento 8, se extrajo el rostro a cada una de las imágenes extraídas en el numeral 5 usando la librería OpenCV, y se le aplicó el método ELA con un ratio de error del 90 %.

En la fase de modelado, se entrenó una red convolucional Xception preentrenada con Imagenet, a la cual se le agregó una capa de clasificación compuesta por un GlobalAveragePooling2D, una ReLU (*Rectified Linear Unit*) y un softmax. Esta arquitectura de red acepta imágenes de 299 x 299 pixeles. Se efectuaron ocho experimentos con variaciones de los datos de entrenamiento y evaluación de la siguiente manera: en los experimentos del 1 al 6 se tomaron 7.024 imágenes falsas y 7.024 imágenes reales para entrenamiento, la evaluación se hizo con 680 imágenes falsas y 356 imágenes reales, las imágenes usadas para evaluación fueron extraídas del conjunto de vídeos definidos por Celeb para evaluación, para así poder comparar los resultados obtenidos con otras propuestas de detección; en los experimentos 7 y 8, se tomaron 2.624 imágenes falsas y 1.686 imágenes reales para entrenamiento, la evaluación se hizo con 1.449 imágenes falsas y 911 imágenes reales.

Para la fase de evaluación, se escogió el mejor modelo teniendo en cuenta las métricas de *accuracy* (por su fácil interpretación) y de AUC (porque permite la comparación con otros métodos de detección existentes), resultantes de evaluar con los vídeos de evaluación del conjunto de datos Celeb-DF. También se tuvo en cuenta la *accuracy* y el AUC resultantes de evaluar los vídeos que se generaron en la fase de comprensión del negocio (tres pares de vídeos).

Respecto de la metodología en general, solo fue necesaria una iteración sobre el proceso, dado que cuando se necesitaron cambios, se regresó a etapas previas buscando completar tareas o corregir errores.

9.4. Resultados

Durante el proyecto se utilizó como lenguaje de programación Python 3 junto con Jupyter Notebook y las librerías de scikit-learn, numpy, pandas, dlib, OpenCV, os, pylab, PIL y Keras.

Como parte del proceso de clasificación de datos, en la fase de comprensión de los mismos, se planteó la hipótesis de que las métricas de evaluación de calidad para un vídeo original respecto de sus compresiones podían servir para comparar clips de vídeo diferentes usando un vídeo como referencia. Las métricas evaluadas fueron la VMAF, desarrollada por Netflix, que busca emular la percepción humana para determinar la calidad, y PSNR, que mide el nivel de distorsión midiendo el error cuadrático medio entre la señal original y la distorsionada.

Para evaluar esta hipótesis se usó el conjunto de datos DFDC [14]. Inicialmente se tomaron seis vídeos del conjunto de datos (1920 x 1080): dos de calidad baja, dos de calidad media y dos de calidad alta, con base en una métrica subjetiva. Se usó el software libre "ffmpeg-quality-metrics" para obtener las métricas VMAF y PSNR. Los resultados de ambas métricas resultaron coherentes con lo clasificado.

Como se trataba de muy pocos datos, se realizó la misma tarea con cincuenta vídeos (usando siempre un vídeo como referencia), los que se verificaron con percepción humana, obteniendo una *accuracy* de 42 %. Lo anterior permitió concluir que las métricas de evaluación de calidad para un vídeo original respecto a sus compresiones no funcionan para comparar la calidad de clips de vídeos diferentes y por lo tanto, no son útiles para clasificar el contenido visual según la calidad.

Al no encontrar un método que permitiera clasificar contenido visual evaluando la calidad de forma algorítmica, se decidió clasificar el contenido como conjunto de datos de primera generación y de segunda generación, donde: los de primera generación incluyen caras sintetizadas de baja calidad, límites de empalme visibles, falta de coincidencia de colores, partes visibles

de la cara original y orientaciones faciales sintetizadas inconsistentes; y los de segunda generación mejoran la cantidad y los problemas de calidad que presentan los conjuntos de datos de primera generación de *deepfakes* [15]. Los conjuntos de datos de primera generación empleados en este proyecto fueron: DF-TIMIT y FF-DF; y los de segunda generación, DFD, DFDC y Celeb-DF.

Experimento Xception # 1

En la fase de modelado, se entrenó el modelo con los datos especificados en la fase de preparación de datos, especialmente con las imágenes completas. Se configuró una red Xception preentrenada con ImageNet, con: learning_rate = 1e-4, epoch = 50 y batch_size = 16. Además, se fijaron los pesos de toda la red, exceptuando la última capa (clasificador).

En la fase de evaluación se utilizó el conjunto de datos de evaluación provisto por Celeb-DF, lo que dio como resultado una *accuracy* de 40,34 % y un AUC del 52,75 %. Además se obtuvo la matriz de confusión que se presenta en la Figura 9.5.

		Reales	
		Verdadera	Fake
Predichos	Verdadera	4.562	129
	Fake	1.095	3.044

Figura 9.5. Matriz de confusión: experimento 1

Experimento Xception # 2

En la fase de modelado, se entrenó el modelo con los datos especificados en la fase de preparación de datos, pero utilizando solo rostros. Se configuró una red Xception preentrenada con ImageNet, con learning_rate = 1e-4, epoch = 50 y batch_size = 16. Además, se fijaron los pesos de toda la red exceptuando la última capa (clasificador).

En la fase de evaluación se utilizó el conjunto de datos de evaluación provisto por Celeb-DF, lo que dio como resultado una *accuracy* de 55,86 % y un AUC del 62,26%. Además se obtuvo la matriz de confusión que se presenta en la Figura 9.6.

		Reales	
		Verdadera	Fake
Predichos	Verdadera	126	161
	Fake	230	519

Figura 9.6. Matriz de confusión: experimento 2

Experimento Xception # 3

En la fase de modelado, se entrenó el modelo con los datos especificados en la fase de preparación de datos con la implementación de ELA y el uso de solo rostros. Se configuró una red Xception preentrenada con ImageNet, con: learning_rate = 1e-4, epoch = 50 y batch_size = 16. Además, se fijaron los pesos de toda la red exceptuando la última capa (clasificador).

En la fase de evaluación, se utilizó el conjunto de datos de evaluación provisto por Celeb-DF, lo que dio como resultado una *accuracy* de 49,03 % y un AUC del 53,68 %. Además se obtuvo la matriz de confusión que se presenta en la Figura 9.7.

		Reales	
		Verdadera	Fake
Predichos	Verdadera	244	416
	Fake	112	264

Figura 9.7. Matriz de confusión: experimento 3

Experimento Xception # 4

En la fase de modelado, se entrenó el modelo con los datos especificados en la fase de preparación de datos (imágenes completas). Se configuró una red Xception preentrenada con ImageNet, con: learning_rate = 1e-4, epoch = 50 y batch_size = 16.

En la fase de evaluación, se utilizó el conjunto de datos de evaluación provisto por Celeb-DF, lo que dio como resultado una *accuracy* de 54,15 % y un AUC del 63,53 %. Además se obtuvo la matriz de confusión que se presenta en la Figura 9.8.

		Reales	
		Verdadera	Fake
Predichos	Verdadera	333	452
	Fake	23	228

Figura 9.8. Matriz de confusión: experimento 4

Experimento Xception # 5

En la fase de modelado, se entrenó el modelo con los datos especificados en la fase de preparación de datos. Se configuró una red Xception preentrenada con ImageNet, con: learning_rate = 1e-4, epoch = 50 y batch_size = 16.

En la fase de evaluación, se utilizó el conjunto de datos de evaluación provisto por Celeb-DF, lo que dio como resultado una *accuracy* de 81,27 % y un AUC del 83,99 %. Además se obtuvo la matriz de confusión que se presenta en la Figura 9.9.

		Reales	
		Verdadera	Fake
Predichos	Verdadera	330	168
	Fake	26	512

Figura 9.9. Matriz de confusión: experimento 5

Experimento Xception # 6

En la fase de modelado, se entrenó el modelo con los datos especificados en la fase de preparación de datos, utilizando ELA y solo rostros. Se configuró una red Xception preentrenada con ImageNet, con: learning_rate = 1e-4, epoch = 50 y batch_size = 16.

En la fase de evaluación, se utilizó el conjunto de datos de evaluación provisto por Celeb-DF, lo que dio como resultado una *accuracy* de 60,91 % y un AUC del 65,74 %, además se obtuvo la matriz de confusión que se presenta en la Figura 9.10.

