

MAGISTER DE INGENIERÍA EN

**SEGURIDAD DE LA INFORMACIÓN**

Método para la detección de malware en aplicaciones Android basado en aprendizaje automático y análisis estático

PROYECTO DE TESIS

Henoch Jelvez Aguila

**Magíster en Seguridad de la Información**

Santiago de Chile, Febrero 2021

Método para la detección de malware en aplicaciones Android basado en aprendizaje automático y análisis estático

PROYECTO DE TESIS

Henoch Jelvez Aguila

Profesor Guía

Helder Castrillón Cobo

**Magíster en Seguridad de la Información**

Santiago de Chile, Mayo 2021

Agradecimientos

El estudiante agradece a todos por la oportunidad de pertenecer al Magister de Seguridad de la Información de la Universidad Mayor, en especial por el apoyo de todos los profesores que participaron en esta promoción, y los que de alguna manera entregaron los conocimientos adquiridos, a través de distintas clases ó entrega de información compartida entre compañeros de promoción.

Así mismo, se agradece al profesor guía Helder Castrillón Cobo por la dedicación, apoyo continuo al proyecto de tesis, colaborando constantemente con el desarrollo y formación del alumno.

Al finalizar, se agradece a todos quienes participaron en las distintas clases incluyendo a la familia del alumno.

ÍNDICE

Contenido

[INTRODUCCIÓN 2](#_Toc76590297)

[CAPÍTULO I. EL PROBLEMA DE INVESTIGACIÓN 5](#_Toc76590298)

[1.1. Malware para Android 5](#_Toc76590299)

[1.2. Antecedentes para comprensión global de la investigación 6](#_Toc76590300)

[1.3. Problemática 9](#_Toc76590301)

[1.3.1 Pregunta de investigación 11](#_Toc76590302)

[1.4.1 Objetivo general 11](#_Toc76590303)

[1.4.2 Objetivos específicos 11](#_Toc76590304)

[1.4.3 Propósito 11](#_Toc76590305)

[CAPÍTULO II. MARCO TEÓRICO 13](#_Toc76590306)

[2.1. Aprendizaje Automático 13](#_Toc76590307)

[2.1.1 Aprendizaje Supervisado: 14](#_Toc76590308)

[2.1.2 Aprendizaje no supervisado: 14](#_Toc76590309)

[2.1.3 Aprendizaje por esfuerzo: 14](#_Toc76590310)

[2.1.4 Algoritmos 15](#_Toc76590311)

[2.2. Antecedentes generales y contexto 21](#_Toc76590312)

[2.2.1 Historia del Malware para Android 22](#_Toc76590313)

[2.2.2 Tipos de Malware 23](#_Toc76590314)

[2.3. Fórmula para evaluación algorítmica 27](#_Toc76590315)

[2.3.1 Matriz de confusión 27](#_Toc76590316)

[2.3.2 Formula de la Precisión 29](#_Toc76590317)

[CAPÍTULO III. MARCO METODOLÓGICO 31](#_Toc76590318)

[3.1. Fórmula para evaluación del algoritmo seleccionado Red Neuronal. 32](#_Toc76590319)

[CAPÍTULO IV. MAPEO SISTEMÁTICO 34](#_Toc76590320)

[4.1 Organización del mapeo sistemático 34](#_Toc76590321)

[4.1.1 Esquema general del mapeo sistemático 36](#_Toc76590322)

[4.2 Definición de protocolos de búsqueda 37](#_Toc76590323)

[4.2.1 Definición de las preguntas de investigación 38](#_Toc76590324)

[4.2.2 Definición de las palabras claves para las búsquedas de información 38](#_Toc76590325)

[4.3. Ejecución de mapeo sistemático 40](#_Toc76590326)

[4.3.1 Búsqueda de artículos 40](#_Toc76590327)

[4.3.2 Lectura de los Artículos encontrados 41](#_Toc76590328)

[4.3.3 Proceso de recolección de datos: 41](#_Toc76590329)

[4.3.4 Selección de artículos relacionados a cada pregunta de investigación. 43](#_Toc76590330)

[4.5. Selección de metadatos de detección Malware 62](#_Toc76590331)

[CAPÍTULO V. DESARROLLO 68](#_Toc76590332)

[5.1. Laboratorio selección de aplicaciones 68](#_Toc76590333)

[5.1.1 Creación de malware en laboratorio 70](#_Toc76590334)

[5.2. Laboratorio de extracción de metadatos en aplicaciones Android 79](#_Toc76590335)

[5.2.1 Extracción de fuentes aplicaciones Android malignas y benignas 79](#_Toc76590336)

[5.2.2 Clasificación de aplicaciones 79](#_Toc76590337)

[5.2.3 Selección de metadatos 82](#_Toc76590338)

[5.2.4 Sistema Scripting Pyhton para extracción de metadatos 85](#_Toc76590339)

[5.2.5 Selección del algoritmo de aprendizaje automático para detección de malware 87](#_Toc76590340)

[5.2.6 Resultados de las pruebas con las predicciones 88](#_Toc76590341)

[5.2.7 Resultados previos de algoritmos para pruebas iníciales 89](#_Toc76590342)

[5.2.8 Selección de algoritmo con mayor precisión 92](#_Toc76590343)

[5.3. Construcción de método para detección de Malware 94](#_Toc76590344)

[5.4. Evaluación del algoritmo 101](#_Toc76590345)

[5.4.1 Ítems 148 aplicaciones dataset virusshare 102](#_Toc76590346)

[5.4.2 Ítems 1000 aplicaciones dataset virusshare 104](#_Toc76590347)

[5.4.3 Item 148 aplicaciones dataset de fuente University New Brunswick(UNB) 106](#_Toc76590348)

[5.4.4 Item de 600 aplicaciones dataset UNB 108](#_Toc76590349)

[5.4.5 Item de 764 aplicaciones Dataset UNB 109](#_Toc76590350)

[5.4.6 Tabla comparativa de resultados 112](#_Toc76590351)

[CAPÍTULO VI CONCLUSIONES 114](#_Toc76590352)

[Trabajos futuros: 116](#_Toc76590353)

[Bibliografía 120](#_Toc76590354)

Resumen

Este proyecto se enfocó en la detección de aplicaciones malware basado en metadatos de las aplicaciones Android, como por ejemplo permisos obtenidos, pudiendo extraer algunos metadatos relevantes del archivo Manifiesto, además se desarrollaron varios conjuntos de datos con aplicaciones malignas y benignas para poder ser analizadas con algún algoritmo de aprendizaje automático. Para lograr los objetivos de la tesis se testeo varios algoritmos hasta llegar a obtener el más preciso según la cantidad y tipo de aplicaciones Android, siendo el más adecuado el denominado Redes Neuronales de aprendizaje automático, para extraer los metadatos seleccionados se desarrollo y utilizó un sistema script Python con librerías de extracción de metadatos del archivo manifiesto de aplicaciones Android con extensión APK, las metadatos extraídos fueron SDK, receiver, MainActivity, y permisos, para lograr todo este cometido se realizó una investigación metodológica mapeando la información de diferentes documentos e investigaciones que lograron aportar a la tesis, extrayendo los datos más importantes basados en los objetivos de la investigación y las preguntas que se desarrollaron, obteniendo de estos documentos algunos datos importantes de detección *malware, algoritmos utilizados para detección malware y vulnerabilidades Android*. En esta tesis se enfocó en lo investigativo y aplicativo desarrollando de laboratorios, conjunto de aplicaciones, algoritmo de redes neuronales, todo metodológicamente ordenando cómo indica esté documento, con el fin de demostrar cómo un algoritmo de aprendizaje automático puede detectar eficientemente un malware en aplicaciones Android.

INTRODUCCIÓN

El aprendizaje automático (Digital Guide IONOS , 2019) apareció para permitir a las computadoras decidir e imitar la mente humana, este ha sido uno de los desarrollos más significativos y notables en el campo de la informática, para lograrlo se desarrollaron múltiples algoritmos a lo largo del tiempo, logrando que las máquinas realicen tareas y procesos cotidianos del ser humano, el aprendizaje automático es un subcampo de las ciencias de la computación y una rama de la inteligencia artificial, cuya finalidad es desarrollar técnicas que permitan a las computadoras aprender de forma automática basándose en información, igual cómo lo realiza el cerebro humano, esto ha permitido contribuir en muchas áreas cómo medicina, construcción, finanzas, robótica, educación, informática, incluso en la carrera espacial, entre muchas más, esta ciencia está enfocada en la mejora el análisis de datos, llegando inclusive a ser capaz de predecir el futuro, en implementación de nuevos sistemas ó mejorando los actuales, a través de algoritmos, para profundizar más del concepto de inteligencia artificial y aprendizaje automático ha aportado en la seguridad informática en sistema de detección de ataques, Firewall inteligentes, modelado de amenazas automático, antivirus, correlacionador de eventos, entre otros aportes que se realizan a diario. Este campo data del año 1943, por el matemático Walter Pitts y el neurofisiólogo Warren McCulloch, los que dieron a conocer un trabajo enfocado a lo que conocemos cómo Inteligencia Artificial (IA), su teoría era analizar el cerebro cómo un organismo computacional y la creación de computadoras que funcionan iguales ó mejor que la red neuronal humana. Desde esos años su aporte ha sido un hito sin olvido para la humanidad, incluidos los trabajos de Alan Mathison Turing (Hinestroza Ramírez, 2018). En los campos que ha sido un aporte la Inteligencia Artificial es la informática y con relación específicamente en temas de seguridad relacionados al sistema operativo Android encontramos sistemas antivirus que algunos no se basan en aprendizaje avanzado y por esto poseen muchos problemas en la detección de malware, debido a la cantidad de dispositivos Android utilizados en el mundo que según un estudio de la fuente NTS Solutions (NTS , 2020) se aproximaron en el año 2019 una cantidad de 2.500 millones, por este principal hecho se necesita una evolución constante en materia de seguridad informática, necesitando de tecnología avanzada para poder ser capaces de detectar, notificar, bloquear, predecir los ataques malware, ya que se transmiten por distintos protocolos y técnicas en la red, estos son ejecutados por ejemplo con ingeniería social. Con el paso del tiempo y con apoyo de las herramientas antivirus, igual en paralelo con la evolución de los sistemas operativos Android, aparecen nuevos malwares desarrollados para distintos ataques, infectando a diferentes tipos de aplicaciones, las que podemos nombrar algunas por categorías teniendo las bancarias, Criptomonedas, redes sociales, entre otras, por esto mismo se hace necesario que se desarrollen nuevas técnicas y métodos para la pronta detección, alerta, notificación y bloqueo de estas aplicaciones malignas, las que pretenden realizar cualquier tipo de acción no autorizada, cada día se desarrollan nuevas investigaciones para la detección de malware basado en aprendizaje automático, el cual es un área de la Inteligencia Artificial, estas técnicas trabajan en paralelo con los sistemas antivirus, pero el problema radica en que la evolución constante y el uso masivo de Android, requiriendo un esfuerzo extra en la innovación de métodos y técnicas de detección, debido a estos las universidades, institutos, entidades gubernamentales, entre otros, constantemente aparecen publicaciones en las bibliotecas cómo Springer Link, entre otra que podemos buscar por google Scholar. Esto se aprovecha en las Universidades para motivar a los estudiantes en el estudio de ataques a dispositivos móviles, y en este trabajo se desarrolla con un método de investigación, uso de herramientas, conclusiones de cada teoría, el cual se espera aportar en palabras simples al menos una idea de cómo poder investigar métodos de detección malware Android, enfocado en los conceptos simples explicados en esta introducción, ya sea que ha futuro pueda utilizarse esta técnica, método, sistema, en algún desarrollo.

CAPÍTULO I. EL PROBLEMA DE INVESTIGACIÓN

1.1. Malware para Android

Tanto cómo Google y Apple han tomado las medidas de prevención para evitar debilidades en sus plataformas, nombraremos algunas cómo por ejemplo corregir vulnerabilidades de sus sistemas operativos antes de su puesta en marcha, implementar medidas de seguridad en las tiendas oficiales de aplicaciones, con el fin de evitar el malware en los dispositivos móviles se propague de forma masiva y bloquear cualquier tipo de ataque. Pero el impacto del malware en estos dispositivos móviles es cada vez más alto debido que según estudios (Genbeta , 2012) el año 2012 un 99% del malware diseñado para Smartphone detectado por ***Kaspersky Lab*** fue diseñado para atacar dispositivos móviles que tenían como software base el sistema operativo Android. En el estudio indicaron que el sistema operativo Android es el más propagado en nuevos Smartphones, y según una empresa estadounidense dedicada a las investigaciones de mercado, análisis y consultorías IDC, en el tercer trimestre del año 2016, Android lidera el mercado de teléfonos inteligentes con una cuota del 86.6%, según la tabla de ejemplo adjunta en este capítulo (Huilcamaigua Pazuña, 2017):

**Tabla 1, Cuota de mercado de sistemas operativos para Smarphones**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Periodo** | **Android** | **iOS** | **Windows Phone** | **Otros** |
| 2015Q4 | 79.6% | 18.7% | 1.2% | 0.5% |
| 2016Q1 | 83.5% | 15.4% | 0.8% | 0.4% |
| 2016Q2 | 87.6% | 11.7% | 0.4% | 0.3% |
| 2016Q3 | 86.8% | 12.5% | 0.3% | 0.4% |

1.2. Antecedentes para comprensión global de la investigación

Los dispositivos móviles han adquirido una popularidad muy elevada en los últimos años, motivo principal por la gran cantidad de funcionalidades que tienen y según estudios comerciales recientes los que demuestran cómo a partir del 2011 en el mundo se venden más teléfonos inteligentes que computadoras del tipo escritorio (Carton, 2018), esto significa que el número de usuarios de teléfonos inteligentes ha aumentado en la misma proporción. En el año 2019 se reportó que el número de usuarios en teléfonos inteligentes fue de 3200 millones y en el 2020 se predecía que este número sería de 3.500 millones (O'Dea, 2021), lo anterior se genera cómo consecuencia que los teléfonos inteligentes sean el objetivo principal de los ataques informáticos cómo programas maliciosos, hackers, botnet, etc., con una gran diversidad de propósitos, como por ejemplo robar información, espionaje, daño, estafas, entre otros. Las amenazas y objetivos principales se pueden apreciar de manera general y estos se consuman desde un fraude hasta el uso indebido del servicio, principalmente impulsado por incentivos económicos, ejecutando ataques informáticos cómo ejemplo: spam, espionaje, denegación de servicio (DoS, por sus siglas en inglés), robo y sabotaje de datos, entre muchos más (G. Suarez-Tangil, 2014). Se debe diferenciar las amenazas de las vulnerabilidades, por ejemplo una amenaza puede ser modificación de información afectando la integridad de los datos, esto podría ser explotado desde una vulnerabilidad cómo por ejemplo inyección de datos, los ataques no solo son aplicados por entes externos a las organizaciones, sino en muchas ocasiones por agentes internos, esto podría ser realizado intencional y no intencional, en la intencionalidad un usuario podría dejar expuesta una vulnerabilidad en alguna aplicación para que un agente externo la explote ó este mismo usuario podría robar información confidencial de la organización para venderla en la Deep web, y en lo no intencional este mismo usuario podría por error humano borrar datos confidenciales afectando la disponibilidad y provocando incluso una denegación de servicios sin ninguna intención de hacerlo.

El malware dirigido a teléfonos inteligentes en particular a los que utilizan el sistema operativo Android, es cada día más exponencial debido a que este sistema operativo Android domina el mercado mundial en un 87% (O'Dea., 2021). Android ha tenido un vertiginoso desarrollo y que en conjunto con su popularidad es un sistema operativo abierto (open source), estos factores convierten a este sistema operativo en el blanco de muchas familias de malware, y las últimas tendencias de estas familias incluyen: Troyanos bancarios, minado de criptomoneda, secuestro de dispositivos (ransomware, en inglés), malware híbrido (por ejemplo, Android/LokiBot) (Kouliaridis, 2020), con múltiples objetivos y en particular el ransomware móvil tiene como fin evitar que los usuarios accedan a los datos en sus dispositivos mediante el cifrado de los mismos, hasta que una cantidad considerable de dinero sea pagado cómo rescate con el riesgo que esas claves nunca se entreguen y que si funcionaran esas contraseñas que se les será entregada por estos usuarios maliciosos, en datos estadísticos para el primer semestre del 2017 el ransomware móvil tuvo un aumento de un 60% (Kouliaridis, 2020), y en la actualidad muchas familias de ransomware móvil están presentes en muchas aplicaciones, este malware representa una amenaza alarmante para los usuarios que deben elegir pagar el rescate ó terminar con la pérdida de datos valiosos, al estar cifrados y no tener la llave de descifrado.

El análisis y detección de malware en Android puede utilizarse tres tipos cómo lo son, estático, dinámico e híbrido (Morales-Ortega S. E.-A.-M.-A., 2015, 2016), para resumir en el análisis estático la aplicación no se instala en el dispositivo, sino que se extraen algunos metadatos de forma estático, por ejemplo, la lista de permisos que solicita la aplicación, el análisis dinámico involucra que la aplicación se instale y pueda correr en un dispositivo real ó virtual, se extraen metadatos de forma dinámica, por ejemplo, el número de llamadas al sistema, y en el análisis híbrido implica la realización de una combinación de análisis estático y dinámico.

En este proyecto se propone diseñar un método y/o técnica, que logre realizar una detección de malware en aplicaciones Android con el fin de poder detectar los metadatos más predispuestas a ser utilizadas para desarrollar malware Android (Cómo por ejemplo Permisos, Tamaño, Rutas, vulnerabilidades, Funcionalidades, etc.), posteriormente al tener estos metadatos se podrá comparar estos mismos con las familias de malware que se asemejen más a las aplicaciones analizadas, pudiendo finalmente poder predecir qué tan susceptible es la aplicación analizada en ser utilizada por usuarios malintencionados para explotar desde estos metadatos, cómo por ejemplo con inyección de código malicioso y/o cualquier otra técnica mal intencionada, explotándose finalmente una amenaza que pudiera evitarse con el análisis por nombrar algunas cómo (Robo de información, secuestro de información, ataque de botnet, Phishing, espionaje, etc.,), incluyendo las amenazas de día cero, afectando la integridad, disponibilidad, trazabilidad y confidencialidad de los datos.

1.3. Problemática

La cantidad de dispositivos que utilizan el sistema operativo Android generan una difusión y descarga constante de una gran cantidad de aplicaciones maliciosas de sitios, bibliotecas, orígenes desconocidos, las que pueden saltar todos los controles de seguridad llegando al dispositivo final móvil víctima, para que los usuarios malintencionados puedan realizar algunos tipos de delitos, incluyendo espionaje, estafas bancarias, robo y secuestro de información, a través de distintas aplicaciones maliciosas tales podemos nombra algunas categorías cómo Botnets, Rootkits, Troyanos, Virus, Gusanos, Ransomware, y otros tipo de ataques avanzados que deben ser descubiertos antes que se propaguen por muchos medios a otros dispositivos en la red, aquí tenemos un problema en la evolución constante de los ataques a los dispositivos Android donde se necesita una investigación continua para detectar, bloquear, y notificar a las bases de datos de malware las aplicaciones maliciosas nuevas detectadas en la red, si estas investigaciones no avanzan seria un problema aun mayor el utilizar los dispositivos inteligentes Android, incluyendo las Tablet’s, Computadoras, Teléfonos inteligentes, televisores inteligentes, y incluso podríamos incluir Android en los vehículos autónomos.

Al revisar distintas investigaciones de detección de malware Android, no se encuentran investigaciones que se enfoquen en predecir qué tan susceptible es una aplicación a ser utilizada por usuarios y/o organizaciones malintencionadas según los metadatos seleccionados de las aplicaciones, incluyendo el máximo de metadatos de malware en esos estudios, para ser utilizados en algunos ataques de inyección de código malicioso y/o cualquier otra técnica cómo (Robo de información, secuestro, Botnet, espionaje, etc.).

**Un ejemplo sencillo:** existen aplicaciones que por su tamaño son más susceptibles a ser utilizadas para inyección de código malicioso y ejecución remota en dispositivos utilizados por alguna víctima, esta característica se podría detectar en ambientes de desarrollo con aplicaciones desarrolladas en aprendizaje automático antes de ser expuesta en producción, inicialmente igual se podría colocar un control de estos metadatos de malware más vulnerable en el diseño, arquitectura, y desarrollo de aplicaciones Android, creando controles de seguridad antes de ser testeadas con alguna prueba de penetración ó un circuito de desarrollo seguro, teniendo esto cómo ventaja de mitigación de todas las amenazas que podrían tener estas aplicaciones de cualquier organización, no solo dejando esto en el desarrollo, sino darle continuidad posteriormente en ambientes productivos.

Para detectar estos metadatos y desarrollar este trabajo de tesis, se debió catalogar distintas muestras de aplicaciones malignas y benignas mediante aprendizaje máquina, seleccionando el método y algoritmo más efectivo para la extracción de estos metadatos de malware de las aplicaciones, posteriormente de catalogar alguna base de datos de los metadatos de malware, las aplicaciones benignas y/o malignas en un trabajo extenso se debió buscar las muestras de aplicaciones, encontrándose con algunos problemas cómo base de datos aplicaciones Android no actualizadas, por lo que fácilmente se podría pasar por alto algunos metadatos vulnerables de día cero (nuevas amenazas y vulnerabilidades), haciendo necesario encontrar fuentes confiables de nuevos ataques malwares, y actualizar constantemente el algoritmo de detección.

1.3.1 Pregunta de investigación

¿Cómo mejorar la detección de malware en aplicaciones Android utilizando técnicas de análisis estático y aprendizaje automático?

**1.4. Objetivos**

1.4.1 Objetivo general

Diseñar un método de detección de malware en aplicaciones Android basado en aprendizaje automático y análisis estático.

1.4.2 Objetivos específicos

* Identificar los metadatos de malware utilizados para detectar malware usando análisis estático y aprendizaje automático.
* Definir un método de aprendizaje automático para la integración en un modelo de detección de malware con base en los metadatos para la identificación del Malware.
* Validar la eficiencia del método diseñado utilizando un conjunto de datos de malware Android y los resultados de la medición con métricas de aprendizaje automático.

1.4.3 Propósito

Diseñar un método para la detección de malware en aplicaciones Android basado en análisis estático, extrayendo algunos metadatos utilizados por malware y comportamientos propios del malware, el que pueda evaluar estos metadatos propios de aplicaciones Android y se pueda hacer comparación con las familias de malware usando un algoritmo de aprendizaje automático.

CAPÍTULO II. MARCO TEÓRICO

Para el desarrollo de los objetivos de este proyecto se necesita contar con los conceptos de aprendizaje automático aplicados a la seguridad de dispositivos móviles, específicamente relacionado al sistema operativo Android, es por ello que en esta sección se hace una descripción general sobre el aprendizaje automático y sus aplicaciones en la seguridad de la información, dentro de esto se explica lo relacionado con malware para dispositivos móviles. Adicionalmente, se aborda el sistema operativo Android, se enuncian sus mecanismos de seguridad, vulnerabilidades y algunas herramientas para pruebas de penetración para la evaluación de aplicaciones Android, y finalmente se demuestra la importancia en el desarrollo de sistemas con algoritmos de aprendizaje automático.

2.1. Aprendizaje Automático

Aprendizaje automático (Gonzalez, 2018) es la ciencia de la inteligencia artificial que puede hacer que un sistema tenga la capacidad de aprender en entornos variables, sin ser programado de forma explícita. Su usabilidad ha venido en auge debido a la gran cantidad de datos que almacenan los medios electrónicos, teniendo un problema en ese sentido cómo por ejemplo (el problema del big data) y dependiendo del problema se puede utilizar tres técnicas: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo, los que resumiremos en los siguientes puntos.

2.1.1 Aprendizaje Supervisado:

Esta técnica en sencillas palabras se tiene conocimiento sobre el conjunto de datos, principalmente una comprensión sobre la salida correcta de los datos en el algoritmo y su relación con la entrada (Gonzalez, 2018). Los tipos de problemas que se abarcan en este método son de regresión y clasificación.

2.1.2 Aprendizaje no supervisado:

En esta técnica no se tiene conocimiento alguno sobre el conjunto de los datos, por lo que el aprendizaje no supervisado tiene como objetivo encontrar la estructura y los patrones que se presentan en la información (Gonzalez, 2018).

2.1.3 Aprendizaje por esfuerzo:

En esta técnica se busca que el algoritmo esté con capacidad de encontrar el conjunto de operaciones más oportunas para cumplir un objetivo a través del aprendizaje de reglas y acciones. Para problemas de clasificación existen cuatro tipos de medidas para evaluar los resultados del algoritmo: False Positive Rate (FPR), False Negative Rate (FNR), True Positive Rate (TPR) y True Negative Rate (TNR). Adicionalmente, también se encuentran otras medidas que ayudan evaluar el desempeño de los algoritmos y diagramas cómo el Receiver Operating Characteristics (ROC) el que permite comparar cada medida encontrada y su área bajo la curva (AUC), y este sirve para conocer el desempeño del clasificador (Alpaydin, 2014).

**Tabla 2, Confusión Matrix for two clases**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Predicted class** | | |
| True Class | Positive | Negative | Total |
| Positive | tp: true positive | fn: false negative | P |
| Negative | fp: false positive | tn: true negative | N |
| Total | p ́ | n ́ | N |

**Tabla 3, Performance measures used in two-class problems**

|  |  |
| --- | --- |
| **Nombre** | **Formula** |
| Error  Precisión | (fp + fn) / N  (tp + tn) / N = 1 – error |
| tp-rate  fp-rate | tp / p  fp / n |
| precisión  recall | tp / p′  tp / p = tp rate |
| Sensitivity  Specificity | tp / p = tp rate  tn / n = 1 − fp rate |

2.1.4 Algoritmos

Un algoritmo (Santos, 2017) es una serie de pasos ordenados que se ejecutan para realizar una tarea determinada, en el aprendizaje automático el objetivo del algoritmo es crear un modelo que permita resolver una tarea, proceso, acción, entre otros objetivos, este tipo de algoritmos de aprendizaje automático entrena un modelo utilizando gran cantidad de datos estructurados, para finalmente ser capaz de hacer predicciones, entre otras acciones avanzadas, dependiendo de la cantidad y calidad de datos que se utilizan para estrenar el diseño del algoritmo.

**Tipos de algoritmos:**

En este item explicaremos algunas clases de algoritmos y su descripción básica, calcificándolos en tres grupos (APD, 2019), los que podemos destacar en aprendizaje supervisado, no supervisado y por refuerzo, daremos una descripción básica de cada uno de estos grupos.

**Aprendizaje supervisado:** Esta clase de algoritmos es útil cuando se tiene una propiedad conocida para un conjunto de elementos, pero no se conoce esa misma en un elemento concreto. Estos algoritmos pueden predecir cuál es esa propiedad que se desconoce.

**Aprendizaje no supervisado:** Esta clase de algoritmos es útil para descubrir relaciones implícitas en un conjunto de datos que no son conocidas, es decir se permite considerar que varios elementos pertenecen a un mismo grupo y diferentes grupos, gracias al estudio de sus características.

**Aprendizaje por refuerzo:** Esta clase de algoritmos es la unión de los dos grupos anteriores, este aprendizaje por refuerzo se centra en los procesos de aprendizajes reglamentados, en los que se proporcionan algoritmos de aprendizaje automáticos con un conjunto de acciones, parámetros y valores finales.

**Algunos algoritmos de ejemplo:**

En este item se describirán algunos algoritmos de aprendizaje máquina, los son utilizados en algunos desarrollo de soluciones aprendizaje automático, esto se realiza para poder entender las reglas básicas, funciones, ventajas, desventajas, en qué tipo de datos se pueden utilizar, formulas matemáticas básicas, los que tienen una breve descripción y se nombran cómo Naive Bayes y K Means (IA, 2021).

* **Naive Bayes**

Naive Bayes (IA, 2021) es un algoritmo simple y poderoso para la clasificación basada en el teorema de Bayes, este es fácil de construir, muy útil de utilizar en los conjuntos de datos muy grandes.

**Ventajas:**

Este algoritmo es rápido y fácil para predecir la clase de conjunto de datos, y funciona muy bien en predicción multiclase.

Cuando se mantiene la suposición de independencia, un clasificador Naive Bayes funciona mejor en comparación con otros modelos cómo los de Regresión Logística, y se necesitan menos datos de entrenamiento. Tiene un funcionamiento correcto comparado con variables numéricas.

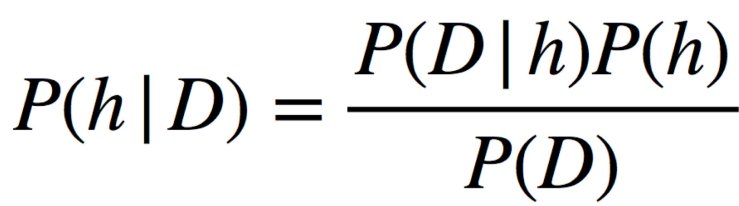
**Desventajas:**

Una de las desventajas es, si una variable categórica tiene una categoría en el conjunto de datos que se utilizan de prueba y que no se logró observar en el conjunto datos de entrenamiento, el modelo asignará una probabilidad de 0 y no dejará hacer una predicción, esto se conoce cómo una frecuencia 0. Este problema se resuelve utilizando una técnica que es denominada alisamiento.

También tiene una limitación en asunción de predictores independientes en casos reales, que es casi imposible obtener un conjunto de predictores que sean completamente independientes.

**Formula:**

En la imagen de la fórmula del algoritmo Naive Bayes, se presenta cada enunciado que se realiza para llegar a la conclusión del resultado, y estos se explican más debajo de este punto.



**Ilustración 1, formula Algoritmo Naive Bayes obtenido de Aprende IA** (IA, 2021)

**Explicación de los enunciados del algoritmo**:

**P (h):** Esta es la probabilidad de que la hipótesis h sea cierta (Cualquiera sean los datos), se conoce cómo la probabilidad previa de h.

**P (D):** Esta es la probabilidad de los datos (Cualquiera que sea la hipótesis), le podemos decir cómo probabilidad previa.

**P (hada):** Esta es la probabilidad de la hipótesis h dada los datos D, esto lo podemos conocer cómo la probabilidad posterior.

**P (Doha):** Esta es la probabilidad de los datos d, dado que la hipótesis h era cierta, esto se conoce cómo la probabilidad posterior.

* **K Means** (Aprende IA, 2021)

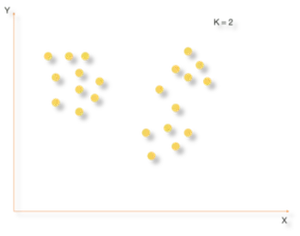
**Tipo de algoritmo:** Aprendizaje no supervisado

**Utilización**: Se utiliza cuando no se tiene datos etiquetados, es decir son datos sin categorías ó grupos definidos.

**Objetivo del algoritmo:** Encontrar grupos en los datos, y los puntos de datos se agrupan según la similitud de metadatos.

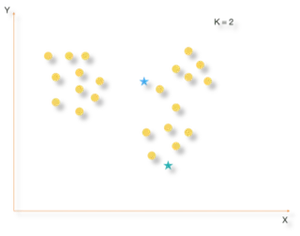
**Pasos para desarrollar este algoritmo:**

* Agrupa en K grupos, donde K está definida, cómo lo demuestra la imagen que prosigue.



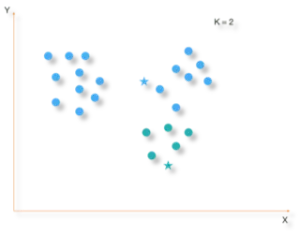
**Ilustración 2, Algoritmo K Means K grupos**

* Selecciona K puntos al azar cómo centros de grupo, cómo lo presenta la imagen que prosigue a este punto.



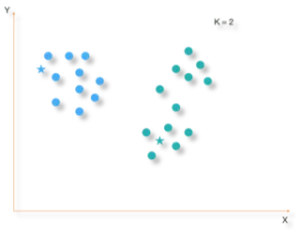
**Ilustración 3, Algoritmo K Means K puntos**

* Asigna objetos a su centro de clúster más cercano según la distancia.
* Calcule el centroide ó la medida de todos los objetivos en cada grupo.



**Ilustración 4, Algoritmo K Means Centroide**

* Repite los pasos anteriores, hasta asignar los mismos puntos a cada grupo en rondas. “Este procedimiento se debe repetir tantas veces hasta que se encuentren el agrupamiento de datos más óptimos. En ocasiones se deberá cambiar el valor de K para mejorar los resultados”.