		Reales	
		Verdadera	Fake
Predichos	Verdadera	289	338
	Fake	67	342

Figura 9.10. Matriz de confusión: experimento 6

Experimento Xception # 7

En la fase de modelado, se entrenó el modelo con los datos especificados en la fase de preparación de datos. Se configuró una red Xception preentrenada con ImageNet, con: learning_rate = 1e-4, epoch = 50 y batch_size = 16.

En la fase de evaluación se utilizó el conjunto de datos de evaluación que incluye los cinco conjuntos de datos seleccionados para este proyecto, lo que dio como resultado una *accuracy* de 92,12 % y un AUC del 92,15 %. Además se obtuvo la matriz de confusión que se presenta en la Figura 9.11.

		Reales	
		Verdadera	Fake
Predichos	Verdadera	841	70
	Fake	116	1.333

Figura 9.11. Matriz de confusión: experimento 7

Experimento Xception # 8

En la fase de modelado, se entrenó el modelo con los datos especificados en la fase de preparación de datos, aplicando rostros y la implementación de ELA. Se configuró una red Xception preentrenada con ImageNet, con: learning_rate = 1e-4, epoch = 50 y batch_size = 16.

En la fase de evaluación se utilizó el conjunto de datos de evaluación que incluye los cinco conjuntos de datos seleccionados para este proyecto, lo que dio como resultado una *accuracy* de 60,90 % y un AUC del 75,95 %. Además se obtuvo la matriz de confusión que se presenta en la Figura 9.12.

		Reales	
		Verdadera	Fake
Predichos	Verdadera	684	336
	Fake	227	1.113

Figura 9.12. Matriz de confusión: experimento 8

En la Tabla 9.8 se resumen los resultados y las características de los métodos evaluados en cada experimento.

Tabla 9.8. Matriz de comparación de resultados

Exp.	Descripción	Imágenes de entrenamiento	Imágenes de evaluación	Kappa	Accuracy (%)	AUC (%)
#1	Imágenes completas, Xception entrenamiento del clasificador	7.024 falsas, 7.024 reales de Celeb-DF	680 falsas, 356 reales de Celeb-DF	0.04	40,34	52,75
#2	Rostros, Xception entrenamiento del clasificador	7.024 falsas, 7.024 reales de Celeb-DF	680 falsas, 356 reales de Celeb-DF	0.12	55,86	62,26
#3	Rostros + ELA, Xception entrenamiento del clasificador	7.024 falsas, 7.024 reales de Celeb-DF	680 falsas, 356 reales de Celeb-DF	0.06	49,03	53,68
#4	Imágenes completas, Xception entrenamiento completo	7.024 falsas, 7.024 reales de Celeb-DF	680 falsas, 356 reales de Celeb-DF	0.21	54,15	63,53
#5	Rostros, Xception entrenamiento completo	7.024 falsas, 7.024 reales de Celeb-DF	680 falsas, 356 reales de Celeb-DF	0.62	81,27	83,99
#6	Rostros + ELA, Xception entrenamiento completo	7.024 falsas, 7.024 reales de Celeb-DF	680 falsas, 356 reales de Celeb-DF	0.26	60,91	65,74
#7	Rostros, Xception entrenamiento completo	2.624 falsas, 1.686 reales de Celeb-DF, DFDC, FF- DF, DFC, DF- TIMIT	1.449 falsas, 911 reales de Celeb-DF, DFDC, FF-DF, DFC, DF-TIMIT	0.84	92,12	92,15
#8	Rostros + ELA, Xception entrenamiento completo	2.624 falsas, 1.686 reales de Celeb- DF, DFDC, FF- DF, DFC, DF-TIMIT	1.449 falsas, 911 reales de Celeb-DF, DFDC, FF-DF, DFC, DF-TIMIT	0.51	60,90	75,95

Por otra parte, al evaluar el modelo resultante del experimento 7, con 914 imágenes reales y 914 alteradas de los rostros de los videos usados para la generación de *deepfake,* en la fase de comprensión del negocio, se obtuvo una *accuracy* de 94,87 % y un AUC del 94,87%, además de la matriz de confusión que se presenta en la Figura 9.13.

		Reales	
		Verdadera	Fake
Predichos	Verdadera	953	79
	Fake	21	895

Figura 9.13. Matriz de confusión: experimento 7 con el conjunto de datos propio

Análisis de resultados

Comparando los experimentos 1 y 2, en donde se entrena y se evalúa con la misma cantidad imágenes y se usa el mismo modelo (Xception con entrenamiento solo en el clasificador), se obtiene una *accuracy* del 55,86 % en el experimento que es entrenado con rostros (experimento 2), valor significativamente mayor al 40,34 % de la *accuracy* del experimento que se entrena con imágenes completas (experimento 1). Lo mismo sucede con los experimentos 4 y 5, en donde el experimento 5 tiene una *accuracy* considerablemente más alta (81,27 %) que la del experimento 4 (54,15 %).

Comparando los experimentos 2 y 3, en donde se entrena y se evalúa con las mismas imágenes y se usa el mismo modelo (Xception con entrenamiento solo en el clasificador), se obtiene una *accuracy* del 55,86 % en el experimento que no hace uso de ELA (experimento 2), valor ligeramente mayor al 49,03 % de la *accuracy* del experimento que usa ELA (experimento 3). Lo mismo sucede con los experimentos 5 y 6, en donde el experimento 5 tiene una *accuracy* considerablemente más alto (81,27 %) que el del experimento 6 (60,91 %). Esto también se presenta en los experimentos 7 y 8, en donde el experimento 7 obtiene una *accuracy* de 92,12 %, cifra considerablemente mayor a la *accuracy* del experimento 8 (60,90 %).

Contrastando los experimentos 1 y 4, en donde se entrena y se evalúa con las mismas imágenes, se obtiene una *accuracy* del 54,15 % en el experimento donde no se fijan los pesos de la red (experimento 4), un valor considerablemente mayor

al 40,34 % de la *accuracy* del experimento que fija los pesos de la red entrenando solo el clasificador (experimento 1). Lo mismo sucede con los experimentos 2 y 5, en donde el experimento 5 tiene una *accuracy* considerablemente alta (81,27 %) que la del experimento 2 (55,86 %). Esto también se presenta en los experimentos 3 y 6, en donde el experimento 6 obtiene una *accuracy* de 60,91 %, considerablemente mayor a la *accuracy* de 49,03 % del experimento 3.

El experimento 7, que se entrenó y evaluó con todos los conjuntos de datos seleccionados para este proyecto, obtuvo una *accuracy* del 92,12 %, la mayor de todos los experimentos realizados. Al calcular este valor para cada conjunto de datos individualmente, se encuentra: 100 % para DF-TIMIT; 6,62 % para FF-DF; 95,25% para Celeb-DF; 85,9% para DFDC; y 81 % para DeepFakeDetection.

Al evaluar el modelo resultante del experimento 7 con el conjunto de datos generado en la fase de compresión del negocio, se evidencia su sobresaliente capacidad para clasificar el contenido visual real y alterado generado con la herramienta Faceswap, lo que valida el nivel de detección del modelo.

De los seis experimentos entrenados y evaluados, solo con el conjunto de datos Celeb-DF 4 obtuvo un AUC mayor que los obtenidos por los métodos de detección expuestos en el estado del arte (Figura 9.14). El experimento 5 fue el que obtuvo la mayor AUC.

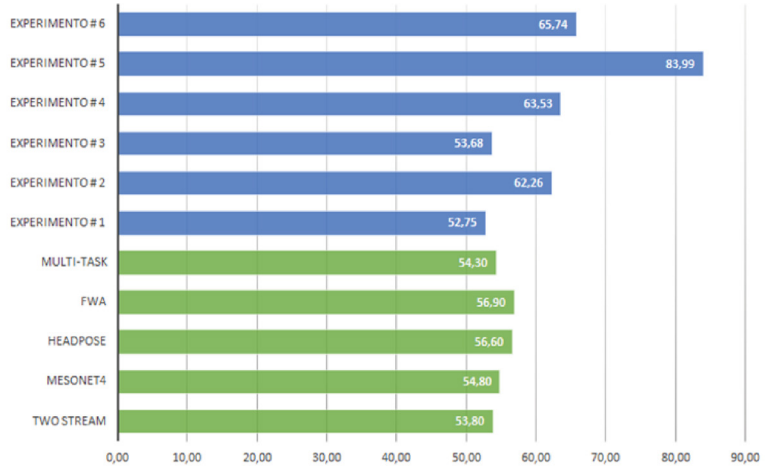

Figura 9.14. Comparativo de las AUC del estado del arte y los experimentos ejecutados

Tal como se aprecia en la Tabla 9.9, el experimento 7, que fue entrenado con rostros de los cinco conjuntos de datos seleccionados obtuvo valor mayor de AUC que los métodos de detección expuestos en el estado del arte.

Tabla 9.9. Comparación del % de AUC entre métodos de detección del estado del arte y el experimento 7

Método	DF-TIMIT	FF-DF	DFD	DFDC	Celeb-DF
Two Stream	73,5	70,1	52,8	61,4	53,8
Mesonet4	68,4	84,7	76,0	75,3	54,8
Headpose	53,2	47,3	56,1	55,9	56,6
FWA	93,2	80,1	74,3	72,7	56,9
Multi-task	55,3	76,3	54,1	53,6	54,3
Experimento # 7	100	96,62	81,0	85,9	95,25

9.5. Conclusiones y recomendaciones

En este proyecto se propuso un método de detección que permite clasificar contenido visual como real o falso (*real*, *fake*) a partir de una red convolucional Xception, un preprocesamiento de imágenes (extracción de rostro) y una optimización del umbral de decisión de la red. Las conclusiones obtenidas en el proceso se listan a continuación.

- Las métricas de evaluación de calidad para un vídeo original respecto de sus compresiones (VMFA, PSNR) no funcionan para comparar la calidad de imágenes diferentes, por lo tanto no es útil para clasificar el contenido visual según la calidad.
- Para el problema de detección de *deepfake*, el modelo Xception obtiene mejores resultados cuando se entrena y se evalúa con imágenes que contienen únicamente rostros, esto se debe a que las imágenes completas contienen información que no es relevante para determinar si una imagen es real o está alterada, lo que hace que el modelo no se enfoque en las características relevantes para hacer la distinción.
- La aplicación del método forense de análisis del nivel de error sobre los datos de entrenamiento y evaluación no mejora el nivel de detección del método, por el contrario, lo desmejora y hace más difícil para la red reconocer patrones diferenciadores entre imágenes reales y alteradas.

- La red Xception mejora su nivel de detección cuando no se fijan los pesos del entrenamiento con Imagenet, es decir la red mejora su predicción cuando se entrena completamente y no solamente al clasificador de la red.
- El método presenta una leve mejoría en la predicción de la clase (*real* o *fake*) al calcular un umbral de decisión que maximiza el AUC.
- Se evaluó un método capaz de detectar contenido visual real y contenido visual falso para un conjunto de datos compuesto por datos de: Celeb-DF, DFDC, DFD, FF-DF, DF-TIMIT, con una *accuracy* de 92,12 % y un AUC del 92,15 % (experimento 7). Este método dio como resultado al menos un 81 % de *accuracy* para cada conjunto de datos de manera individual. Además, obtuvo el mayor valor de AUC para cada conjunto de datos de manera individual, comparado con los métodos detectores expuestos en el estado del arte.

Como trabajo a futuro se propone:

- experimentar con otro tipo de métodos forenses como el análisis de ruido (*noise analysis*) aplicados a los datos, con el objetivo de que los algoritmos de *deep learning* puedan identificar con mayor facilidad artefactos distintivos entre los datos reales y alterados;
- realizar un entrenamiento del método aquí propuesto con una mayor cantidad de imágenes y evaluar si esto mejora la predicción; y
- llevar a producción el método propuesto, mediante un aplicativo web o móvil que permita a un usuario identificar si un vídeo o imagen es real o está alterado con *deepfake*.

Los recursos (documentos, gráficos, códigos, etc.) obtenidos durante la ejecución del proyecto están disponibles en [16].

Referencias

[1] R. Chesney, y D. Citron, "Deepfakes and the new disinformation war: the coming age of post-truth geopolitics," *Foreign Affairs*, vol. 98, No. 1, pp. 147–155, 2019.

[2] O. Schwart, (2018, Nov. 2), "You thought fake news was bad? Deep fakes are where truth goes to die," *The Guardian*. Disponible: https://www.theguardian.com/technology/2018/nov/12/deep-fakes-fake-news-truth

[3] M. E. Weems, "Fake America great again?" *Qualitative Inquiry*, vol. 23, pp. 168–170, 2017. https://doi.org/10.1177/1077800416674752

[4] M. Koopman, A. M. Rodriguez, y Z. Geradts, "Detection of deepfake video manipulation," *Proceedings of the 20th Irish Machine Vision and Image Processing Conference*, 2018, pp. 133–136.

[5] N. Krawetz, "A picture's worth," documento incluido en Black Hat Briefings, 2007). Disponible: https://www.blackhat.com/presentations/bh-usa-07/Krawetz/Whitepaper/bh-usa-07-krawetz-WP.pdf

[6] P. Zhou, X. Han, V. I. Morariu, y L. S. Davis, "Two-stream neural networks for tampered face detection," en *2017 IEEE conference on computer vision and pattern recognition workshops (CVPRW)*, pp.1831-1839. https://doi.org/10.48550/arXiv.1803.11276

[7] Y. Li y S. Lyu, "Exposing deepfake videos by detecting face warping artifacts," *arXiv*:1811.00656 [cs.CV]. https://doi.org/10.48550/arXiv.1811.00656

[8] D. Afchar, V. Nozick, J. Yamagishi, e I. Echizen, "MesoNet: a compact facial vídeo forgery detection network", *arXiv*.1809.00888

[9] X. Yang, Y. Li, y S. Lyu, "Exposing deep fakes using inconsistent head poses," en *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing, Proceedings*, 2019-May, pp. 8261–8265. https://doi.org/10.1109/ICASSP.2019.8683164

[10] H. H. Nguyen, F. Fang, J. Yamagishi, e I. Echizen, "Multi-task learning for detecting and segmenting manipulated facial images and videos," en *Proceedings of the IEEE International Conference on Biometrics: Theory, Applications and Systems (BTAS), 2019*, https://doi.org/10.48550/arXiv.1906.06876

[11] P. Korshunov, y S. Marcel, Deepfakes: a new threat to face recognition? Assessment and detection," *arXiv preprint arXiv*:1812.08685, 2018.

[12] A. Rossler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, y M. Nießner, "Faceforensics++: Learning to detect manipulated facial images," en *Proceedings of the IEEE International Conference on Computer Vision*, 2019. https://arxiv.org/pdf/1901.08971.pdf

[13] L. Li, J. Bao, T. Zhang, H. Yang, D. Chen, F. Wen, y B. Guo, "Face X- ray for more general face forgery detection," *arXiv*:1912.13458 [cs.CV], 2019.

[14] B. Dolhansky, R. Howes, B. Pflaum, N. Baram, y C. C. Ferrer, "The deepfake detection challenge (DFDC) preview dataset," *arXiv preprint arXiv*:1910.08854, 2019.

[15] Y. Li, X. Yang, P. Sun, H. Qi, y S. Lyu, "Celeb-DF: a new dataset for deepfake forensics," *arXiv*:1909.12962 [cs.CR], 2019.

[16] C.C Urcuqui. (2021). Metodo_de_detección_de_deepfake. *i2tResearch* [repositorio GitHub]. Disponible: https://github.com/i2tResearch/Ciberseguridad_web/tree/master/Deepfake/Metodo_de_deteccion_de_Deepfake

Capítulo 10
Sistema para la detección de deepfake de sonido

Cristhian Eduardo Castillo, Kevin Zarama, Christian Camilo Urcuqui

10.1. Introducción

Deepfake es el nombre que se le da al contenido generado mediante inteligencia artificial, que sin serlo parece real. Con la de aparición de tecnologías como las redes generativas antagónicas (*Generative Adversarial Networks*, GAN) [1] en 2014, el *deepfake* se empezó a extender y a mejorar ampliamente, al punto que hoy se logran audios, videos e imágenes falsos, cada vez más realistas, tanto que es difícil distinguir entre contenido real y sintético. Este desarrollo ha traído consigo la generación de una gran cantidad de noticias falsas acompañadas de videos, imágenes o audios alterados, en las que es casi imposible determinar qué es contenido cierto y qué es contenido manipulado.

La investigación en el campo del *deepfake* de audio no tiene la misma importancia y relevancia que la que se da al *deepfake* de video e imágenes, dos formatos en gran auge por su mayor disponibilidad. Sin embargo, el *deepfake* de audio es igualmente importante y puede ser aún más peligroso por su potencial de poner en situaciones de decisiones rápidas a las personas, por ejemplo, con una llamada realizada con inteligencia artificial, el emisor se puede hacer pasar por el dueño de una compañía y pedir que se realice una serie de pagos.