****

**Ilustración 5, Algoritmo K Means repetición pasos 2 y 4**

**Obligaciones para en el uso del algoritmo:** Se debe especificar el número de clústeres de antemano, y los resultados finales son sensibles a la inicialización.

2.2. Antecedentes generales y contexto

**Malware**

La palabra malware es una abreviación de “Maliciosus Software” (Software malicioso) el que se refiere a todo tipo de programa informático creado para realizar acciones no deseadas, y que son perjudiciales para el usuario de una computadora ó dispositivo inteligente. Este programa informático se presenta de distintas maneras, y se ha podido detectar distintas versiones a través del tiempo.

2.2.1 Historia del Malware para Android

En orden cronológico se ordenan los sucesos y nombres de aplicaciones maliciosas para Android detectadas a través de los años(Bustos, 2018)**:**

* **Fake Player**: Con fecha Agosto del 2010, Denis Maslennikov de la empresa Kaspersky descubrió el primer troyano SMS, este aparece en una aplicación multimedia.
* **GPS Spy:** Con fecha Agosto del 2010, la empresa Symantec descubrió el primer GPS Software malicioso, este programa estaba disfrazado cómo un juego serpiente capaz de enviar las coordenadas GPS a un servidor remoto.
* **Geinimi:** Finales del año 2010, este primer software que presenta cómo Botnet.
* **PjApps: C**ontiene troyano mostrado cómo un bot tradicional, permite que se abra una puerta trasera (Backdoor) y recibe comandos de un servidor remoto.
* **Droid Dream:** Aparece año 2011, infecto a más de 50 aplicaciones en Android, con este malware aparece la nueva era softwares maliciosos, y este se uso para el robo de datos, exploits de root y funciones de bootnet.
* **ZitMo:** Aparece a fines de 2011, este utiliza SMS para dirigir y controlar.
* **FakeInst:** Aparece año 2012, es otro troyano SMS que se disfrazo de aplicaciones populares.
* **SMSZombie**: Aparece año 2012, en mercados de terceros en China y explota el sistema de pago en línea de China Mobile.
* **NotCompatible:** Aparece año 2012, fue descubierto por Lookout Mobile Security, es el primer malware en usar sitios Web cómo medio de propagación, y infecta el teléfono inteligente cuando visita un sitio web específico.
* **Android.Bmaster:** Aparece año 2012, en un red de bot móviles, este género un ingreso de 10.000 y 30.000 dólares.
* **LuckyCat:** Aparece año 2012, este estaba dirigido a industrias aeroespaciales y de energía en Japón, crea una puerta trasera (Backdoor) que se abre en el dispositivo infectado.
* **FakeDefender:** Aparece año 2013, y es el primer ransomware para Android, este programa malicioso se descarga cómo si fuera un antivirus, pero su fin era bloquear el dispositivo y después pedir recate para el desbloqueo.

2.2.2 Tipos de Malware

Estos programas informáticos maliciosos denominados malware se les conoce distintas versiones y nombres, los cuales nombraremos:

* **Virus:** Este tipo de programa tienen un código que se replica para sí mismo y se puede propagar por medio del software, esta aplicación maliciosa es desarrollada para infectar los sistemas operativos, se puede propagar cuando el usuario ejecuta ó acepta de forma poco visible la instalación de software malicioso, este puede llegar a los dispositivos por medio de correos electrónicos, enlaces de Internet, compartiendo archivos vía P2P, memorias USB, discos duros, entre otros medios de propagación. Este software empieza a funcionar cuando se ejecuta un programa infectado permitiendo que el código quede alojado en memoria RAM del dispositivo. Posteriormente este programa puede infectar archivos ejecutables que sean llamados por este software, añadiendo el código malicioso a estos archivos y grabándose en el disco (Pazuñ, 2017).
* **Troyano:** Este tipo de software se presenta como un programa legítimo el que se ejecuta y cumple las funciones maliciosas para las que fue desarrollado, algunas funciones son controlar el dispositivo sin ser advertido, estos programas se difunden solos y generan un gran número de infecciones usando herramientas de Internet. Los troyanos se clasifican dependiendo de la acción que realiza en el dispositivo. Para cual tiene distintas clasificaciones:
  + **Blackdoor:** Este tipo de troyano otorga al atacante acceso control remoto sobre el dispositivo infectado, este permite realizar cualquier acción, estas acciones pueden ser envío, ejecución y eliminación de archivos, igual la visualización de datos y hasta el reinicio del equipo. Este software se usa usualmente para unir un grupo de computadores u otros dispositivos infectados con la finalidad de usarlos cómo botnet con propósitos malignos.
  + **Exploit**: Estos softwares contienen información ó código que pueden tener ventaja de vulnerabilidades del software que se está ejecutando en un equipo.
  + **Rootkit:** Este programa está diseñado para ocultar objetos y actividades en el sistema, y su objetivo es evitar ser detectado, para incrementar el tiempo de dichos programas que se ejecutan en un dispositivo infectado.
  + **Trojan-Banker:** Este software malicioso es diseñado para robar información de cuentas bancarias, sistemas de pago en línea, tarjetas de crédito y débito.
  + **Trojan-DDoS:** Su función es para ataques de denegación de servicios distribuido, y se debe tener en consideración que además de la víctima del ataque, se usan más dispositivos infectados.
  + **Trojan-Downloader:** Este software puede descargar e instalar nuevas versiones de malware en el dispositivo.
  + **Trojan-Dopper:** Este software es usado para instalar troyanos ó virus, para prevenir la detección de malware en las herramientas Antivirus y que no son capaces de detectar todos los componentes de este tipo de troyano.
  + **Trojan-Ransom:** Este tipo de troyano puede modificar información y archivos del equipo infectado, para que no pueda ser utilizada esa información específica del usuario, incluso puede bloquear el inicio de sesión de una cuenta de usuario. Generalmente los atacantes restablecen el funcionamiento del equipo después de pagar el rescate, pero no es segura esta maniobra.
  + **Trojan-SMS:** Estos programas se utilizan para enviar mensajes de texto desde el teléfono móvil infectado hacía números de tarifa premium, la cual representa al usuario una pérdida de dinero.
  + **Trojan-Spy:** Estos programas pueden espiar, capturando información que se ingresa al teclado, tomando capturas de pantalla ó consiguiendo una lista de aplicaciones que se están ejecutando.
  + **Ransomware:** Es un software malicioso que puede bloquear el equipo desde una ubicación remota, encriptar los datos almacenados y los archivos, para desbloquear se pide generalmente una suma de dinero cómo rescate. Antiguamente el ransomware se utilizaba para los equipos de escritorio, pero ya ha llegado también a los dispositivos móviles cómo los Smartphone.
  + **Gusano:** Igualmente conocidos como Worms, estas aplicaciones que pueden auto reproducirse hasta llegar a saturar un sistema. Su fin solo es destruir el sistema no para obtener beneficios del dispositivo.

2.3. Fórmula para evaluación algorítmica

Para cada evaluación algorítmica de datos en aprendizaje automático y otras aéreas de las ciencias de la informática, cómo otros campos de la investigación, se requieren formulas para poder evaluar una acción, en algunas ocasiones entre menos datos mejor es el resultado, pero para otras se quiere más datos para llegar a una mejor precisión, para esta investigación de definieron dos conceptos de medición las que son, matriz de confusión y formula de precisión, explicándolas brevemente en este apartado investigativo con sus respectivas imágenes ilustradas (Arce, 2019).

2.3.1 Matriz de confusión

Esta es una medición del rendimiento para el problema de clasificación en aprendizaje automático, entregando una salida de cuatro tipos de datos en un conjunto de datos. Se puede apreciar esta información generalizada de la salida de datos del algoritmo en una tabla con cuatro combinaciones diferentes de valores predichos y reales, el que demuestra de manera detallada cómo funciona este módulo de muestra, para esto explicamos de manera general con el fin de entender los resultados que serán presentados posteriormente en esta investigación.

Para el trabajo de investigación se evaluaron cinco conjuntos de datos, debido que entre más datos se analizan se obtiene una mayor presión y descripción de resultados, si bien cómo se explica de manera generalizada en este capítulo se debe aclarar que entre más datos se proponen en el análisis se obtendrán resultados más optimizados, precisos y estructurados, todo esto para explicar cómo es el funcionamiento general de cada análisis, casi finalizando se obtienen las métricas respectivas para evaluar la precisión. Todos los metadatos de aplicaciones malignas (Malware) y Benignas (Sin Malware) son la mitad del conjunto de aplicaciones, siendo varios dataset originarios de un mismo conjunto de datos, y también de otros conjuntos de datos, logrando evaluarse de forma general, entendiendo que esta es la introducción generalizada de la cada funcionalidad de los algoritmos, obteniendo en modo general en la matriz de confusión, y precisión. Para poder hacer cualquier análisis de datos se requiere formulas matemáticas, y este caso se explica cada enunciado:

* Aplicaciones APK que tienen malware y el modelo los clasifica cómo malware positivo sin errores (+). Esto sería un verdadero positivo con las letras VP.
* Aplicaciones que no tienen malware y el modelo los clasifica cómo malware negativo (-). Esto sería un verdadero negativo con las letras VN.
* Aplicaciones que tienen malware y el modelo los clasifica cómo malware negativo (-). Este sería un error tipo II ó falso negativo con las letras FN.
* Aplicaciones que no tienen malware y el modelo clasifica cómo malware positivo (+). Esto sería un error tipo I, ó falso positivo con las letras FP.

Se puede claramente identificar de forma generalizada con esta matriz de ejemplo donde están los errores en rojo, y los datos seleccionados correctamente en color verde, para que se pueda entender fácilmente cómo lo muestra la siguiente imagen, con esta explicación entenderemos más fácilmente los resultados que entregaremos más adelante de la tesis.

Para mayor comprensión de lo anteriormente explicado se presenta la imagen explicativa de la matriz de confusión explicando los cuatro puntos de resultados, como verdaderos positivos, falsos positivos, falsos negativos y verdaderos negativos, los cuales en los resultados en cada celda se mostrara el resultado en números y porcentajes.



**Ilustración 6, ejemplo matriz de confusión**

2.3.2 Formula de la Precisión

La precisión ó accurancy en inglés, se refiere de forma generalizada lo cerca que está el resultado de una medición del valor verdadero. Lo que sería en términos estadísticos, la exactitud que está relacionada con el sesgo de una estimación. Igualmente se conoce cómo verdadero Positivo (True positive rate). Esto está representado por la proporción entre reales predichos por el algoritmo y todos los casos positivos, lo que se explicara con las formulas para hacernos una idea cómo se calcularan los resultados en esta investigación y en los capítulos posteriores.

Para cual se utiliza la siguiente fórmula:

**Precisión = (VP+VN) / (VP+FP+FN+VN)**

**Descripción**

**VP=** Verdadero positivo

**VN=** Verdadero Negativo

**FP=** Falso Positivo

**FN=** Falso Negativo

**2.3.3 La precisión explicada de forma técnica**

La precisión se refiere a la dispersión del conjunto de valores obtenidos a partir de las mediciones repetidas de una magnitud. En cuanto menor es la dispersión mayor la precisión. Esta se representa por la proporción entre el número de predicciones correctas (tanto positivas como negativas), y el total de las predicciones.

**La fórmula práctica es:**

**VP/ (VP+FP)**

CAPÍTULO III. MARCO METODOLÓGICO

Para el desarrollo de este proyecto se utilizaron pasos metodológicos con una investigación aplicada y descriptiva, debido a que se espera proponer una solución para la detección de malware en aplicaciones Android, explicándolo de lo general a lo particular y/o detallado. En este marco metodológico se describirá de manera generalizada la metodología que se utilizó para conllevar este proyecto tesis, destacando algunas fases. No se describen puntos en detalle, debido a que es necesario explicar este proceso de manera generalizada para posteriormente en puntos posteriores entregar detalles de cada una de las fases.

* **Fase de pre desarrollo:** se elaboró un mapeo sistemático basado en la guía descrita por Kai Peteren en (Kai Petersen, 2015), también se diseña y desarrolla un laboratorio para pruebas de metadatos utilizados por malware en aplicaciones Android.
* **Fase de desarrollo:** se elaboró con la construcción de varios conjuntos de datos y pruebas de distintos algoritmos para identificar el modelo más adecuado. Finalmente, se realizó la construcción de un modelo de aprendizaje automático capaz de predecir si una aplicación Android contiene metadatos utilizados por malware.
* **Fase de validación:** se elaboró los niveles de precisión y su comparación con distintos conjuntos de datos, y así demostrar que el modelo es el adecuado para esos datos específicos, proponiendo algunos métodos.

En este punto, se describe cada fase de forma generalizada como se demuestra en la ilustración siete, llevados a cabo para concretar cada uno de los puntos, estos pasos son generales y se entrega una breve descripción sobre mapeo sistemático, laboratorio, desarrollo, y validación.

***Ilustración 7, metodología de desarrollo del proyecto.***

3.1. Fórmula para evaluación del algoritmo seleccionado Red Neuronal.

Se realizó un análisis general del algoritmo de Red Neuronal, debido a que este fue seleccionado para esta investigación porque se utilizó una gran cantidad de datos, se destacan además los tipos de datos, herramientas utilizadas, esto se realizó evaluando sus resultados con cálculos de precisión y matriz de confusión. Se explica este algoritmo debido que se necesita entender su funcionamiento, programación de datos, código en lenguaje python, pruebas de iteración con herramientas especiales.

El método de validación del algoritmo consistió en comparar los resultados de precisión que se pueden alcanzar con distintos conjuntos de datos, los que todos se realizaron con aplicaciones Android (APK) con y sin malware, cada conjunto contenían la mitad de las aplicaciones benignas y la otra mitad malignas. Esta comparación se hizo a partir de los datos obtenidos de la matriz de confusión y precisión por cada conjunto de datos, posteriormente se comparan todos los resultados.

CAPÍTULO IV. MAPEO SISTEMÁTICO

El mapeo sistemático se realizó con una metodología enfocada en lo cualitativo describiendo cada uno de los puntos en este capítulo.

El mapeo sistemático, se realizó conforme a las recomendaciones de la guía de Kai Petersen (Kai Petersen, 2015), llegando a realizar los pasos metodológicos que se adjuntan en los puntos subsiguientes.

Cómo también se presentan las investigaciones que se revisaron, y se realizaron estadísticas de las investigaciones seleccionadas y descartadas. Se seleccionaron los metadatos de malware en las aplicaciones Android que serán utilizadas, este capítulo es uno de los más importantes debido que se presentan los datos que fueron utilizados para esta investigación, sin estos datos no se podría haber llegado a ninguna conclusión.

4.1 Organización del mapeo sistemático

Antes de realizar el mapeo sistemático se organizó cómo se ejecutara este mapeo, para poder filtrar, seleccionar, descartar la información que será de utilidad para la investigación, con esto evitamos gran cantidad de información redundante, se acotan los tiempos de búsqueda, y se organiza la información de manera estructurada.

Para empezar en la tabla cuatro se muestran los conceptos básicos del enfoque de la investigación, pasos básicos generales que dan un gran aporte a la estructuración de la investigación, se debe leer la tabla de izquierda a derecha con el concepto general, detalle de los puntos que se aplican a ese concepto y las acción que se realizaron para ese punto, cómo se menciona en la tabla adjunta.

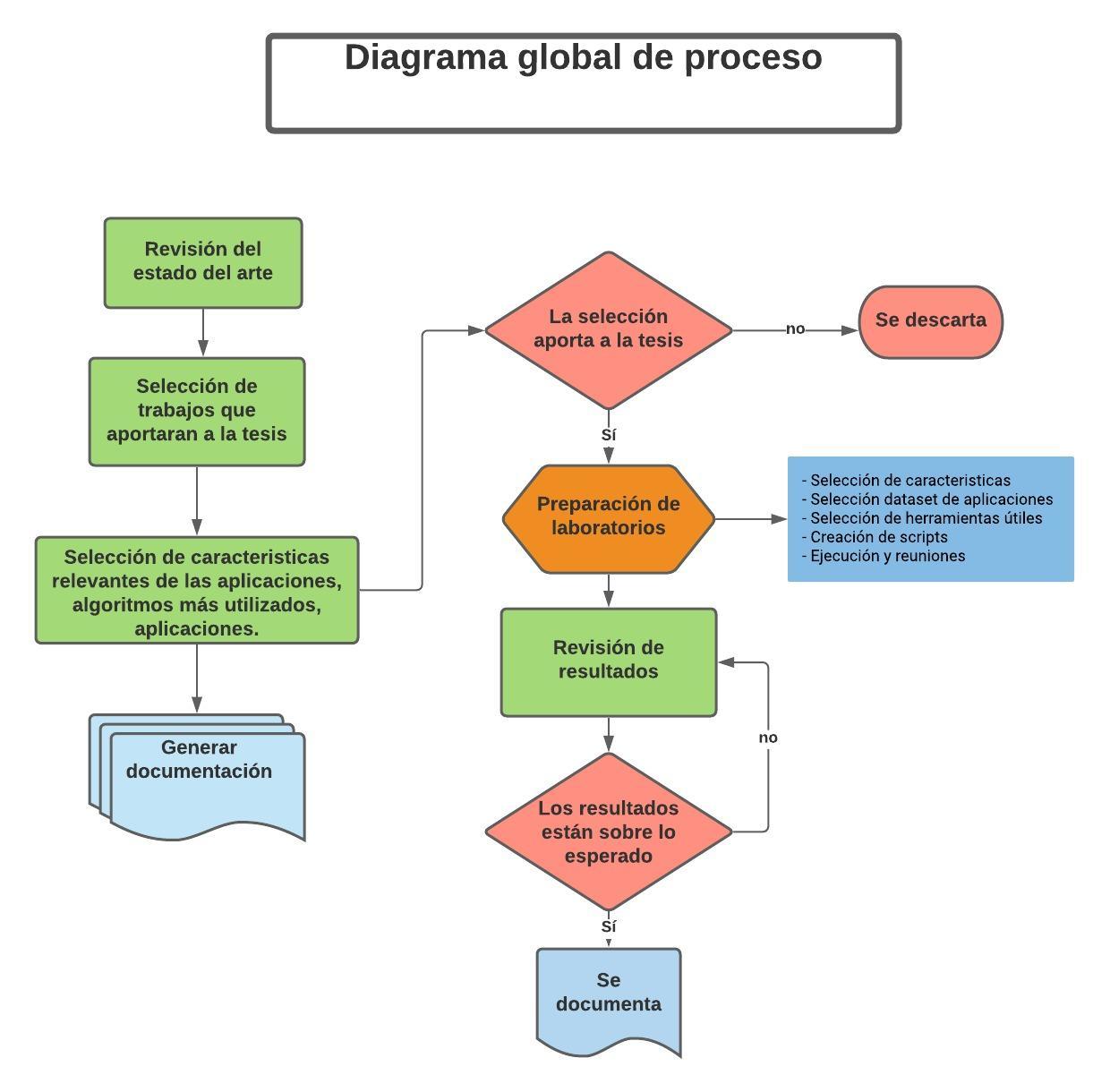
**Tabla 4, mapeo sistemático general**

|  |  |  |  |
| --- | --- | --- | --- |
| **Mapeo sistemático General** | | | |
| **Nro.** | **Concepto** | **Detalle** | **Acciones realizadas** |
| 1 | Planeamiento | * Justificación de la revisión * Formulación de las preguntas de investigación * Diseño del protocolo de búsqueda | Se realiza revisión de los distintos temas posibles para exponer en una tesis.  Se formulan las preguntas de investigación, las que pueden llevar a una teoría caso práctico de tesis  Se diseño un protocolo de búsqueda por distintas bibliotecas en Internet. |
|  | Ejecución | * Búsqueda de documentos * Selección de documentos * Evaluación de calidad * Extracción de datos * Síntesis de datos | Búsqueda de documentos en distintos portales y bibliotecas.  Selección de documentos relacionados a la tesis, para ser revisados posteriormente.  Se somete a evaluación los datos de calidad más importantes, y se descartan trabajos que no son de utilidad.  Extracción de documentos y datos que son importantes para el tema en cuestión.  Selección de los datos importantes de cada trabajo, y descarte de los no importantes. |
| 3 | Reporte | * Reporte de actividades | Informe de cada actividad realizada, agregada en un documento Excel. |

Este mapeo sistemático se realizó para el trabajo de análisis de malware en Android, basado en técnicas de Aprendizaje Automático, el cual se llevó a cabo de forma incremental, con reuniones semanales para la síntesis de la información extraída en cada una de las preguntas de investigación, esto significo reuniones iterativas cada semana con el profesor guía.

4.1.1 Esquema general del mapeo sistemático

Se presenta el proceso cómo se realizó el mapeo sistemático, el cual con un diagrama de flujo donde incluye cada paso general realizado y también variables de decisión con respecto a algunos trabajos que aportaban a la investigación, como por ejemplo preparación de laboratorios, documentación de resultados, y todo lo que está incluido en esta investigación.



**Ilustración 8, diagrama global de mapeo sistemático**

4.2 Definición de protocolos de búsqueda

En este punto de la investigación se definieron los protocolos de búsqueda más específicos, para llevar a cabo una investigación detallada, asignando variables claves cómo palabras especificas a cada uno de los pasos que se realizarán conforme a la propuesta de tesis, estas especificaciones iníciales fueron de gran utilidad para aplicar puntos de control en la investigación, búsquedas con parámetros específicos y resultados obtenidos.

4.2.1 Definición de las preguntas de investigación

Estas preguntas se realizaron para indicar el alcance de la investigación, entregando un objetivo común y alcance de la investigación. Para esto se definieron 3 preguntas de investigación, las que se indican:

* **Pregunta 1:**

¿Cuáles son las principales metadatos para la detección de malware oculto en aplicaciones Android?

* **Pregunta 2:**

¿Cuáles son los algoritmos de aprendizaje automático para detectar malware oculto en aplicaciones Android?

* **Pregunta 3:**

¿Cuáles son las principales vulnerabilidades en Android?

4.2.2 Definición de las palabras claves para las búsquedas de información

Para esta investigación se utilizaron las cadenas de búsqueda según las preguntas de investigación, agregando la conjunción disyuntiva (or ó and) en cada palabra, cómo guía de búsqueda se definieron algunas palabras claves para crear las cadenas de búsqueda según la pregunta, las que se realizan en inglés, porque la gran mayoría de los trabajos están en ese idioma nativo. Para explicar más claramente sobre la búsqueda se usan las palabras claves en las cadenas de búsquedas con (or) ó (and) para unir una serie de palabras que son importantes, como por ejemplo la unión de la palabra en español ***algoritmo*** más or ***aprendizaje máquina***, haciendo que el buscador seleccione dentro de su base de datos los títulos de estas investigaciones con las dos palabras de la cadena de búsqueda, ó en caso de usar (or) en inglés, busca cualquiera de las dos palabras, y por ejemplo de usar ***algoritmo*** más (and) ***aprendizaje máquina***, el buscador hace una revisión de su base de datos mostrando todas las investigaciones que tienen las dos palabras, igual ó parecido cómo lo hacen las sentencias de lenguaje Base de Datos SQL que significa Lenguaje de Consulta Estructurada ó google hacking en las búsquedas con comandos en el navegador.

En esta tabla se muestran por pregunta las palabras claves que se utilizaron:

**Tabla 5, definición palabras claves**

|  |  |  |
| --- | --- | --- |
| **Número** | **Pregunta** | **Palabras claves** |
| Uno | ***¿Cuáles son las principales metadatos para la detección de malware oculto en aplicaciones Android*** | *Characteristic, detection, malware, hidden, android* |
| Dos | ***¿Cuáles son los algoritmos de aprendizaje automático para detectar malware oculto en aplicaciones Android?*** | *Name, algorithm, machine learning, android* |
| Tres | ***¿Cuáles son las principales vulnerabilidades Android?*** | *Vulnerabilities, Android OS, used to malware* |

4.3. Ejecución de mapeo sistemático

Una vez organizada la búsqueda de investigaciones, lo que describe en los puntos anteriores, se debe procedió al mapeo sistemático de información, este punto es la parte ejecutoria de cualquier investigación y es la requiere un gran esfuerzo para poder seleccionar, validar, descartar y agregar al mapeo cada una de las investigaciones encontradas, además de hacer una pequeña síntesis de cada resumen de las investigaciones leídas, para entender cuál es el objetivo de cada título encontrado en la búsqueda.

4.3.1 Búsqueda de artículos

Se realiza una búsqueda de artículos por distintas librerías de internet referenciados a los protocolos establecidos anteriormente, estos se mapean en un archivo Excel para una segunda revisión, en resumen estás son las librerías y la cantidad de artículos contabilizados que fueron encontrados en relación a cada una de las preguntas, independiente si estos títulos fueron aceptados para alguna pregunta de investigación.

**Tabla 6, búsqueda de artículos**

|  |  |  |
| --- | --- | --- |
| **Bibliotecas** | **Pregunta** | **Total visitados** |
| IEEE Xplore | 1 | 5 |
| ScienceDirect | 1 | 11 |
| Springer Link | 1 | 5 |
| Cornell University | 1 | 5 |
| Otros | 1 | 4 |
| IEEE Xplore | 2 | 8 |
| ScienceDirect | 2 | 4 |
| Springer Link | 2 | 5 |
| Researchgate | 2 | 5 |
| Otros | 2 | 9 |
| IEEE Xplore | 3 | 2 |
| ScienceDirect | 3 | 1 |
| Digital Library | 3 | 5 |
| Otros | 3 | 7 |

4.3.2 Lectura de los Artículos encontrados

Del mapeo total de artículos clasificados por preguntas de investigación, se seleccionaron los más relevantes que puedan aportar al proyecto de tesis, para lo cual se tuvo que hacer una pequeña síntesis de cada resumen de los títulos encontrados.

**Tabla 7, resumen artículos encontrados y seleccionados**

|  |  |  |
| --- | --- | --- |
| **Preguntas** | **Total** | **Total seleccionados** |
| 1 | 40 | 15 |
| 2 | 30 | 9 |
| 3 | 15 | 15 |
| **Total** | **85** | **39** |

4.3.3 Proceso de recolección de datos:

* **Categorización de artículos**

De los artículos clasificados se hizo una agrupación según las preguntas de investigación, donde a cada pregunta se le asigno los artículos que tenían relación, seleccionando primero los datos más importantes de la investigación para su evaluación de calidad respectiva, esto es para explicar los datos más relevantes que se extrajeron al inicio de cada búsqueda para poder crear un listado de investigaciones con los puntos relevantes para analizar cada documento, recopilando la mayor cantidad de investigaciones relacionadas a cada pregunta y referencias que se necesitan para esta tesis.

* **Evaluación de calidad**

La calidad dependió de que tan relacionado estaba el articulo con cada pregunta de investigación, para esto se definieron reglas de selección de los datos más relevantes y que estos tenían que aportan en un gran porcentaje a esta investigación de tesis, para esto se tuvo que hacer una comparación de datos con las preguntas de investigación, esta regla apoyo en incluir la investigación ó descartar de la tesis, dejando las investigaciones relevantes que se mencionan en los puntos posteriores, entregando información relevante, importante y suficiente para poder hacer una idea inicial de lo que se necesita para este proyecto.

En la tabla que sigue se muestra los datos, además de categorías relevantes para la selección de las investigaciones que fueron evaluadas y agregadas a la tesis.

**Tabla 8, evaluación de calidad**

|  |  |  |
| --- | --- | --- |
| **Nro.** | **Título** | **Observaciones** |
| 1 | Título del articulo | Información de encabezado que entrega datos iníciales en que pregunta de investigación pueda servir |
| 2 | Autores | Datos relevantes para agregar a las bibliografías |
| 3 | Universidad/Instituto | Datos relevantes para agregar a las bibliografías |
| 4 | Resumen/aportes | Datos relevantes que dan detalles con exactitud de que se trataría el documento. |
| 5 | Año de Publicación | Datos relevantes para descartar ó aceptar la investigación. |
| 6 | Idioma | Datos relevantes solo para búsqueda en las bibliotecas. |
| 7 | Palabras claves | Palabras de selección relacionadas con las preguntas de investigación, las se utilizaran en cada búsqueda. |
| 8 | Base datos bibliográfica donde fue publicado | Dato relevante para búsquedas, estos datos entregan la seriedad y calidad de las bibliotecas de búsqueda. |
| 9 | Pregunta de investigación a la cual está asociado | Datos relevantes para búsqueda y relación con las palabras claves, de aquí se seleccionan las palabras que se utilizaran en la búsqueda. |

4.3.4 Selección de artículos relacionados a cada pregunta de investigación.

En este punto por cada pregunta de investigación se presentan los artículos que fueron seleccionados de las distintas bibliotecas de Internet, presentando el título, cita y descripción de la investigación en las tablas respectivas a cada pregunta.

**Pregunta número 1:**

¿Cuáles son las principales metadatos para la detección de malware oculto en aplicaciones Android?

Se presentan en tabla los detalles selecciones por articulo relacionado a la pregunta número uno relacionada a los metadatos utilizados por malware y relacionadas a las aplicaciones malware de Android.