El termino *deepfake* se originó en el 2017 dentro de Reddit luego de que un usuario llamado ”Deepfake” posteara haber desarrollado un algoritmo que permitía transponer rostros de celebridades porno [2]. A partir de ese momento empezaron a surgir muchas publicaciones bajo el concepto de *deepfake* con diferentes variantes (audio, imágenes y video).

Bajo ese concepto, BuzzFeed publicó un video falso de Barack Obama, donde “el presidente” advertía que el país estaba entrando en una era en donde sus enemigos podrían generar contenido para hacer parecer real un

hecho o mensaje falso [3] y usarlo para la manipulación de conciencia o ingeniería social.

El *deepfake* entonces, amenaza con volver muy difusa la línea entre la verdad y el engaño, lo que lleva a que las personas no puedan diferenciar lo real de lo falso y se creen ambientes llenos de desconfianza.

Por lo anterior, es importante tener la capacidad de identificar cuándo un sonido ha sido creado o manipulado por medio de inteligencia artificial, para poder diferenciarlo de sonidos reales y así evitar situaciones de chantaje, intimidación y sabotaje [4].

La alteración de la información ha visto un gran aumento con la llegada de modelos de inteligencia artificial que facilitan esta tarea. Dichos modelos, que se dedican a la falsificación de información, se conocen como modelos de *deepfake*, su aparición se ha convertido en un problema debido al mal uso que se le puede, lo que incluye cosas como la falsificación de pruebas en juicios legales, la difamación y la creación de noticias falsas, entre otras aplicaciones mal intencionadas.

El objetivo general del proyecto es desarrollar un sistema para la detección de contenido auditivo alterado, para lo cual se establecieron como objetivos específicos: recopilar un conjunto de datos de datos reales y alterados; proponer un método para detección de *deepfake* auditivo; y evaluar el método propuesto con otros existentes.

10.2. Estado del arte

Previo a la realización de este proyecto, se revisaron cinco trabajos que abordan desde distintas perspectivas el problema objeto de esta investigación: *DeepFake audio detection*; *Adversarial audio synthesis (WaveGAN)*; *DeepSonar: Towards effective and robust detection of AI synthesized fake voices*; *Generalization of audio deepfake detection*; y Método de detección de *deepfake* mediante técnicas de *machine learning*.

Los criterios de comparación entre ellos y el proyecto actual (Tabla 10.1) fueron: la tecnología usada para identificar audio alterado; el acceso público al código fuente; el acceso público a los *datasets*; el uso de inteligencia artificial para la detección del audio alterado; y las métricas usadas para la evaluación de los mecanismos.

Tabla 10.1. Comparación de los métodos de detección de deepfake de audio

Propiedad	Tecnología usada	Acceso al código fuente	Métricas
DeepFake audio detection	Redes neuronales convolucionales	Si	Exactitud = 85 %
WaveGAN	Redes generativas antagónicas	Si	Puntaje SC09 = 4.7
DeepSonar	Redes neuronales profundas	No	Exactitud = 98.1 %
Generalization of audio deepfake detection	Redes neuronales residuales	No	Puntaje EER = 1.26 %
Método de detección de deepfake mediante técnicas de ML	Red convolucional Xception + Transfer learning + ImageNet.	Si	Accuracy = 92,12 %; AUC = 92,15 %.
Proyecto actual	Redes neuronales convolucionales	Si	Exactitud = 92.23 %

DeepFake audio detection

En este proyecto, Delgado, Evans, Kinnunen, et al. [5] proponen realizar modelos para la verificación automática de hablante, es decir para la detección de audio *deepfake*. En su trabajo utilizan el conjunto de datos ASVSpoof de Google, liberado en 2019, para fomentar el desarrollo de detección de audio *deepfake*. Su modelo consta de una red neuronal profunda que utiliza la convolución temporal.

Una descripción general de alto nivel de su arquitectura, se presenta en la Figura 10.1. Al inicio, el audio sin procesar, se preprocesa y se convierte en un espectrograma de frecuencia Mel, este espectrograma es la entrada para el modelo; luego, el modelo realiza convoluciones sobre la dimensión de tiempo del espectrograma y usa una agrupación enmascarada para evitar el sobreajuste; finalmente, la salida se pasa a una capa densa y una función de activación sigmoide, que finalmente genera una probabilidad entre 0 (falso) y 1 (real) [6].

En su evaluación, el modelo obtuvo una exactitud de: 99 % en entrenamiento, 95 % en validación y 85 % en prueba, como es evidente, todos ellos son muy buenos resultados, lo que indica que se trata de un método de detección muy fiable y seguro.

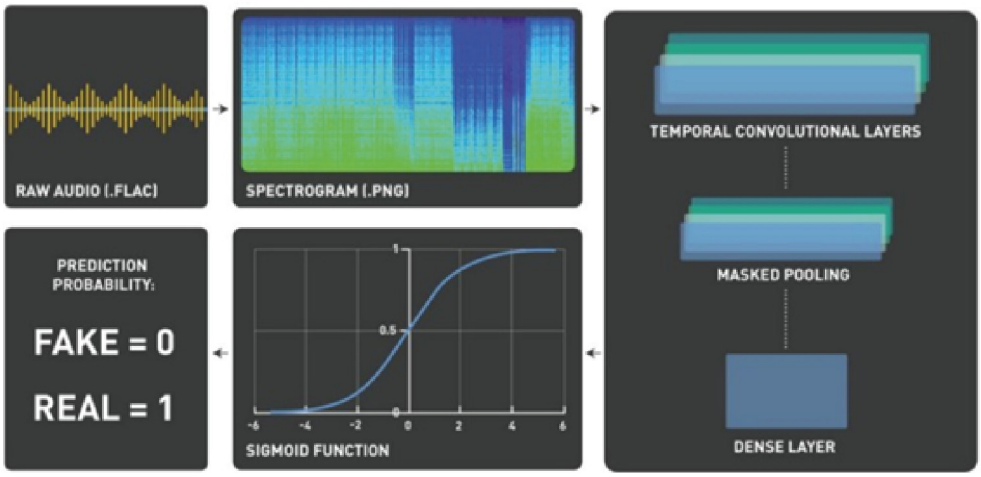

Figura 10.1. Arquitectura presente en el modelo DeepFake Audio Detection [6]

Adversarial Audio Synthesis (WaveGAN)

Miller, Donahuc y McAuley [7] usan redes adversarias generativas (GAN) para la generación y detección de contenido de *deepfake* de audio. En su trabajo, exponen la posibilidad de ver las señales de audio como series de tiempo, de tal manera que en cada momento el sonido se pueda representar como un valor, y así se vea al sonido (o amplitud a) como una función del tiempo. Ver las señales de audio como una serie de tiempo permite representar a un audio como un vector, donde $t = f(t)$, donde la función f es continua, por naturaleza, por lo que para convertir una señal en un conjunto finito de números, se debe elegir una frecuencia de muestreo.

También plantean que el uso de convoluciones transpuestas para escalar el audio, desde el ruido hasta un vector, tiene un impacto en el audio sintetizado, ya que produce artefactos de tablero de ajedrez en las imágenes y su equivalente en señales de audio. Por esto, el discriminador podría aprender fácilmente a detectar audio falso basándose solo en este artefacto, deteriorando todo el proceso de entrenamiento.

El modelo de WaveGAN propone una solución llamada mezclado de fase, evitando convoluciones transpuestas y usando capas de muestreo superior (vecinos más cercanos) seguidas de convoluciones normales.

DeepSonar: Towards effective and robust detection of AI synthesized fake voices

En este proyecto, Wan et al. [8] construyen una red neuronal profunda para discernir las voces falsas sintetizadas por inteligencia artificial. Usan la función de activación de neuronas por capas con la conjetura de que pueden capturar las diferencias sutiles entre las voces falsas reales y sintetizadas por IA, al proporcionar una señal más limpia a los clasificadores que las entradas sin procesar. Sus experimentos se llevaron a cabo sobre tres conjuntos de datos: FoR, Sprocket-VC y MC-TT, los cuales contienen datos en inglés y chino.

Generalization of audio deepfake detection

Chen et al. [9] usan una red neuronal residual con función de pérdida LMCL y aumento de enmascaramiento de la frecuencia en línea, para forzar que la red neuronal aprenda más sobre la incorporación de características robustas. El modelo se evaluó sobre los datos de acceso lógico de ASVspoof 2019 (y sobre su versión con ruido), con el fin de simular escenarios más realistas. En el modelo que proponen, los investigadores reemplazan softmax con LCML y obtienen mejores, por lo que plantean que LCML es capaz de forzar al modelo para aprender características más robustas, con una mayor capacidad de generalización.