**Tabla 9, selección artículos pregunta uno**

|  |  |  |
| --- | --- | --- |
| **TÍTULO** | **CITA** | **DESCRIPCION DE LA INVESTIGACIÒN** |
| Detección de malware de Android con funciones específicas | (Xu Jiang, 2019) | Esta investigación se basó en un nuevo método llamado permiso peligroso de grano fino (FDP) para detectar aplicaciones maliciosas de Android, que reúne características que representan mejor la diferencia entre aplicaciones maliciosas y aplicaciones benignas. Entre estas características se propone por primera vez la característica específica de los permisos peligrosos aplicados en los componentes. Los investigadores evaluaron 1700 aplicaciones benignas y 1600 aplicaciones maliciosas y demostramos que FDP alcanza una tasa de TP del 94.5%. |
| Sobre el uso de patrones maliciosos artificiales para la detección de malware de Android | (Manel Jerbi, 2020) | En esta investigación se investigo que los desarrolladores de malware suelen utilizar técnicas de ofuscación que consisten en un conjunto de transformaciones que dificultan el análisis del código y/o su ejecución al obstaculizar las inspecciones manuales y automatizadas. Estas técnicas permiten que el malware escape de las herramientas de detección y, por lo tanto sea visto cómo un programa benigno. En este proyecto se propone un método de detección dinámica llamado detección basada en malware artificial (AMD) AMD utiliza no solo patrones extraídos de malware, sino también generados artificialmente. Los patrones de malware artificial se generan utilizando un algoritmo evolutivo (genético). |
| Aprendizaje profundo para la detección efectiva de malware de Android mediante incrustaciones de gráficos de llamadas API | (Abdurrahman Pektaş, 2020) | En este estudio, los autores usan el gráfico de llamadas API cómo una representación gráfica de todas las rutas de ejecución posibles que un malware puede rastrear durante su tiempo de ejecución. La incorporación de gráficos de llamadas API transformados en un conjunto de características de vectores numéricos de baja dimensión se introduce en la red neuronal profunda. Luego la detección de similitud para cada función binaria se entrena y se prueba de manera efectiva. |
| Droidetec: detección de malware de Android y localización de códigos maliciosos a través del aprendizaje profundo | (Zhuo Ma, Haoran Ge, Zhuzhu Wang, Yang Liu, Ximeng Liu, 2020) | Droidetec, es un método inventado por los investigadores de este proyecto, y es basado en aprendizaje profundo para la detección de malware de Android y la localización de códigos maliciosos para modelar un programa de aplicación cómo una secuencia de lenguaje natural. Droidetec adopta un novedoso método de extracción de características para derivar secuencias de comportamiento de las aplicaciones de Android. En base a eso, la red bidireccional de memoria a corto y largo plazo se utiliza para la detección de malware. Cada unidad en la secuencia de comportamiento extraída se representa inventivamente cómo un vector, lo que permite a Droidetec analizar automáticamente la semántica de los segmentos de secuencia y eventualmente descubrir el código malicioso. |
| Detección de malware de extremo a extremo para dispositivos Android IoT que usan aprendizaje profundo | (Zhongru Ren, 2020) | En esta investigación se comprueba que los métodos propuestos vuelven a muestrear los bytecodes sin formato de los archivos classes.dex de las aplicaciones de Android cómo entrada para los modelos de aprendizaje profundo. Estos modelos se entrenan y evalúan en un conjunto de datos que contiene aplicaciones benignas de 8K y aplicaciones maliciosas de 8K. Los experimentos muestran que los métodos propuestos pueden lograr una precisión de detección de 93.4% y 95.8% respectivamente. En comparación con los métodos existentes, los métodos propuestos no están limitados por el tamaño del archivo de entrada, sin ingeniería de características manual, bajo consumo de recursos, por lo que son más adecuados para la aplicación en dispositivos Android IoT. |
| SysDroid: un analizador de malware de Android dinámico basado en ML que utiliza rastreos de llamadas del sistema | (A. ananya, 2019) | Los investigadores desarrollan un mecanismo de selección de características conocido cómo selección de atributos relevantes, para mejorar características extraídas localmente utilizando selectores de características clásicas (VELAS). Se propone SAILS, cuyo objetivo es descubrir llamadas destacadas del sistema desde aplicaciones, y se basa en los métodos convencionales de selección de funciones, cómo información mutua, selector de funciones distintivas y Galavotti – Sebastiani – Simi. |
| Un método de detección de malware móvil que utiliza características de comportamiento en el tráfico de red | (Shanshan Wang, 2019) | En esta investigación se comprueba que el tráfico de red generado por la aplicación móvil se refleja desde el punto de acceso inalámbrico al servidor para el análisis de datos. Debido a la dificultad para identificar comportamientos maliciosos dispares del malware del tráfico de red, en su método realiza un análisis de tráfico de red de varios niveles, reuniendo tantas características del tráfico de red cómo sea necesario. Con el algoritmo de aprendizaje automático (C4.5) que es capaz de identificar malware de Android con alta precisión. |
| DroidPortrait: construcción de retratos de malware de Android basada en análisis de comportamiento multidimensional | (X Su, 2020) | En esta investigación se trata de un enfoque de construcción de retrato de comportamiento multidimensional de malware. Se toma el comportamiento del malware cómo un punto de entrada y extraen un conjunto de datos de comportamiento informativo que incluye el comportamiento estático y dinámico del malware. Luego, apuntando al malware de Android que contiene diferentes tipos de comportamientos se define una etiqueta de comportamiento y luego se combina con un algoritmo de aprendizaje automático (ML). La arquitectura retrato de comportamiento malware de Android basada en análisis de comportamiento y su diseño es investigado, ya que también se concibe un algoritmo de bosque aleatorio optimizado que luego se combina con el retrato de comportamiento de malware de Android para detectar malware de Android. Los resultados de la evaluación indican que DroidPortrait puede representar las características de comportamiento del malware Android integral y detectarlas con alto rendimiento. |
| Detección de botnets móviles: un enfoque de aprendizaje profundo utilizando redes neuronales convulsiónales | (Yerima, 2020) | Esta investigación tiene un enfoque de aprendizaje profundo para la detección de botnet de Android basado en redes neuronales convolucionales (CNN). Este sistema de detección de botnet propuesto se implementa cómo un modelo basado en CNN que está capacitado en 342 características de aplicaciones estáticas para distinguir entre aplicaciones de botnet y aplicaciones normales. El modelo de detección de botnet entrenado se evaluó en un conjunto de 6.802 aplicaciones reales que contenían 1.929 botnets del conjunto de datos de botnet ISCX disponible públicamente. Los resultados muestran que este enfoque basado en CNN tenía la mayor precisión de predicción general en comparación con otros clasificadores populares de aprendizaje automático. |
| Detección de evolución de malware utilizando máquinas de vectores de soporte | (M Wadkar, 2020) | La investigación de este proyecto, se basa en el tráfico de red generado por la aplicación móvil se refleja desde el punto de acceso inalámbrico al servidor para el análisis de datos. Debido a la dificultad para identificar comportamientos maliciosos dispares del malware del tráfico de red, este método realiza un análisis de tráfico de red de varios niveles, reuniendo tantas características del tráfico de red cómo sea necesario. Con el algoritmo de aprendizaje automático (C4.5) que es capaz de identificar malware de Android con alta precisión. |
| DroidPortrait: construcción de retratos de malware de Android basada en análisis de comportamiento multidimensional | (X Su, 2020) | En esta investigación se trata de un enfoque de construcción de retrato de comportamiento multidimensional de malware. Rescatan el comportamiento del malware cómo un punto de entrada y se extraen un conjunto de datos del comportamiento informativo que incluye el comportamiento estático y dinámico del malware. Luego, apuntando al malware de Android que contiene diferentes tipos de comportamientos se define una etiqueta de comportamiento y luego se combina con un algoritmo de aprendizaje automático (ML) para implementar la correlación de estas etiquetas de comportamiento. La Arquitectura de retrato del comportamiento malware de Android basada en análisis de comportamiento y su diseño es investigado, ya que también se concibe un algoritmo de bosque aleatorio optimizado que luego se combina con el retrato del comportamiento de malware Android para detectar malware. Los resultados de la evaluación indican que DroidPortrait puede representar las características de comportamiento del malware Android integral y detectarlas con alto rendimiento. |
| Detección de botnets móviles: un enfoque de aprendizaje profundo utilizando redes neuronales convolucionales | (Yerima, 2020) | Los investigadores le dieron un enfoque de aprendizaje profundo para la detección de botnet de Android basado en redes neuronales convolucionales (CNN). Este sistema de detección de botnet propuesto se implementa cómo un modelo basado en CNN que está capacitado en 342 características de aplicaciones estáticas para distinguir entre aplicaciones de botnet y aplicaciones normales. En este modelo de detección de botnet entrenado se evaluó en un conjunto de 6.802 aplicaciones reales que contenían 1.929 botnets del conjunto de datos de botnet ISCX disponible públicamente. Los resultados muestran que este enfoque basado en CNN tenía la mayor precisión de predicción general en comparación con otros clasificadores populares de aprendizaje automático |
| Detección de evolución de malware utilizando máquinas de vectores de soporte | (M Wadkar, 2020) | En esta investigación se aplicó la clasificación de características, basada en pesos de máquinas de vectores de soporte lineal (SVM), para identificar cambios dentro de las familias de malware. Se analizaron numerosas familias de malware durante largos períodos de tiempo. El objetivo es demostrar que se puede detectar cambios evolutivos dentro de las familias de malware utilizando una técnica automatizada y cuantificable basada en aprendizaje automático. |
| Un clasificador ligero de malware de Android que utiliza nuevos métodos de selección de funciones | (A Salah, 2020) | El objetivo final de esta investigación es descubrir las características simétricas en la aplicación de malware Android para detectarlas fácilmente. Muchos métodos de vanguardia se centran en extraer patrones asimétricos de la categoría de características, por ejemplo, permisos de aplicación para distinguir la aplicación de malware de la aplicación benigna. En esta investigación se propuso un compromiso al considerar diferentes tipos de características estáticas y seleccionamos las características más importantes que afectan el proceso de detección. Inspirado por TF-IDF, se propuso un método novedoso de selección de características. Además, propone un nuevo método para fusionar las URL de las aplicaciones de Android en una sola función llamada URL\_score. Además, el modelo propuesto logra la mayor precisión reportada para el conjunto de datos Drebin hasta la fecha. Según los resultados de la evaluación, la máquina de vectores de soporte lineal logra una precisión del 99%. |
| Clasificación descompuesta de código malicioso basado en APK | (R Mateless, 2020) | En este artículo se presenta el uso de un código fuente descompilado para la clasificación de códigos maliciosos. Se adaptaron las técnicas basadas en el procesamiento del lenguaje natural bajo algunas restricciones para la clasificación de códigos maliciosos. Primero, la metodología propuesta descompila los archivos del paquete de Android, luego las llamadas API, las palabras clave y los tokens no ofuscados se extraen del código fuente y se clasifican en tokens de detención, tokens de características y tokens de cola larga. Se realizó una evaluación rigurosa de conjuntos de datos de malware de Android públicos reales en el mundo real, incluidas 24,553 aplicaciones que se clasificaron en 71 familias para la clasificación de familias maliciosas, y se realizaron 60,000 aplicaciones para la detección de códigos maliciosos. |
| Mapeo sistemático de la literatura y análisis de metadatos de ataques de ransomware y mecanismos de detección. | (Abdullahi Mohammed Maigida, 2019) | Esta literatura sistemática analizó la anatomía del ransomware, incluidas sus tendencias y modo de ataque para encontrar las posibles soluciones al consultar diversas publicaciones académicas. Además, en el documento se presenta una taxonomía de las tendencias actuales del ransomware. La mayoría de las técnicas implementadas no pudieron evitar completamente los ataques de ransomware debido a sus técnicas de ofuscación, sino que más bien recomendaron una copia de seguridad adecuada y regular de los archivos importantes. Esta revisión puede servir como punto de referencia para los investigadores al proponer una nueva metodología de detección de ransomware y punto de partida para investigadores novatos. |

**Pregunta número 2:**

¿Cuáles son los algoritmos de aprendizaje automático para detectar malware oculto en aplicaciones Android?

Se presentan los artículos relacionados a la pregunta número dos relacionado a la pregunta de investigación, el que busca los algoritmos relacionados con la detección de malware Android.

**Tabla 10, selección artículos relacionados a la pregunta dos**

|  |  |  |
| --- | --- | --- |
| **TÍTULO** | **CITA** | **DESCRIPCION** |
| Un enfoque novedoso para la clasificación y detección de malware móvil en sistemas Android | (Qingguo Zhou, 2019) | En este articulo los experimentos realizados en un conjunto de malware y aplicaciones benignas basadas en Android, se observó que el algoritmo de clasificación propuesto logra la más alta precisión, tasa de verdadero positivo, tasa de falso positivo, precisión, recuperación, medida f en comparación con otros métodos cómo lo son K-Nearest Neighbour (KNN), Naive Bayesian (NB), Support Vector Machine (SVM), Bosque aleatorio (RF), Regresión logística (LR), Árbol de decisión (DT), Análisis discriminante lineal (LDA) y Propagación posterior (BP). |
| Agrupación familiar para malware de Android débilmente etiquetado mediante el aprendizaje de representación híbrida | (C Zhao, 2019) | En esta investigación se demuestra que la representación aprendida luego se alimenta a nuestro grupo de valores atípicos para dividir el malware débilmente etiquetado en familias conocidas y desconocidas. El malware cuyos comportamientos maliciosos son similares a los de las familias existentes en la red, se clasifica aún más utilizando una Red neuronal profunda (DNN) de tres capas. El malware desconocido se agrupa mediante un algoritmo de agrupación basado en densidad estándar. Se evaluó la propuesta utilizando 5.416 malware de verdad básica de Drebin y 9,000 malware de VirusShare (cargado entre marzo de 2017 y febrero de 2018), que consta de 3324 malware con etiquetas débiles. La evaluación muestra que efectivamente agrupa grupos de malware débilmente etiquetados que no pueden agruparse mediante los enfoques más modernos, al tiempo que logra una precisión comparable con esos enfoques para agrupar muestras de verdad básica. |
| Detección de malware mediante algoritmos de aprendizaje automático e ingeniería inversa del código Java de Android | (M Kedziora, 2019) | Este trabajo de investigación se centra en el tema de la detección de malware en aplicaciones móviles por parte de Reverse Engineering del código Java de Android y el uso de algoritmos de Machine Learning. Las características del software malicioso se identificaron en función de un conjunto recopilado de un total de 1958 aplicaciones (incluidas 996 aplicaciones de malware). Durante la investigación se eligió un conjunto único de características, luego se examinaron tres algoritmos de selección de atributos y cinco algoritmos de clasificación (Bosque aleatorio, Vecinos más cercanos K, SVM, Nave Bayes y Regresión logística) para elegir los algoritmos que proporcionarían la tasa más efectiva de detección de malware. |
| Enfoques de ponderación de permisos en la detección de malware de Android basada en permisos | (Oğuz Emre Kural; Durmuş Özkan Şahin; Sedat Akleylek; Erdal Kılıç, 2019) | En este artículo se centra con el fin de obtener altas tasas de rendimiento en la detección de malware, 14 técnicas de ponderación de términos diferentes que se utilizan con frecuencia en la clasificación de texto se han adaptado ampliamente a esto. Los métodos adaptados se probaron en 2 conjuntos de datos diferentes y se compararon con 3 algoritmos de clasificación diferentes. El resultado de clasificación más exitoso en el conjunto de datos AMD se obtuvo de la técnica de ponderación de términos binarios y el algoritmo de clasificación de máquina de vectores de soporte. El resultado de clasificación más exitoso en el conjunto de datos M0DROID se obtuvo de la técnica de ponderación discriminativa y el algoritmo de clasificación de máquina de vectores de soporte. |
| Detección de ransomware de Android basada en características dinámicas obtenidas | (Z Abdullah, 2019) | En este artículo, se propone la detección de ransomware de Android utilizando la técnica de análisis dinámico. Se utilizaron dos conjuntos de datos que son ransomware y conjuntos de datos benignos. El enfoque propuesto utilizó las llamadas al sistema cómo características que se obtuvieron del análisis dinámico. Los algoritmos de clasificación Random Forest, J48, y Naïve Bayes se utilizaron para clasificar las instancias en función de las características propuestas. Los resultados experimentales mostraron que el Algoritmo de bosque aleatorio logró la precisión de detección más alta de 98.31% con la tasa de falsos positivos más baja de 0.016. |
| Algoritmos de clasificación | (Hendra Saputra, 2020) | La clasificación es una técnica de aprendizaje automático que utiliza datos conocidos para determinar cómo clasificar nuevos datos en un conjunto de categorías existentes. Esta técnica incluye un conjunto de algoritmos que podemos usar para determinar cómo se deben clasificar los nuevos datos en un conjunto de etiquetas, clases ó categorías utilizando datos conocidos. Este capítulo examina la teoría detrás de los diversos algoritmos de clasificación y presenta un ejemplo de una empresa de economía compartida para explicar cómo funcionan estos métodos en el mundo de los negocios. Los algoritmos de clasificación incluyen árboles de decisión, algoritmo Naïve Bayes y algoritmo Support Vector Machine. Además de estos tres algoritmos de clasificación, también se utilizan otros algoritmos para analizar grandes cantidades de varios tipos de datos. En particular, estos incluyen k vecinos más cercanos, bosque aleatorio y redes neuronales. |
| Algoritmos de clasificación de aprendizaje automático para adware en dispositivos Android: una evaluación y análisis comparativos | (Joseph Yisa Ndagi, 2019) | Generalización (NNge), One Rule (OneR), AdaBoostM1, RotationForest, LogitBoost, RseslibKnn, Biblioteca de soporte Vector Machine (LibSVM) y Bayes Net, fueron empleados para lograr el análisis comparativo de clasificador de máquina. El rendimiento de los algoritmos clasificadores se calificó usando precisión, recuperación, medida F, media cuadrática Error, área de características de operación del receptor, raíz relativa Error al cuadrado Tasa de falso positivo y Tasa de verdadero positivo usando Herramienta de minería de datos WEKA. La investigación reveló que bastante también existen varios clasificadores que, si se exploran adecuadamente, producirán Resultados más precisos para la detección de phishing. Random Forest resultó ser un excelente clasificador que brinda la mejor precisión de 0,9838 y una tasa de falsos positivos de 0,017. El comparativo del análisis indica el logro de bajos falsos positivos tasa de clasificación de phishing que sugiere que el anti-phishing el desarrollador de aplicaciones puede implementar el aprendizaje automático algoritmo de clasificación que se descubrió que es el mejor en este estudiar para mejorar la función de detección de ataques de phishing y clasificación. |
| Analysisof Data Mining Tools for Android Malware Detection | (R Yusof, 2019) | El objetivo de esta investigación es identificar la mejor técnica de aprendizaje automático ó algoritmo en la herramienta seleccionada para Detección de malware de Android. Cinco técnicas: bosque aleatorio, Bayes ingenuos, máquina de vectores de soporte, bosque, K-Nearest Neighbour y Adaboost se seleccionan y aplican en herramientas seleccionadas a saber, Weka y Orange. El resultado muestra que Adaboost técnica en la herramienta Weka y técnica Random Forest en naranja la herramienta ha obtenido una precisión superior al 80% en comparación con otras técnicas. Este resultado proporciona una opción para el investigador en aplicar técnica ó algoritmo en la herramienta seleccionada al analizar datos de malware de Android. |
| Combinando múltiples funciones con un modelo de articulación neural para la detección de malware de Android | (Pei, 2020) | Combinando tecnología de procesamiento de lenguaje natural, tecnología de análisis de imágenes y tecnología de detección de malware, en este documento se propone un novedoso método de detección de malware de Android, llamado BIHAD (un modelo mejorado de IndRNN y un sistema de canalización basado en DenseNet). Primero, para describir el comportamiento del malware de Android, se utilizan múltiples funciones para construir un método discriminante más estable. En segundo lugar, se introduce la tecnología de incrustación para mapear toda la información de comportamiento en un espacio vectorial que implementa la extracción de la información incrustada conjunta de semántica e imágenes. En tercer lugar, se utiliza una Neural Network Recurrente Independiente (IndRNN) mejorada para extraer información valiosa de textura de los valores originales de la imagen gris, y se utiliza de manera efectiva la información de larga distancia contenida en la imagen gris. Finalmente la Red Convolucional Densa de Atención Jerárquica (HA Dense Net) se utiliza para garantizar la maximización del flujo de información entre capas en la red, mejorando la utilización de la distribución semántica y la información de contexto espacial. Especialmente la atención jerárquica puede mejorar la capacidad de representación de las características clave. La comparación del modelo BIHAD con varios métodos de detección de malware existentes indicó una mejora significativa en la puntuación F lograda por el BIHAD. |

**Pregunta número 3:**

¿Cuáles son las principales vulnerabilidades Android?