Método de detección de deepfake mediante técnicas de machine learning

Gutiérrez, Campaz y Díaz [10] (ver capítulo anterior de este libro), revisan varios métodos de detección de *deepfakes* en video y proponen uno propio, basado en una red convolucional Xception, que usa *transfer learning* mediante ImageNet, entrenada con rostros extraídos de varios conjuntos de datos de primera y segunda generación de *deepfake*. Con esta configuración obtuvieron una *accuracy* de 92,12 % y un AUC del 92,15 %.

10.3. La investigación

En el desarrollo del proyecto se utilizaron: metodologías ágiles para la gestión del proyecto, y la metodología CRISP-DM (*Cross-Industry Standard Process for Data Mining*), un estándar para el proceso de ciencia de datos, que fue descrito en la sección correspondiente al marco conceptual.

Metodologías ágiles

Las metodologías ágiles permiten adaptar la forma de trabajo a las condiciones de un proyecto: por una parte, se pueden realizar varias iteraciones del ciclo de vida del proyecto, y por otra, su esquema de trabajo es incremental, lo que le aporta flexibilidad y facilidad de respuesta para adaptar el proyecto a los cambios que puedan suceder. Uno de los principios de esta metodología es que los requisitos y las soluciones pueden evolucionar con el tiempo, según la necesidad del proyecto.

Se decidió realizar una iteración basada en la metodología ágil por cada una de las entregas. El proyecto cubrió las etapas de planificación, diseño, codificación y pruebas. Asimismo, se construyó un prototipo en Figma para el home (Figura 10.2), el demo (Figura 10.3) y el *paper* (Figura 10.4), para que sirvieran como guía para el desarrollo del sistema.

En cuanto a tecnologías, se decidió: desarrollar con apoyo en el sistema de Gitlab CD/CD; para el *frontend*, utilizar React Js junto con Apllo Client para la conexión con la API Graph Q (usada en el *backend*) y el servicio de Vercel para los despliegues; y para el *backend*, usar Python con Django, para construir la citada API GraphQL, y utilizar un entorno AWS (Amazon Web Service)

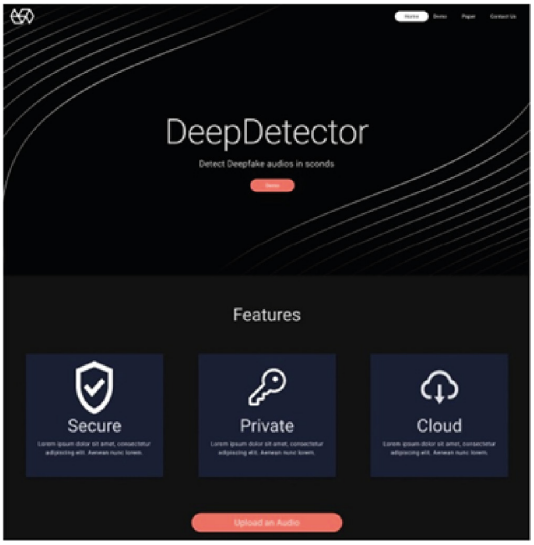

Figura 10.2. Prototipo del home[6]

Figura 10.3.Prototipo del demo[6]

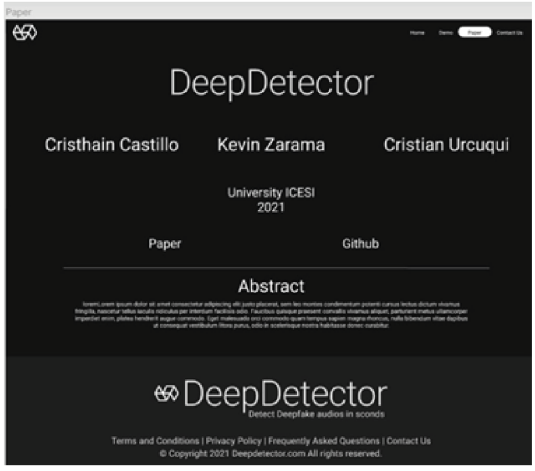

Figura 10.4. Prototipo del paper[6]

para su despliegue, compuesto por un sistema de S3 para almacenamiento, un EC2 para el API y un RDS para la base de datos. Asimismo, usando Gitlab se implementaron prácticas de DevOps, para así realizar despliegues e integraciones continuas (ver Figura 10.5) que permitieran reducir los tiempos de desarrollo. En la Figura 10.6 se describe el flujo del sistema y el *stack* de tecnologías que se usaron en el proyecto.

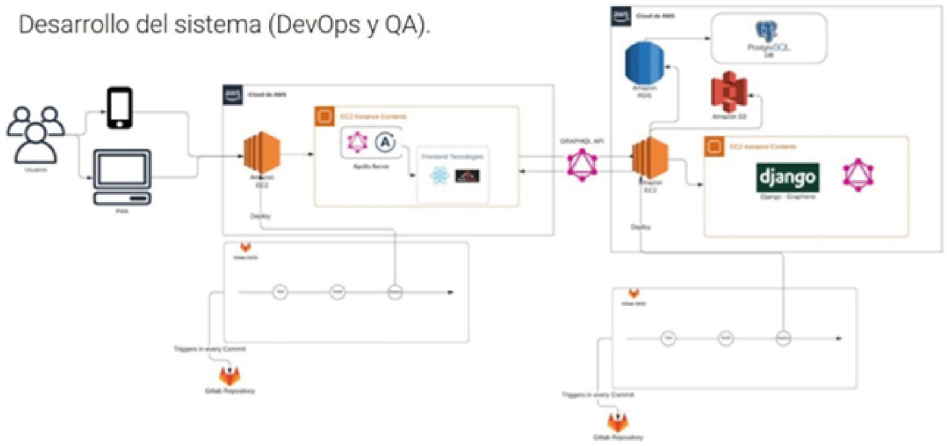

Figura 10.5. Despliegues continuos del sistema[6]

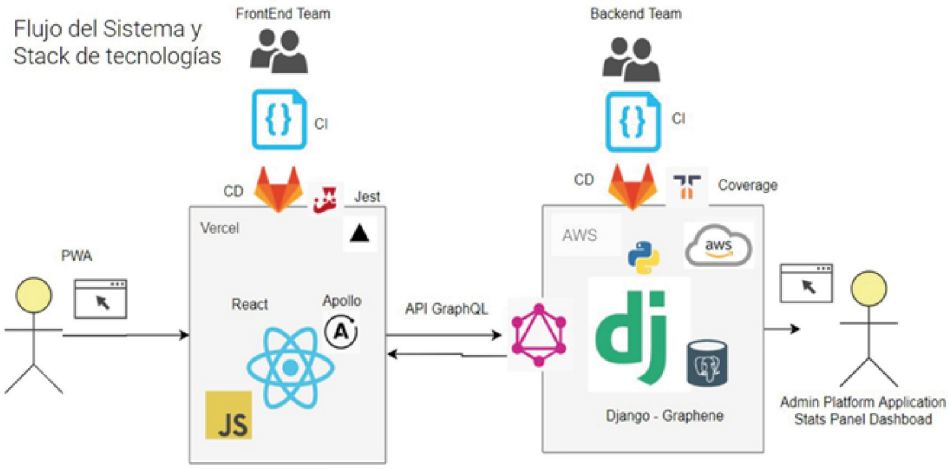

Figura 10.6. Flujo del sistema y stack de tecnologías usadas en el proyecto[6]

CRISP-DM

Para la fase de comprensión del negocio, se decidió determinar los objetivos de la minería de datos y realizar un diseño preliminar para lograr dichos objetivos. En esta fase también se estudiaron algunos métodos que detectan los *deepfakes* de sonido y se identificaron las estrategias que usadas para la detección por cada uno de ellos. De estas estrategias se eligió *fake voice detection*, la cual, tanto

para entrenar el modelo de síntesis de voz, como para discernir entre audio real y audio falso, usa representaciones visuales de clips de audio (espectrogramas). Como conjunto de datos, se seleccionó Google's 2019 ASVSpoof dataset [11], liberado por Google en el desafío ASVSpoof de 2019 para el desarrollo de modelos de detección de *deepfake* de sonido.