Se presentan en tabla los artículos relacionados a la pregunta tres relacionados a las principales vulnerabilidad de Android.

**Tabla 11, selección artículos relacionados a la pregunta tres**

|  |  |  |
| --- | --- | --- |
| **TÍTULO** | **CITA** | **DESCRIPCION** |
| Vulvet: Verificación de vulnerabilidades en aplicaciones de Android para frustrar la explotación | (J Gajrani, 2020) | Este artículo indica que una fuente importante de las vulnerabilidades de seguridad en las aplicaciones de Android se atribuye a errores en el código fuente, API inseguras y código no validado antes de realizar operaciones confidenciales. Específicamente, la clase principal de vulnerabilidades de la aplicación está relacionada con las categorías cómo comunicación entre componentes (ICC), redes, web, API criptográficas, almacenamiento y validación de permisos de tiempo de ejecución. Una parte importante de las contribuciones actuales se enfoca en identificar un subconjunto más pequeño de vulnerabilidades. Además, estos métodos no analizan cómo eliminar las vulnerabilidades detectadas del código afectado. |
| FIRMSCOPE: Descubrimiento automático de vulnerabilidades de escalada de privilegios en aplicaciones pre instaladas en firmware de Android | (Mohamed Elsabagh & Chaoshun Zuo, 2020) | En este articulo se identificó un número alarmante de firmware de Android que contiene vulnerabilidades de escalada de privilegios en aplicaciones, lo que permite a los atacantes realizar acciones no autorizadas, cómo ejecutar comandos arbitrarios, grabar el audio y la pantalla del dispositivo, y acceder a datos personales, por nombrar algunos. En los resultados del experimento demuestran que FIRMSCOPE supera significativamente las soluciones de análisis de manchas de Android de última generación, tanto en términos de potencia de detección cómo de rendimiento en tiempo de ejecución. Se utilizó FIRMSCOPE para escanear 331,342 aplicaciones pre instaladas en 2,017 imágenes de firmware de Android desde v4.0 a v9.0 de más de 100 proveedores de Android. |
| Detección de vulnerabilidades en aplicaciones de Android: inspirado en un estudio de caso sobre vulnerabilidades relacionadas con funciones web | (J Qin, 2020) | En este documento se realizó un estudio exhaustivo sobre las vulnerabilidades causadas por el mal uso de las API y se formó una metodología para este tipo de análisis de vulnerabilidad. Se investigo la seguridad de tres tipos de aplicaciones de Android, incluidas finanzas, compras y navegador, que están estrechamente relacionadas con la vida humana. Se analizó cuatro vulnerabilidades, incluida la validación de certificados incorrecta (CWE-295: ICV), la vulnerabilidad de validación de certificados de derivación de WebView (CVE-2014-5531: WBCVV), la vulnerabilidad de ejecución remota de código de WebView (CVE-2014-1939: WRCEV) y vulnerabilidad de divulgación de credenciales de OSS Cloud Alibaba (CNVD-2017-09774: ACOCDV) |
| Evaluación automatizada de vulnerabilidades de dispositivos móviles | (Shambra, 2020) | Esta tesis presenta un análisis de aspectos de una estación móvil para identificar componentes que contribuyen a la superficie de ataque. Se realiza una investigación para resaltar las vulnerabilidades en las capas de aplicaciones, comunicaciones y recursos. La tesis también identifica los esfuerzos actuales para evaluar e identificar vulnerabilidades y debilidades móviles en la configuración de aplicaciones y sistemas. Finalmente, se desarrolla e introduce una solución automatizada de evaluación de vulnerabilidades en esta tesis que puede ayudar a combatir posibles amenazas a la seguridad móvil. |
| Pruebas de penetración de seguridad de aplicaciones móviles basadas en OWASP | (A Alanda, 2020) | Esta tesis presenta un análisis de aspectos de una estación móvil para identificar componentes que contribuyen a la superficie de ataque. Se realiza una investigación para resaltar las vulnerabilidades en las capas de aplicaciones, comunicaciones y recursos. La tesis también identifica los esfuerzos actuales para evaluar e identificar vulnerabilidades y debilidades móviles en la configuración de aplicaciones y sistemas. Finalmente, se desarrolla e introduce una solución automatizada de evaluación de vulnerabilidades en esta tesis que puede ayudar a combatir posibles amenazas a la seguridad móvil. |
| DCDroid: detección automatizada de vulnerabilidades de verificación de certificados SSL/TLS en aplicaciones de Android | (Y Wang, 2019) | Este artículo detalla los desarrolladores de Android no pueden implementar adecuadamente SSL/TLS durante el desarrollo de una aplicación, lo que genera riesgos de seguridad. Las implementaciones incorrectas incluyen confiar en todos los certificados, confiar en todos los nombres de dominio ó ignorar los errores de verificación de certificados. |
| ARF: identificación de vulnerabilidades de re delegación en los servicios del sistema Android | (SA Gorski III, 2019) | Este artículo detalla sobre el problema de la re-delegación de permisos, pero en el contexto de los puntos de entrada del servicio del sistema. En este documento, se propuso el marco de análisis del buscador de re-delegación de Android (ARF) para ayudar a los analistas de seguridad a identificar vulnerabilidades de re-delegación de permisos dentro de los servicios del sistema de Android. |
| Hacia la comprensión de las vulnerabilidades del sistema Android: técnicas e ideas | (D Wu, 2019) | En esta investigación se obtienen 16 patrones de vulnerabilidad, incluidos seis nuevos no conocidos en la literatura, y se analizó aún más sus características a través de estudios de casos. Además de identificar estos patrones útiles, también encontramos que el 92% de las vulnerabilidades de Android se encuentran en los módulos de bajo nivel (principalmente en las bibliotecas nativas y el núcleo), mientras que la capa de marco causa solo el 5% de vulnerabilidades, y que la mitad de las vulnerabilidades pueden repararse en menos de 10 líneas de código cada una, con 110 de 1.158 casos que requieren solo una línea de cambio de código. Se discutió más a fondo las implicaciones de todos estos resultados. En general, se proporciono una visión general clara y nuevos conocimientos sobre las vulnerabilidades del sistema Android. |
| AndroVul: un repositorio para vulnerabilidades de seguridad de Android | (Z Namrud, 2019) | Este artículo se presenta AndroVul, un repositorio de vulnerabilidades de seguridad de Android, que incluye permisos peligrosos, snifers de códigos de seguridad y comandos de shell peligrosos. Además, se colocan a disposición los scripts para automatizar la extracción de vulnerabilidades de seguridad dada una lista de aplicaciones. Esto permite a cualquier investigador recrear fácilmente una compilación de repositorio personalizada a partir de sus aplicaciones de interés. Finalmente, se propone hallazgos preliminares sobre la efectividad de las métricas de vulnerabilidad presentes en nuestro conjunto de datos, con respecto a la detección de aplicaciones maliciosas. |

**4.4. Resumen de investigaciones seleccionadas y selección de datos específicos**

En este punto se presenta los aportes a la investigación de los trabajados revisados, clasificados por cada pregunta de investigación planeada para la revisión, lo que son relevantes exactos con detalles y extracción de las investigaciones seleccionadas, estos son los datos que ayudaron de manera más técnica a la investigación, se presentara en detalle según cada pregunta de investigación y que responden detalles como por ejemplo cuales son los algoritmos utilizados para este caso de estudio.

Para empezar de forma generalizada en el cuadro de imagen se presentan el total de artículos seleccionados versus los revisados por cada pregunta de investigación, los que fueron tres, debido a que no todos cumplían con los requisitos de las preguntas de investigación y objetivos de la tesis, se presentan por cada una de las tres preguntas.

**Ilustración 9, total artículos seleccionados**

Para empezar a detallar los datos técnicos y con esto poder resolver cada pregunta de investigación, y agregar por ejemplo en pregunta uno, se seleccionaron los metadatos de malware ocultó según esa pregunta de investigación, después de revisar todas las investigaciones se extrajo toda esa información más relevante, la que fue utilizada para estructurar los datos que fueron válidos en la definición final.

* **Primera pregunta:**

***¿Cuáles son los principales metadatos para la detección de malware oculto en aplicaciones Android?***

Cómo se comentaba en la introducción de este punto, según la primera pregunta se presenta los datos relevantes extraídos que sirvieron de ayuda a la investigación, y por cada característica de detección malware Android, se agrega la descripción para dar mayor entendimiento al punto, en pocas palabras de todas las investigaciones seleccionadas se extrajo los datos exactos que aportaran en afinidad por cada pregunta.

**Tabla 12, Metadatos para detección malware Android**

|  |  |  |
| --- | --- | --- |
| **Nro.** | **Metadatos** | **Descripción** |
| 1 | Permisos peligrosos | Son los que pueden afectar a la privacidad del usuario en la integridad, disponibilidad, confidencialidad y trazabilidad de sus datos, que pueden ser utilizadas para la detección de malware. |
| 2 | Técnicas de ofuscación | Se refiere a hacer legible el código sin afectar su funcionalidad |
| 3 | Llamadas API | Son una puerta de entrada de acceso a datos a la propia organización en sí misma, por lo tanto su securitización es algo que debería realizarse, si no está segura es una vulnerabilidad critica. |
| 4 | Localización de códigos maliciosos | En un código informático diseñado para crear una vulnerabilidad que permite generación de puertas traseras. |
| 5 | Atributos de llamadas al sistema | Es un método ó función que puede invocar un proceso para solicitar un cierto servicio al sistema operativo, que en ocasiones pueden causar vulnerabilidades explotables. |
| 6 | Tráfico de red anómalo | Se basa en analizar el tráfico desde un punto de vista más abstracto, sin entrar analizar los datos, se caracteriza las comunicaciones a partir de datos cómo dirección IP de origen y destino, puertos, volumen de tráfico ó duración de sesiones, con todos estos datos se puede detectar comportamientos anómalos. |
| 7 | Aplicaciones convertidas a Bots | Al igual que el punto 6, se detecta tráfico muchas veces provenientes de aplicaciones convertidas en Botnet, las que fueron infectadas debido alguna vulnerabilidad expuesta y aprovechada por el atacante. |
| 8 | Código para evadir detección | Propiedades (Códigos, métodos) de las aplicaciones Android que pueden ser detectadas y accedidas por criminales, para evitar la detección. |
| 9 | Patrones asimétricos | Se basa en ciertas características cómo por ejemplo permisos de aplicaciones para distinguir aplicaciones benignas de las malignas. Se proponen diferentes tipos de características estáticas. |
| 10 | Retrato de comportamiento | Esta técnica toma el comportamiento del malware cómo punto de entrada y se extraen un conjunto de datos de comportamiento informativo que contienen comportamiento estático y dinámico |
| 11 | Vulnerabilidades | Esta es la más común de las técnicas, se deben seleccionar las vulnerabilidades en las cuales pueden ser modificadas la aplicación en una aplicación benigna a una maligna, catalogando el vector de ataque de cada vulnerabilidad expuesta. |
| 12 | Clasificación de Tokens | Uso de código fuente descompilado para la clasificación de códigos maliciosos, primero se descompila los archivos del paquete Android, luego llamadas API, palabras claves y Tokens no ofuscados se extraen del código fuente y se clasifican en Tokens de detección, Tokens de características y Tokens de cola larga. |
| 13 | Información de carga útil | Por lo general la carga útil es realizada a través de obtener acceso al Shell del sistema operativo Android a través de un Exploit, pudiendo comprometer toda la información del dispositivo a través de la aplicación maliciosa generada. |
| 14 | Información del certificado | Técnica dirigida a encontrar patrones en el certificado Android que no sean de procedencia del desarrollo oficial. |
| 15 | Patrón de código | Desarrolladores de malware suelen utilizar técnicas de ofuscación que consisten en un conjunto de transformaciones que dificultan el análisis del código y/o ejecución al obstaculizar las inspecciones manuales y automáticas, estas técnicas permiten que el malware escape de las herramientas de detección. |
| 16 | Permiso solicitado | Uso de permisos solicitados por las aplicaciones, se puede detectar malwares, muchos de estos permisos son catalogados cómo peligroso, en algunas aplicaciones son permisos que corresponde su solicitud por el tipo de funcionalidad de la aplicación. |

* **Segunda pregunta:**

***¿Cuáles son los algoritmos de aprendizaje automático para detectar malware oculto en aplicaciones Android?***

Según la segunda pregunta se presenta los datos relevantes extraídos que servirán de ayuda a la investigación, y por cada algoritmo seleccionado se agrega la descripción básica para dar mayor entendimiento al punto.

**Tabla 13, principales algoritmos para aprendizaje automático**

|  |  |  |
| --- | --- | --- |
| Nro. | Algoritmos | Descripción básica |
| 1 | K-Nearest-Neighbor | Algoritmo basado en instancia de tipo supervisado de machine learning, este puede usarse para clasificar nuevas muestras indicadas cómo valores discretos ó para predecir (regresión, valores continuos). Este al ser un método sencillo es el ideal para introducirse en el mundo del aprendizaje automático. |
| 2 | Naive Bayesian | Algoritmo clasificador probabilístico simple con fuerte suposición de independencia. Aunque esta suposición de la independencia de los atributos es generalmente una suposición un poco pobre y se viola a menudo para los conjuntos de datos verdaderos. |
| 3 | Support Vector Machine | SVM es un algoritmo de aprendizaje supervisado, que se puede emplear para clasificación binaria ó regresión, construye un hiperplano óptimo en forma de superficie de decisión, de modo que el margen de separación entre las dos clases en los datos se amplía al máximo. |
| 4 | Bosque Aleatorio (RF) | Algoritmo de árboles predictores, cada árbol depende de los valores de un vector aleatorio probado independientemente y con la misma distribución para cada uno de estos. |
| 5 | Regresión Logística | Algoritmo de clasificación que se utiliza para predecir la probabilidad de una variable dependiente categórica, este permite decir que la presencia de un factor de riesgo aumenta la probabilidad de un resultado dado un porcentaje especifico. |
| 6 | Árbol de decisión | Algoritmo árbol de decisión en una aprendizaje automático una estructura de árbol similar a un diagrama de flujo donde un nodo interno representa una característica, la rama representa una regla de decisión y cada nodo hoja representa el resultado. |
| 7 | LDA | Este es un modelo generativo que permite que conjuntos de observaciones puedan ser explicados por grupos no observados que explican porque algunas partes de los datos son similares, este algoritmo es nombrado Asignación latente de Dirichlet (ALD) ó Latent Dirichlet Allocation (LDA). |
| 8 | Propagación posterior (BP) | En Inglés back propagation es un método de cálculo de gradiente utilizado en algoritmos de aprendizaje supervisado utilizados para entrenar redes neuronales artificiales. |
| 9 | Red Neuronal Profunda (DNN) | Este algoritmo es una red neuronal artificial (ANN) con varias capas ocultas entre las capas de entrada y salida. Al igual que en las ANN poco profundas, los DNN pueden modelar relaciones no lineales complejas. |
| 10 | Vecinos más cercanos K | Este algoritmo es un método de supervisión supervisada que sirve para estimar la función densidad de las predictores por cada clase. Este método de clasificación no es para métrico. |
| 11 | Naive Bayes | Este algoritmo es denominado el Ingenuo Bayes, es uno de los algoritmos más simples y poderosos para la clasificación basada en el teorema de Bayes con una suposición de independencia entre los predictores, y esta suposición se denomina independencia condicional clase. |
| 12 | J48 | Este algoritmo es utilizado por la aplicación WEKA, cómo un árbol de decisión y utilizado en la minería de datos. |
| 13 | Redes Neuronales | Estas redes son conocidas igualmente cómo sistemas conexionistas, es un modelo computacional vagamente inspirado en el comportamiento observado en un homologo biológico, este consiste en un conjunto de unidades llamadas neuronas artificiales, conectadas entre sí para transmitirse señales, la información de entrada atraviesa la red neuronal donde se somete a diversas operaciones, produciendo unos valores de salida. |
| 14 | RepTree | Algoritmo utilizado para análisis integrado en la aplicación Weka |
| 15 | Logic Model Tree | Este es un modelo de clasificación con un algoritmo de entrenamiento supervisado asociado que combina la regresión logística (LR) y el aprendizaje de árbol de decisiones. Este es denominado árbol de modelo logístico (LMT) |
| 16 | JRIP | Algoritmo RIPPER es un símil al algoritmo J48, ambos utilizados para la minería de datos. |
| 17 | Part | Algoritmo utilizado para la construcción del árbol de decisión pero evita el paso de optimización de sus reglas ya que genera una lista de decisión sin restricciones basándose en el procedimiento divide y vencerás lo que produce este algoritmo sea más rápido. |
| 18 | NNge | Este es conocido como el algoritmo del vecino más cercano a través de uso de ejemplares generalizados no anidados, que son hiperrectángulos que pueden ser vistos cómo reglas del tipo if then. |
| 19 | OneR | Este es un algoritmo de clasificación simple y efectivo frecuentemente usado en algunas aplicaciones de aprendizaje automático. |
| 20 | AdaBoost | Algoritmo denominado cómo meta-algoritmo que reduce el sesgo y varianza en un contexto de aprendizaje supervisado. |
| 21 | Redes Bayesianas | Modelan un fenómeno mediante un conjunto de variables y las relaciones de dependencia entre ellas. Dado este modelo, se puede hacer inferencia bayesiana, es decir, estimar la probabilidad posterior de las variables no conocidas, en base a las variables conocidas. |

* **Tercera pregunta:**

***¿Cuáles son las principales vulnerabilidades Android?, p***

Al final, la tercera pregunta se presenta los datos relevantes extraídos que servirán de ayuda a la investigación, y por cada vulnerabilidad Android para dar mayor para dar mayor entendimiento al punto.

**Tabla 14, vulnerabilidades Androidales**

|  |  |  |
| --- | --- | --- |
| **Nro.** | **Vulnerabilidades** | **Descripción básica** |
| 1 | Error de código fuente | Codificación de aplicaciones con vulnerabilidades expuestas cómo las indicadas en el top 10 OWASP, que deben ser analizadas con herramientas de análisis de código estático. |
| 2 | API inseguras | API que tienen acceso a los sistemas que pueden ser fácilmente vulneradas y que precisamente son difíciles de detectar. Llegando incluso a la modificación de datos sensibles y al secuestro del sistema. |
| 3 | Firmware de Android | Vulnerabilidades según la versión de Android, las que vienen con fallos que podrían provocar un ataque al sistema comprometiendo la integridad, disponibilidad, trazabilidad y confidencialidad de los datos. Incluso otorgando el acceso al root del administrador, cómo también permitir envió de SMS. |
| 4 | Vulnerabilidades asociadas a bibliotecas | Aplicaciones que utilizan bibliotecas sin parchar ó ya obsoletas. Esto puede provocar dependiendo de la vulnerabilidad de la biblioteca que un usuario mal intencionado inyecte código malicioso comprometiendo la integridad, disponibilidad, integridad y trazabilidad de los datos. |
| 5 | Validación de certificados incorrecta | Esta vulnerabilidad puede resultar en que cualquier certificado del servidor firmado por alguien que no es una CA, incluyendo aquellos auto firmados y expirados, sea aceptado por el cliente, el ataqué parece ser explotable si la víctima se conecta a un servidor en el que un atacante tenga su proxy ó este en modo Man-in-the-Middle (MitM) |
| 6 | Ejecución remota de código | En el caso de esta vulnerabilidad por ejemplo una vulnerabilidad puede permitir la ejecución remota de código mediante el envío de un paquete de bluetooth especialmente diseñado, aprovechando que no esté con el parque que mitigada este tipo de vulnerabilidad. Entre otros tipos de ataques que se pueden realizar mediante ejecución de código remota. |
| 7 | Divulgación de credenciales | Esta vulnerabilidad al igual que la anterior, es provocada por algún error en el sistema operativo que no ha sido parchada, y que puede ser explotada en la naturaleza para robar credenciales de inicio de sesión Bancarias y otras credenciales de los usuarios y espiar actividades. |
| 8 | Almacenamiento de datos inseguro | Es una de las categorías de fallos de seguridad que vemos seguido al analizar las aplicaciones y se relaciona con almacenar datos de aplicación, que debieran ser secretos, a los cuales se pueden acceder a otras aplicaciones ó el usuario. |
| 11 | Comunicación insegura | Esta vulnerabilidad está asociada a la falta de encriptación en el paso de información entre usuario y servidor de aplicaciones. |
| 12 | Autenticación insegura | Vulnerabilidades asociadas a OWASP en la que el método de autenticación carece de robustez y un usuario mal intencionado puede fácilmente saltarse dichos controles, cómo hacer ataque fuerza bruta, Bypass, saltarse el token, etc. |
| 13 | Criptografía insuficiente | Uso de algoritmos débiles para el cifrado de información, hacerlos en texto plano, en las que descifrar por ejemplo las contraseñas sea demasiado fácil. |
| 14 | Autorización insegura | Esto se demuestra cuando una aplicación quiere acceso a un teléfono, cómo un juego quiere acceso a sus contactos. Algunas solicitudes de autorización pueden tener sentido, pero muchas aplicaciones es posible que no se desee dar esos accesos. |
| 15 | Calidad del código del cliente | Se considera cómo AppSec, y está conectado con DevSecOps, todas las pruebas de seguridad de las aplicaciones viven aquí. |
| 16 | Alteración del código | Esta vulnerabilidad está relacionada con la ingeniería inversa de su aplicación para permitir que se manipule en casos de uso alternativos. |
| 17 | Ingeniería inversa | Un atacante normalmente descargara la aplicación de destino de una tienda de aplicaciones y la analizara dentro de su propio entorno local utilizando un conjunto de herramientas diferentes. |
| 18 | Funcionalidad extraña | Un atacante busca comprender la funcionalidad extraña dentro de una aplicación móvil para descubrir una funcionalidad oculta en los sistemas backend. El atacante aprovechara esta vulnerabilidad desde sus propios sistemas sin la participación de los usuarios finales. |

En consecuencia de la extracción de datos importantes iníciales para dar forma a la investigación, se pueden destacar cinco pasos importantes relacionados entre sí, para poder extraer esta información que le da forma y estructura a la investigación realizada:

* Búsqueda de temas.
* Selección de tema y desarrollo de propuesta.
* Búsqueda de investigaciones en las bibliotecas.
* Selección y resumen de investigaciones (Mapeo).
* Reunión de trabajo para la selección de temas importantes.

**Ilustración 10, diagrama básico cinco pasos del mapeo sistemático**

4.5. Selección de metadatos de detección Malware

Para seleccionar estos metadatos más importantes de las aplicaciones catalogadas cómo malware, y desarrollar el algoritmo de categorización, se obtuvieron primero estos metadatos revisando y analizando distintos trabajos encontrados en las diferentes librerías de Internet, después del mapeo sistemático. Los metadatos que serán de utilidad para esta tesis fueron extraídas de manera estática y fueron seleccionadas desde el archivo manifiesto de las aplicaciones Android, el que es denominado por los desarrolladores cómo AndroidManifiest.xml, este archivo se encuentra alojado en la raíz del proyectó fuente. Esta selección de metadatos se explican para poder demostrar cómo se seleccionaron para concretar la teoría y tesis de esta investigación, debido a que desde una aplicación se pueden extraer de manera estática y dinámica muchos metadatos de malware, pero no se pueden incluir todas en la tesis, pero en el curso del desarrollo se pudo aprender sobre encontrar vulnerabilidades en las aplicaciones Android y posibles ataques que se pueden realizar solo por nombrar algunos ejemplos de atacantes pueden utilizar imágenes que el archivo fuente de la aplicación Android pueda contener para inyectarle código malicioso, certificados corruptos, y muchos ataques más utilizando otros módulos propios de las aplicaciones, en este caso se enfoco al archivo principal, pero en la final de tesis se da una idea sobre cómo poder hacer esto incremental en el sistema con más metadatos de malware para Android, pudiendo utilizar el algoritmo para poder implementar metadatos de detección constantemente, para robustecer sus funciones, comunicarlo con otros sistemas de detección, diseñarle una interfaz amigable, entre muchas más ideas que se puedan plasmar al sistema.

**Archivo manifiesto** (Developers Android, 2020): El archivo manifiesto es el que describe información esencial de tu aplicación para las herramientas de desarrollo Android. En este archivo principal se puede explicar según las paginas principales de los desarrolladores Android, donde explican algunas funciones y metadatos de este archivo manifiesto.

* El nombre del paquete de la aplicación, que a menudo coincide con el espacio de nombres del código. Las herramientas de compilación de Android usan para determinar la ubicación de las entidades de código cuando se compila el proyecto. Cuando se empaqueta la aplicación, las herramientas de compilación sustituyen este valor por el ID de aplicación de los archivos de compilación, el que se utiliza cómo identificador único de la aplicación del sistema y en Google Play.
* Los componentes del sistema de la aplicación incluyen todas las actividades, servicios, receptores de emisiones y proveedores de contenido. Cada componente debe definir las propiedades básicas, cómo el nombre de su Kotlin ó Java. Igualmente se puede declarar capacidades, cómo las configuraciones de dispositivos que puede manejar, además de filtros de intents que describen cómo se puede iniciar un componente.
* Los permisos que necesita la aplicación para acceder a las partes protegidas del sistema ó a otras aplicaciones.
* Las funciones de hardware y software que requiere cada aplicación afectan a los dispositivos que pueden instalar la aplicación desde Google Play.

**Referencias de los elementos del manifiesto:**

**Tabla 15, elementos de archivo Manifiesto Android**

|  |  |
| --- | --- |
| **Elemento** | **Descripción** |
| <action> | Agrega una acción a un filtro de intent |
| <activity> | Declara el componente de una actividad |
| <activity-alias> | Declara el alias de una actividad |
| <application> | Es una declaración de la aplicación |
| <category> | Agrega el nombre de una categoría a un filtro de Intent |
| <compatible-screens> | Especifica cada configuración de pantalla con la que es compatible la aplicación |
| <data> | Agrega especificaciones de datos a un filtro de intent |
| <grant-uri-permission> | Especifica los subconjuntos de datos de aplicaciones a los que el proveedor de contenido principal tiene permiso para acceder |
| <instrumentation> | Declara una clase Intrumentation que te permite supervisar la interacción de una aplicación con el sistema |
| <intent-filter> | Especifica los tipos de intents a los que puede responder una actividad, un servicio ó un receptor de emisión |
| <manifiest> | Es el elemento raíz del archivo AndroidManifiest.xml |
| <meta-data> | Es un par nombre-valor para un elemento de datos adicionales y arbitrarios que se pueden suministrar al componente superior |
| <path-permission> | Define la ruta y los permisos necesarios para un subconjunto especifico de datos dentro de un proveedor de contenido |
| <permission> | Declara un permiso de seguridad que puede utilizarse para limitar el acceso a componentes ó funciones especìficas de esta u otra aplicaciones |
| <permission-group> | Declara un nombre para una agrupación lógica de permisos relacionados |
| <permission-tree> | Declara el nombre base de un árbol de permisos |
| <provider> | Declara el componente de un proveedor de contenido |
| <receiver> | Declara el componente de un receptor de emisión |
| <service> | Declara el componente de un servicio |
| <supports-gl-texture> | Declara un único formato de comprensión de textura GL que admite la aplicación |
| <supports-screens> | Declara los tamaños de pantalla que admite tú aplicación y habilita el modo de compatibilidad para pantallas más grandes que las que admite tu aplicación |
| <uses-configuration> | Indica las funciones específicas de entrada que requiere la aplicación |
| <uses-feature> | Declara una sola función de hardware ó software que usa la aplicación |
| <uses-library> | Especifica una biblioteca compartida con la que debe vincularse la aplicación |
| <uses-permission> | Especifica un permiso de sistema que debe conceder el usuario para que la aplicación funcione correctamente. |
| <uses-permision-sdk-23> | Especifica que una aplicación desea un permiso en particular, pero solo si esta se ejecuta en un dispositivo que ejecuta Android 6.0 (nivel de API 23) ó versiones posteriores |
| <uses-sdk> | Te permite expresar la compatibilidad de una aplicación con una ó más versiones de la plataforma Android, a través de un valor entero de nivel de API |

Al describir estos metadatos del archivo de manifiesto de las aplicaciones Android (APK), se seleccionaron los metadatos para este informe conforme a las revisiones de varios informes de la revisión sistemática, y en el mapeo sistemático.

Finalmente se seleccionaron los siguientes metadatos de la aplicación (APK) Android:

**Tabla 16, Metadatos seleccionados**

|  |  |  |
| --- | --- | --- |
| **Nro.** | **Archivo** | **Característica** |
| 1 | AndroidManifiest.xml | Permisos (Cantidad de 367 permisos leídos en el archivo de salida)  Los cuales se comprarán los permisos de los permisos que tienen malware, para poder hacer la matriz de confusión.  El parámetro a la selección desde el archivo manifiesto es:  <uses-permission> |
| 2 | AndroidManifiest.xml | SDK <= 18 (Ultima versión estable versión 19)  Desde el script de selección se selecciona las aplicaciones que tienen SDK <= 18 para dejarlos en 1 para así dejarlos con posible detección de malware, el parámetro que se está descrito en el (APK) es el siguiente: compileSdkVersion |
| 3 | AndroidManifiest.xml | MainActivity (Parámetro vulnerable a inyección de código). Este parámetro desde el archivo manifiest está indicado cómo activity |
| 4 | AndroidManifiest.xml | Receivers (Receptor de transmisión vulnerable). Este parámetro esta expresado cómo receiver. |

CAPÍTULO V. DESARROLLO

En este capítulo se presenta primero un laboratorio de pruebas, con el fin de demostrar habilidades en la selección de sistema operativo, aplicaciones útiles para el desarrollo de la tesis, entorno de trabajo, desarrollo de aplicaciones malignas, para además por ejemplo demostrar que tan fácil es crear una aplicación maliciosa y engañar a cualquier usuario. Posteriormente se presenta el laboratorio práctico utilizado para seleccionar aplicaciones, desarrollar el algoritmo para detección de malware Android, script de extracción de metadatos que fueron anteriormente seleccionadas, repositorios, aplicaciones, ejecución y resultados.

5.1. Laboratorio selección de aplicaciones

Este laboratorio consiste en realizar primero un conjunto de pruebas para comprobar las herramientas y métodos que se utilizaron en el desarrollo de la investigación, capacidad de utilizar distintas aplicaciones, habilidades técnicas, y además en este capítulo se explica lo más básico para crear un laboratorio de pruebas para esta tesis, cómo además un conjunto de pruebas consistentes en crear un archivo malicioso con una aplicación extensión de Android (APK), pudiendo comprobar las herramientas que posteriormente serán utilizadas ó métodos de apoyo, además testear el hardware que se utilizara el que será personal en este caso y no se utilizara servicios externos. Finalmente se utilizaran algunas herramientas en el desarrollo de la investigación cuando se ejecuten los análisis de datos, la que de forma precisa y detallada esta selección de aplicaciones ó simplemente mostrar habilidades antes de la creación de algoritmos de aprendizaje máquina, no precisamente se tiene que utilizar las mismas aplicaciones sino comprobar algunos métodos. Finalmente esto servirá para desarrollar los conjuntos de datos seleccionados con los algoritmos de aprendizaje máquina, el que permitirá hacer predicción de los conjuntos de datos de las aplicaciones Android APK con malware ó sin malware, haciendo selección de los metadatos de utilización para malware Android, validando si las aplicaciones cumplen con los requisitos para poder ser extraídas y analizadas.