La fase de comprensión de los datos inició con la recolección y comprensión de los datos necesarios para lograr el objetivo del proyecto, incluyendo la identificación y valoración de los problemas que podrían tener. Una vez seleccionado el modelo, se continuó con la comprensión del *dataset* citado, el cual, como se observó, contiene algo más de 133.000 clips de audio (entre auténticos y falsos), con las particularidades descritas en la Tabla 10.2.

Tabla 10.2. Distribución de los datos de Google's ASVSpoof 2019 [11]

Subconjunto	Oradores		Declaraciones (#)			
			Acceso lógico		Acceso físico	
	Hombres	Mujeres	Autémticos	Falsos	Auténticos	Falsos
Entrenamiento	8	8	2.580	22.800	5.400	48.600
Desarrollo	12	12	2.548	22.296	5.400	24.300

La base para acceso lógico en ASVspoof 2019 es una base de datos estándar de síntesis de voz de varios hablantes, llamada VCTK2 [12]. Las declaraciones genuinas se recopilan de 107 hablantes (46 hombres, 61 mujeres), sin efectos significativos de ruido de canal o de fondo; las declaraciones falsificadas se generan a partir de datos genuinos, utilizando varios algoritmos de suplantación de identidad. El conjunto de datos completo se divide en tres subconjuntos: entrenamiento, desarrollo y evaluación.

En la fase de análisis, se seleccionaron todos los datos del *dataset*, no fue necesario realizar una limpieza previa porque el *dataset* estaba preparado específicamente para este fin en el desafío. Finalmente, se realizaron los espectrogramas de cada uno de los audios del conjunto de datos, con el fin de habilitarlos para ser introducidos en el modelo. Los espectrogramas son representaciones visuales de los sonidos. En la Figura 10.7 se puede notar la diferencia en la nitidez de las imágenes, lo que ocurre porque el audio con el que se generó el espectrograma de la derecha es falso.

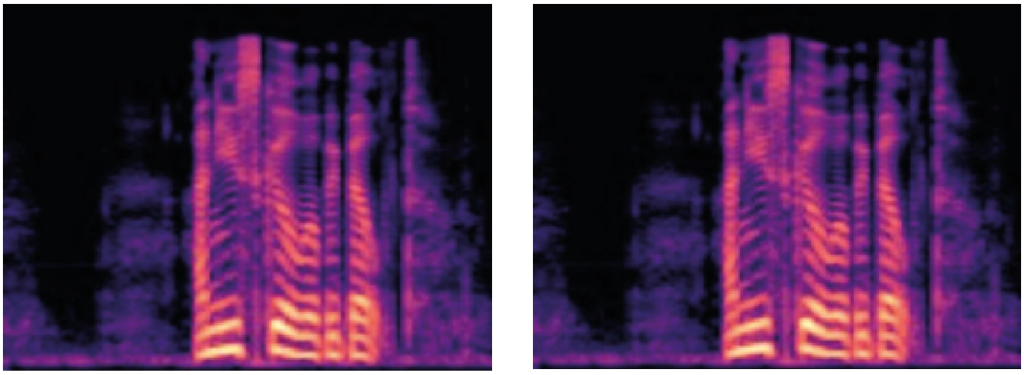

Figura 10.7. Espectrogramas: audio real (izquierda); audio falso (derecha)

En la fase siguiente se desarrolló un modelo basado en la arquitectura propuesta por Dessa IA lab [6] (ver *Deepfake audio detection*), como se indicó, en ella se usan redes neuronales profundas compuestas por capas de convolución temporal. La entrada del modelo son los espectrogramas Mel; el modelo los toma, realiza convoluciones que crean matrices bidimensionales (muy efectivas para tareas de visión artificial, como la clasificación y la segmentación de imágenes); usa una agrupación enmascarada para evitar el sobreajuste del modelo; y finalmente, su salida pasa primero a una capa densa y después a una función de activación sigmoide, para así obtener una probabilidad de predicción entre 0 y 1, en donde 0 significa "audio falso" y 1 "audio real" (Figura 10.8).

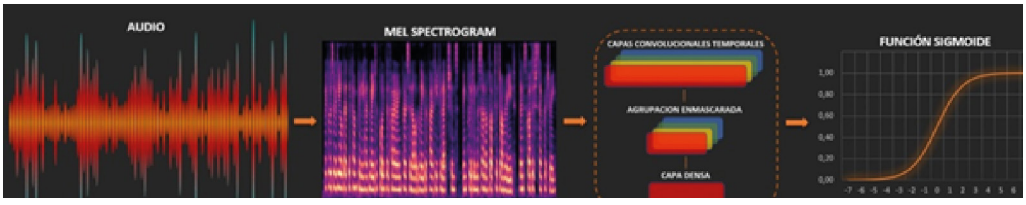

Figura 10.8. Arquitectura del modelo propuesto

En esta fase se realizaron dos experimentos (ver detalle en la sección Resultados): el primero consistió en armar un conjunto de datos más grande para el entrenamiento y las pruebas del modelo, mezclando el conjunto original

de datos de entrenamiento y prueba del modelo, el ASVSpoof2019, con una muestra del conjunto de datos de The fake-or-real dataset [13]; el segundo consistió en adicionar capas de ConvLSTM a la arquitectura del modelo que se tenía en un principio, para observar si añadir una dimensión temporal en el modelo mejora sus resultados.

Exactitud fue el criterio seleccionado para la fase de evaluación. El resultado, luego de correr el modelo cinco épocas, fue de 98.5 % en entrenamiento, 95.2 % en validación y 82.3 % en prueba.

En la fase de implementación, se llevó a cabo el ordenamiento de los datos obtenidos a partir del modelo y se realizó una revisión final para asegurar que cumplieran con los objetivos propuestos al inicio del proyecto. Como se mencionó, en esta fase se utilizaron, entre otros: GraphQL, para manejar las peticiones y las consultas; Django para manejar el *backend*; React para el *frontend*; y Apollo, para las peticiones del *frontend* hacia la API.

10.4. Resultados

Como se mencionó, durante el proyecto se realizaron dos experimentos: el primero, agregando datos al *dataset* original; el segundo, adicionando capas de ConvLSTM a la arquitectura del modelo.

En el primero se mezcló el *dataset* Asvspoof 2019, usado en el entrenamiento y las pruebas, con una muestra de Fake-or-Real (FoR) [13], un *dataset* con más de 195.000 declaraciones de personas reales y de audios generados por computadores. La muestra se conformó con 8.000 audios reales y 8.000 audios falsos. Esta adición tenía como finalidad: aumentar los datos con los cuales estaba trabajando el modelo; aumentar la proporción de datos reales en los datos de entrenamiento (10 % a 25 %); y observar el comportamiento del modelo con los nuevos datos. El resultado fue una exactitud de 99.25 % en entrenamiento, 96.78 % en validación y 87.8 % en pruebas.

Para el experimento 2, se adicionó una capa de ConvLSTM a la arquitectura del modelo. Estas capas se utilizan frecuentemente para predicciones espaciotemporales, por su estructura convolucional en las transiciones: de entrada a estado y de estado a estado. Sus características se usan para procesar series temporales de imágenes. La hipótesis detrás de este experimento es que agregar una capa de ConvLSTM al inicio del modelo (Figura 10.9), le otorga

una característica muy relevante para la clasificación de las imágenes, como reales o falsas, en la dimensión temporal de los espectrogramas. Su resultado fue una exactitud de 99.8 % en entrenamiento, 97.25 % en validación y 92.23 % en pruebas.

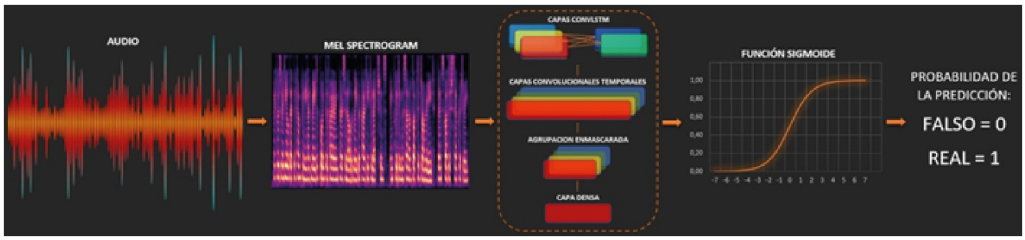

Figura 10.9. Arquitectura del modelo en el experimento 2, con una capa convLSTM añadida al inicio

Los resultados de estos dos experimentos permiten afirmar que la mezcla de los conjuntos de datos de ASVSpoof2019 y FoR mejoró el resultado inicial, y que la mejora fue aún mayor al agregar la capa de convLSTM a la arquitectura.

En la Tabla 10.3 se presenta la comparación de los resultados obtenidos luego del entrenamiento y los experimentos del modelo, con los resultados logrados por los modelos que se presentaron en el estado del arte (ver sus características en la Tabla 10.1).

Tabla 10.3. Comparación de los resultados de los métodos de detección de deepfake de audio

Proyecto	Resultado
DeepFake audio detection	Exactitud = 85 %
Adversarial Audio Synthesis (WaveGAN)	Puntaje SC09 = 4.7
DeepSonar: Towards effective and robust detection of AI synthesized fake voices	Exactitud = 98.1 %
Generalization of audio deepfake detection	Puntaje EER = 1.26 %
Método de detección de deepfake mediante técnicas de machine learning	Accuracy = 92,12 %; AUC = 92,15 %
Proyecto actual	Exactitud 92.23 %

10.5. Conclusiones y recomendaciones

En términos generales, este proyecto propuso una herramienta que permite clasificar audios entre reales y falsos a partir de una red neuronal convolucional, de su implementación se obtienen las siguientes conclusiones.

Es de vital importancia contar con un conjunto de datos balanceado o usar técnicas para contrarrestar problemas de balance al afrontar una tarea de clasificación de datos entre reales y falsos. Aumentar el conjunto de datos para contrarrestar el problema de balanceo, mejora la exactitud; hacer uso de capas LSTM convolucionales, más aun. Es claro que usar estas capas en problemas de clasificación audios representados en imágenes, como se hizo en el experimento 2, presenta muy buenos resultados, tal como sucede con otros tipos de imágenes. En el proyecto, con el experimento 2 se obtuvo la mayor exactitud

El uso de espectrogramas Mel es una buena aproximación para resolver el problema de clasificación de audios reales y falsos. Incluso en el peor escenario, con el modelo base sin capas LSTM y un conjunto de datos desbalanceado, se logró obtener una exactitud mayor al 80 %, un valor que aun cuando está lejos de las mejores propuestas, que alcanzan niveles de exactitud de 98 %, es una solución viable cuando no se cuenta con grandes recursos para implementar soluciones robustas que requieren un poder computacional mucho mayor. La dimensión temporal en los espectrogramas para la clasificación de audios es una característica que cuenta con un gran peso.

La etapa de preprocesamiento de los datos, esto es la de generación de los espectrogramas, es la más difícil y que mayor tiempo consume debido a que ellos deben estar construidos de tal forma que puedan pasar por las diferentes capas de un modelo convolucional. Esto es un problema que no se presenta en otros tipos de clasificación con imágenes, puesto que los espectrogramas provienen de un audio que cuenta con una dimensión temporal.

Como trabajo futuro se propone:

- verificar si el modelo planteado y los descritos en el estado del arte obtienen los mismos resultados con audios en idiomas distintos al inglés;
- encontrar los hiperparámetros que optimicen los resultados en el modelo;
- integrar la API (*Application Programming Interface*) con el *frontend* para generar una herramienta accesible al público; y

- usar prácticas de tareas asíncronas, como colas de tareas, para que los tiempos de respuesta del API no afecten la experiencia de usuario.

Los recursos (documentos, gráficos, códigos, etc.) obtenidos durante la ejecución del proyecto están disponibles en [14], con excepción de la presentación del proyecto realizada durante Ekoparty Security Conference 2021, disponible en [15].

Referencias

[1] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, y Y. Bengio, "Generative Adversarial Networks," *arXiv*: 1406.2661 [stat.ML], 2014.

[2] *Deepfakes: what are they and why would I make one?* (2019, jul.). Disponible: https://www.bbc.co.uk/bitesize/articles/zfkwcqt

[3] C. Silverman. (2018). *How to spot a deepfake like the Barack Obama–Jordan Peele video*. Disponible: https://www.buzzfeed.com/craigsilverman/obama-jordan-peele-deepfake-video-debunk-buzzfeed

[4] D. K. Citron y R. Chesney. (2019). *Deep fakes: a looming challenge for privacy, democracy, and national security*. Disponible: https://ssrn.com/abstract=3213954 or http://dx.doi.org/10.2139/ssrn.3213954

[5] H. Delgado, N. Evans, T. Kinnunen, K. et al. ASVspoof. *Inf. Téc*. https://www.asvspoof.org/

[6] IA lab Dessa, (2019). *Detecting audio deepfakes with AI*. Disponible: https://medium.com/dessa-news/detecting-audio-deepfakes-f2edfd8e2b35

[7] C. Donahuc J. McAuley, y M. Puckette, "Adversarial audio synthesis," *arXiv*:1802.04208 [cs.SD], 2019. Disponible: https://arxiv.org/abs/1802.04208

[8] R. Wan, F. Juefei-Xu, Y. Huang, Q. Guo, X. Xie, L. Ma, y Y. Liu, "DeepSonar: towards effective and robust detection of AI-synthesized fake voices," *arXiv*:2005.13770 [eess.AS], 2020. https://doi.org/10.48550/arXiv.2005.13770

[9] T. Chen, A. Kumar, P. Nagarsheth, G. Sivaraman, y E. Khoury, "Generalization of audio deepfake detection," en *Odyssey 2020 The*

Speaker and Language Recognition Workshop. Disponible: https://www.isca-speech.org/archive_v0/Odyssey_2020/pdfs/29.pdf

[10] S. Gutiérrez, B. Campaz, J. D. Díaz, "Método de detección de deepfake mediante técnicas de machine learning," trabajo de grado. Cali, Colombia: Universidad Icesi, 2020.

[11] J. Yamagishi, M. Todisco, Md. Sahidullah, H. Delgado, X. Wang, N. Evans, T. Kinnunen, K. A. Lee, V. Vestman, y A. Nautsch, *ASVspoof 2019: the 3rd automatic speaker verification spoofing and countermeasures challenge database* [sound], University of Edinburgh, The Centre for Speech Technology Research (CSTR), 2019. https://doi.org/10.7488/ds/2555.

[12] K. MacDonald, C. Veaux, y J. Yamagishi, "Superseded-CSTR VCTK corpus: English multi-speaker corpus for CSTR voice cloning toolkit," Inf. Téc., University of Edinburgh. The Centre for Speech Technology Research (CSTR), 2017.

[13] "The fake-or-real dataset," Inf. téc., APTLY lab Disponible: https://bil.eecs.yorku.ca/datasets/#:~:text=Datasets%20created%20by%20members%20of%20the%20APTLY%20lab%20can%20be%20found%20here.&text=The%20Fake%2Dor%2DReal%20(,classifiers%20to%20detect%20synthetic%20speech

[14] C.C Urcuqui. (2021). Deepaudio detector. *i2tResearch* [repositorio GitHub]. Disponible: https://github.com/i2tResearch/Ciberseguridad_web/tree/master/DeepAudio%20Detector

[15] C. Castillo, K. Zarama, y C. Urcuqui. (2021). *System for deepfake detection – DeepAudio Detector* [video], https://youtu.be/woV8phj5gxU

Índice de tablas

Índice de figuras

Acrónimos

ACL	Access-Control List
AdaIN	Adaptive Instance Normalization
ADWA	Anomaly Detection of Web-based Attacks
APK	Android Application Package
API	Application Programming Interface
APT	Advanced Persistent Threats
ARFF	Attribute-Relation File Format
ASUM-DM	Analytics Solutions Unified Method
AUC	Area Under the ROC Curve
AVD	Android Virtual Device
Avg-bps	Average Bits per Second
Avg_pps	Average packets per Second
AWS	Amazon Web Service
C&C	Comando & Control
CART	Classification And Regression Tree
CCI	Correctly Classified Instances
CNN	Convolutional Neural Networks
CRISP-DM	Cross-Industry Standard Process for Data Mining

CSV	Comma-Separated Values
DDoS	Distributed Denial of Services
DFDC	Deep Fake Detection Challenge
DHCP	Dynamic Host Configuration Protocol
DL	Deep Learning
DMSML	Defending Malicious Script attacks using Machine Learning classifiers
DMWH	Detection of Malicious Web pages based on Hybrid analysis
DNS	Domain Name System
DOM	Document Object Model
DoS	Denegation of Services
DPI	Deep Packet Inspection
DQEAF	Deep Q-network to Evade Antimalware engines Framework
DQL	Deep Q Learning
DRL	Deep Reinforcement Learning
DSSA	Detection of Server-side Web Attacks
EDA	Exploratory Data Analysis
ELA	Error Level Analysis
FoR	Fake-or-Real
FN	Falsos Negativos
FP	Falsos Positivos
GAN	Generative Adversarial Network
HAMW	Hybrid Approach for detecting Malicious Web pages using decision tree and naive Bayes algorithms
HPC	Hardware Performance Counters
HQ	High Quality
HTML	Hyper Text Markup Language

IA	Inteligencia Artificial
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IP	Internet Protocol
IPS	Intrusion Prevention Systems
IRC	Internet Relay Chat
IRLS	Iteratively Reweighted Least Squares
JSON-RPC	Remote Procedure Call Protocol Encoded in JSON
KDD	Knowledge Discovery in Databases
k-NN	k-Nearest Neighbors
LDA	Latent Dirichlet Allocation
LQ	Low Quality
MitM	Man-in-the-Middle
ML	Machine Learning
MLCMW	Machine Learning Classifiers to detect Malicious Websites
MLP	Multi Layered Perceptron
OGNL	Objects Graphics Navigation Library
OSI	Open Systems Interconnection
OWASP	Open Web Application Security Project
P2P	peer-to-peer
PDM	Procesos de Decisión de Markov
PoW	Proof of Work
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SAGAN	Self-Attention Generative Adversarial Networks
SecaaS	Security as a Service

SIEM	Security Information and Event Management
SL	Secure Learning
SVM	Support Vector Machine
TCP	Transmission Control Protocol
TIC	Tecnologías de la Información y las Comunicaciones
TNR	True Negative Rate
TPR	True Positive Rate
UCSB	Universidad de California Santa Barbara
UDP	User Datagram Protocol
UID	User ID
UNAD	Universidad Nacional Abierta y a Distancia
UTV	Universidad Técnica de Vienna
VN	Verdaderos Negativos
VP	Verdaderos Positivos
WEKA	Waikato Environment for Knowledge Analysis
XSS	Cross-Site Scripting

Acerca de los autores

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 Especialista en Deep Learning y Natural Language Processing; científico de datos en la industria de softwate; 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

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); profesor de tiempo completo y director del Grupo de Investigación en Informática y Telecomunicaciones (i2t) de la Universidad Icesi (Cali, Colombia); investigador senior (Minciencias); miembro senior del IEEE y Director Regional de IEEE COMSOC LATAM (2022-2024); consultor internacional; y miembro de Grupo de Estudio 1 de la Unión Internacional de Telecomunicaciones. anavarro@icesi.edu.co

Brayan Andrés Henao

Ingeniero de Sistemas con énfasis en desarrollo de software de la Universidad Icesi, su mayor experiencia se centra en el desarrollo backend y el desarrollo en la nube. Mediante el desarrollo de aplicaciones Cloud Native (buildpacks), contribuye a la comunidad Open Source. Sus áreas de interés profesional incluyen además la arquitectura de software y DevOps. bryanhenao96@gmail.com

Juan Sebastián Prada

Ingeniero de Sistemas de la Universidad Icesi, se desempeña como desarrollador de Java Backend en la firma Perficient (https://www.perficient.com/). juan.prada1@correo.icesi.edu.co

Andrés Felipe Pérez Belalcazar

Ingeniero de Sistemas con énfasis en ingeniería de software y desarrollo web de la Universidad Icesi, se desempeña como desarrollador de software para la firma DreamCode Software (https://www.dreamcodesoft.com/). Sus áreas de interés profesional son: desarrollo web, ciencia de datos y arquitectura de software. belalcazar180@gmail.com

Steven Bernal Tovar

Ingeniero de Sistemas de la Universidad Icesi con diplomado en Big Data Data Science por la Universidad de Cataluña (España). Se desempeña como coordinador del Club de Hacking de la Universidad Icesi. Sus áreas de interés profesional son: ciencia de datos, machine y deep learning, Natural Language Processing (NLP) y ciberseguridad. stevenazul1995@gmail.com

Julio César Gaviria Jaramillo

Ingeniero de Sistemas de la Universidad Icesi y durante su vida de estudiante miembro de su Grupo de Investigación en Informática y Telecomunicaciones (i2t). Sus áreas de interés profesional son: el desarrollo multiplataforma para móviles (Android, iOS), la seguridad informática y el machine learning. jcgj56@hotmail.com

Ánderson Ramírez

Ingeniero de Sistemas de la Universidad Icesi. Se desempeña como desarrollador frontend en la industria de software. Sus áreas de interés profesional son la ciencia datos y el desarrollo para móviles. anderson.ramirez.icesi@gmail.com

Jhoan Steven Delgado Villareal

Máster en Ciencia de Datos e Ingeniero de Sistemas de la Universidad Icesi. Ha participado como coautor en dos publicaciones académicas del Grupo de Investigación en Informática y Telecomunicaciones (i2t) de la misma universidad. Es desarrollador certificado de Amazon Web Service (AWS) y actualmente trabaja como científico de datos en Globant (https://www.globant.com/). jsdelvil@outlook.com

David Alejandro Huertas Trujillo

Ingeniero de Sistemas de la Universidad Icesi, se desempeña como analista de datos y desarrollador backend con enfoque en la inteligencia de negocios. Sus áreas de interés profesional son: la ciencia de datos, el desarrollo de software y la ciberseguridad. aletru321@gmail.com

Brayan José Vargas Plaza

Ingeniero de Sistemas de la Universidad Icesi con enfoque en el desarrollo web fullstack. Sus áreas de interés profesional son: desarrollo de software, DevOps, ciencia de datos y ciberseguridad. branvp@protonmail.com

Bayron Daymiro Campaz Hurtado

Ingeniero de Sistemas de la Universidad Icesi y desarrollador Fullstack. Sus áreas de interés profesional son: el desarrollo de software, la ciencia de datos y el machine learning. bayroncampaz1@gmail.com

Juan David Díaz Monsalve

Ingeniero de Sistemas de la Universidad Icesi y desarrollador Fullstack. Sus intereses profesionales están centrados en aspectos relaciones con el desarrollo de software, jddiaz82@gmail.com

Santiago Gutiérrez Bolaños

Ingeniero de Sistemas de la Universidad Icesi y desarrollador Fullstack. Sus áreas de interés profesional son: la ciencia de datos, el machine learning y el diseño y desarrollo de software. santiago_bolaos96@hotmail.com

Cristhian Eduardo Castillo Meneses

Ingeniero de Sistemas y estudiante de la Maestría en Ingeniería con énfasis en Ciencia de Datos (modalidad de investigación) de la Universidad Icesi. Cuenta con experiencia en el liderazgo de equipos de desarrollo de software; sus principales áreas de interés profesional son: la ciberseguridad, la ciencia de datos, el machine learning y las metodologías ágiles de desarrollo. cristhianeduardo03@hotmail.com

Kevin Gianmarco Zarama Luna

Ingeniero de Sistemas y estudiante de la Maestría en Ingeniería con énfasis en Ciencia de Datos (modalidad de investigación) de la Universidad Icesi. Sus principales áreas de interés profesional son el machine learning y la ciencia de datos. zaramaluna1999@hotmail.com

Javier Díaz Cely

Ph.D. en Informática de la Sorbonne Université (París); Máster en Finanzas Corporativas; Máster en Inteligencia Artificial, Reconocimiento de Patrones y Aplicaciones; e Ingeniero de Sistemas. Cuenta con más de quince años de experiencia en la aplicación de inteligencia artificial y machine learning a problemas del mundo real en sectores diversos. Fue Director de la Maestría en Ciencia de Datos de la Universidad Icesi y es parte del equipo de docentes e investigadores de su Facultad de Ingeniería de dicha universidad. jgdiaz@icesi.edu.co

Este libro se terminó de editar en junio de 2022. 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.

Este libro presenta la importancia de la información y la manera de utilizarla para el proceso de ciencia de datos, para luego introducir los fundamentos teóricos de ambas disciplinas y presentar su aplicación en ocho proyectos de investigación, los cuales buscan orientar al lector sobre cómo utilizar la inteligencia artificial tanto desde la perspectiva defensiva como en la ofensiva. En el aspecto defensivo, en el libro se aborda la realización de experimentos precisos para el desarrollo de modelos para la detección de *malware* en dispositivos Android, *cryptojacking*, *deepfakes* y *botnets* maliciosas; y en el enfoque ofensivo, se exploran modelos para la generación de contenido multimedia no legítimo y se introduce el concepto de aprendizaje seguro y la aplicación de *adversarial machine learning* como enfoque para encontrar los valores que permiten sesgar los resultados de un modelo de aprendizaje de máquina.

ISBN: 978-628-7538-78-8