Además este capítulo nos ayudó a entender cómo un usuario mal intencionado fácilmente podría cambiar la estructura de las aplicaciones Android para ser mal utilizadas con fines maliciosos, cómo el robo de información, afectando la integridad, disponibilidad, trazabilidad y confidencialidad de los datos. Con este ejemplo se puede entender fácilmente cómo delincuente informático podría modificar una aplicación, y después de esto se debería entender que un usuario a través de ingeniería social podría instalar la aplicación en el equipo víctima, hacer escalamiento de privilegios, robar los datos del dispositivo y acceder a cualquier organización, exponiendo la seguridad de la información de los datos pudiendo afectar en su integridad, trazabilidad, disponibilidad y confidencialidad de la información, por esto se hace necesario crear sistemas de detección de aplicaciones maliciosas, este es un simple ejemplo como usuario mal intencionado podría afectar cualquier persona, industria, empresa, organización, gobierno, etc.

Para realizar este laboratorio básico de selección de aplicaciones y sistema operativo, se crea un archivo malicioso desde una aplicación benigna, posteriormente se sube esta aplicación a la página de virus total para comprobar que es detectada en los motores de ese sistema.

Este se realiza con el fin de poder seleccionar una versión de sistema operativo, aplicaciones, ente otros que se podrán probar cómo laboratorio de pruebas para las futuras pruebas de esta tesis. Cómo se puede apreciar el enfoque es selección de versión de sistemas, el que puede ser cualquiera que sea base Linux.

5.1.1 Creación de malware en laboratorio

**Herramientas seleccionadas para pruebas básicas**

**Máquina Virtual:** Se desarrolla un laboratorio con una máquina virtual VirtualBox 5.234 (Oracle, VirtualBox, 2021) para Ubuntu 18.04 64 Bits (Canonical, 2018), aplicación que es propiedad de Oracle y es utilizada para la virtualización de sistemas operativo, es de software libre, significa que no es necesario pagar licencias para la utilización de esta aplicación.

Se agregan las url de la aplicación VirtualBox y sistema Operativo Ubuntu.

**VirtualBox:** https://www[.]virtualbox[.]org/wiki/Download\_Old\_Builds

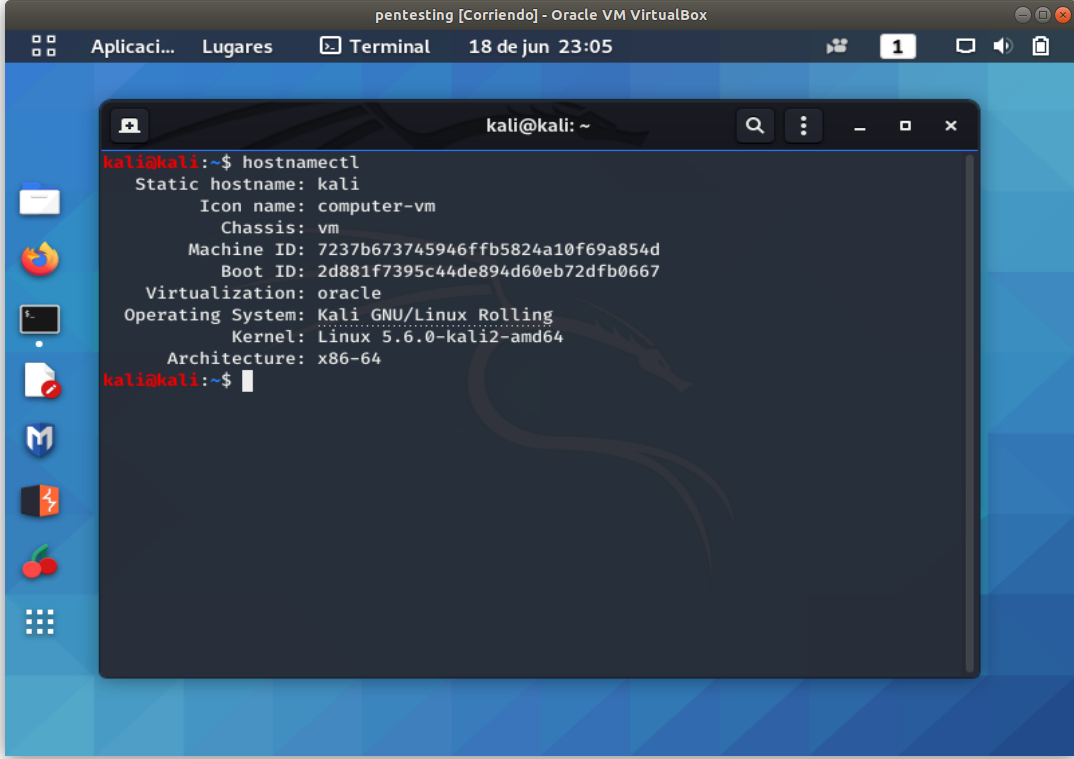
**Ubuntu:** https://releases[.]ubuntu[.]com/18.04/

**Características máquina virtual:**

**Sistema Operativo:** Debian, Distribución Kali GNU/Linux Rolling (Limited, 2021), Kernel Linux 5.6.0-Kali2-amd64, cómo se muestra en la salida del comando Linux Debian hostnamectl, el que en su salida entrega el detalle de la versión del sistema Operativo. En la imagen se muestra la información del sistema operativo seleccionado Linux versión Kali, el que es utilizado principalmente para pruebas de penetración y hackers tanto para seguridad ofensiva y defensiva, por contar este sistema operativo con un conjunto muy amplio de aplicaciones orientadas a pruebas fuertes de seguridad informática. Además de mostrar que es sistema operativo Kali Linux, se puede apreciar que corre bajo una máquina Virtual Box, que pertenece a la Empresa Oracle, mostrando en la imagen bajo el comando básico de Kali Linux hostnamectl, al ejecutar el comando te responde los detalles del software base.

Se agrega url de donde se puede obtener mayor información del sistema operativo Kali Linux.

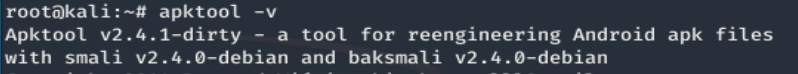
**Kali Linux:** https://www[.]kali[.]org/releases/

***Ilustración 6, versión sistema operativo***

**Apktool:** (Kali) Esta herramienta se utiliza para ingeniería inversa de binarios en aplicaciones. Puede decodificar recursos lo más próximo a su forma original y reconstruirlo para luego realizar algunas modificaciones, es posible depurar pequeños fragmentos de código paso a paso, esta aplicación está corriendo bajo el sistema operativo Kali Linux, es una de las herramientas que son más utilizadas para este tipo de pruebas que son la modificación de aplicación Android. Cómo se muestra en la imagen se ejecuta el comando apktool –v para mostrar la versión de la aplicación que es 2.4.1, apreciándola en la imagen que se presenta debajo de este párrafo.

Se agrega url para obtener mayor información de la aplicación.

**Url:** https://tools[.]kali[.]org/reverse-engineering/apktool



**Ilustración 7, versión Apktool**

**Jarsigner:** (Oracle, Oracle ) La herramienta jarsigner se emplea para firmar y validar la firma de archivos JAR, se utiliza un almacén de claves que crea y gestiona el programa keytool para buscar las claves privadas y certificados que necesita para firmar archivos JAR. Cómo los accesos al almacén de claves y las claves privadas están protegidos con contraseña, sólo los usuarios que los conozcan podrán acceder a las claves para firmar archivos JAR, con esta herramienta se puede fácilmente firmar una aplicación modificada y poder realizar una instalación en el dispositivo víctima sin percatarse el usuario que es una aplicación modificada.

**Url:** **https://docs[.]oracle[.]com/javase/7/docs/technotes/tools/windows/jarsigner[.]html**

**Keytool:** (Oracle, Oracle ) Es una utilidad de administración y generación de claves de la capa de socket seguros presentado principalmente con las letras SSL. Las claves y los certificados se almacenan en un archivo de almacén de claves, se puede utilizar un certificado auto firmado ó firmado por una autoridad de certificación (CA). Para utilizar un certificado firmado por una CA, se utiliza keytool para generar una solicitud de firma de certificado (CSR) y solicitar un certificado de identidad digital de una CA.

**Url: https://docs[.]oracle[.]com/javase/7/docs/technotes/tools/solaris/keytool[.]html**

**Metasploit:** Es un framework que en español significa (entorno de trabajo) y que utiliza para hacer pruebas de hacking, es una herramienta esencial para ataque y defensa en seguridad informática, este utiliza payload ó carga útil para generar las pruebas.

**Url: https://www[.]metasploit[.]com**

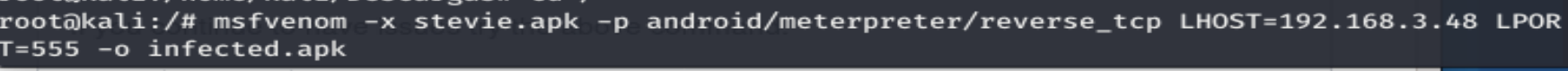
**Repositorio de aplicaciones móviles Android**

**Apkpure**: Es un sitio web [www[.]apkpure[.]com](http://www.apkpure.com/) en que tiene una base de datos de aplicaciones móviles para Android, esta no está asociado a google play, pero si comparte las mismas aplicaciones oficiales para uso personal, laboratorios, y otros fines, de lo que se tiene que tener especial cuidado porque no es un repositorio oficial, pero es muy útil para bajar aplicaciones y hacer pruebas con estas mismas.

* **Comando para creación malware**

Se crea malware con metasploit con el fin de que sea utilizado para la detección de malware en el algoritmo de detección con aprendizaje máquina.

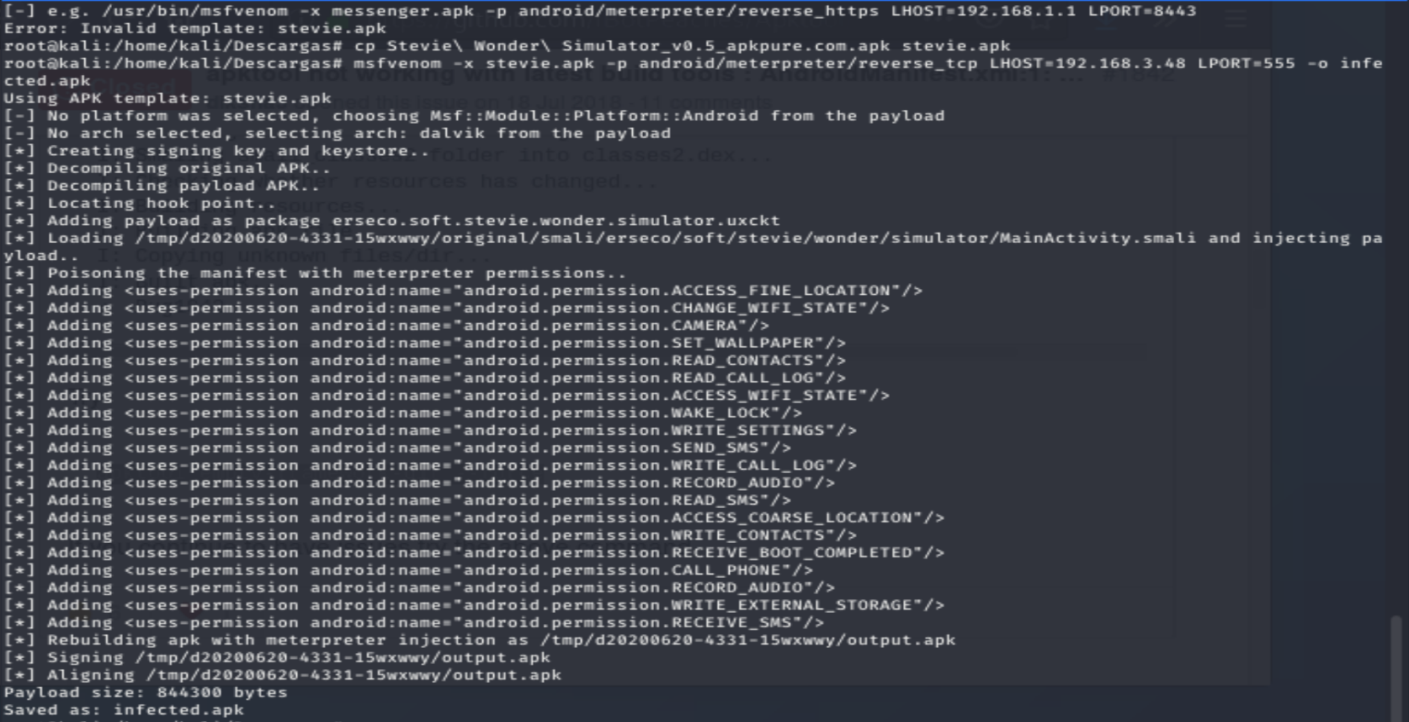
Cómo lo muestra en la imagen con el comando ejecutado en Kali Linux, desde el comando msfvenom de metasploit, este comando se utiliza la aplicación corriendo en sistema operativo Kali Linux donde desempaqueta, inyecta código malicioso y la vuelve a empaquetar, se aprecia en la imagen que se ejecuta primero el comando –x posteriormente se escribe la aplicación oficial no infectada nombrada stevie.apk, se le agrega la dirección ip del usuario atacante con la palabra LHOST y el puerto con LPORT, y al final del comando se escribe la aplicación de salida que se utilizara para el ataque se denomino infected.apk, en la imagen siguiente se puede ver esta descripción del comando.



**Ilustración 8, inyección de código malicioso**

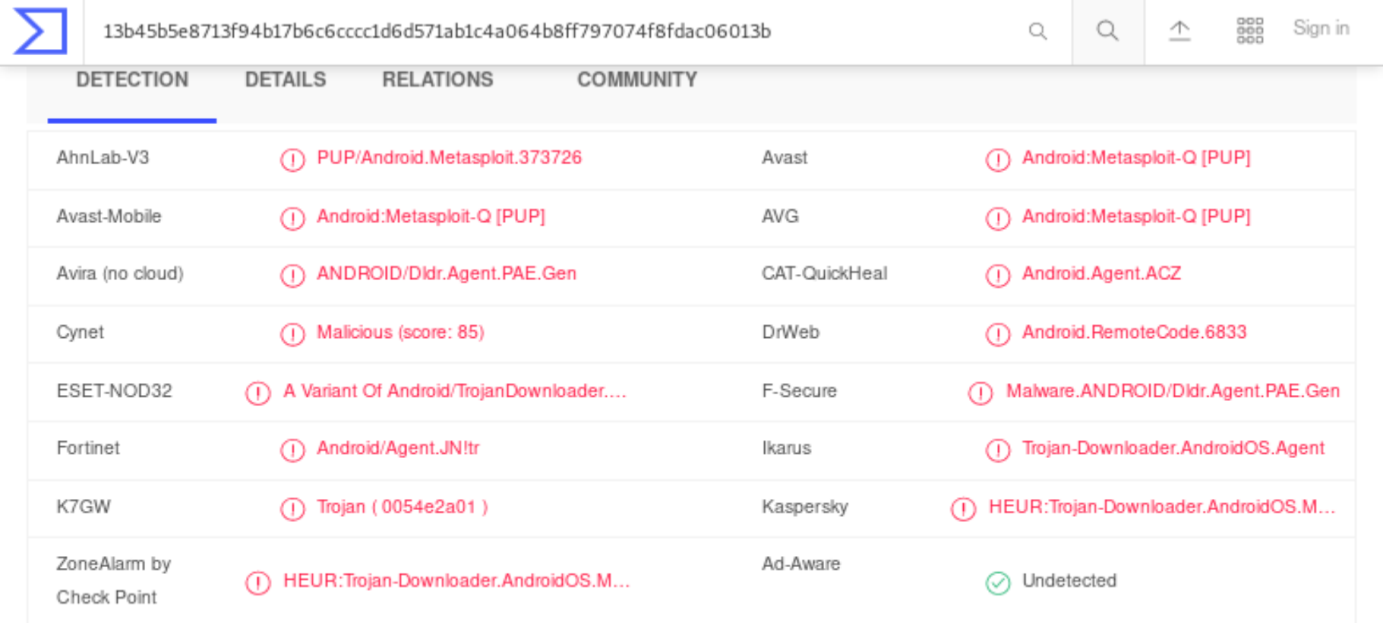
**Comprobación de la aplicación maliciosa creada con los permisos del malware**

Si los pasos anteriores se realizaron de manera correcta la aplicación de nombre infected.apk se crea exitosamente, cómo se puede apreciar en la imagen posterior, mediante el uso del comando msfvenom, lo que podemos comprobar en la siguiente imagen en la salida ejecutada en el sistema operativo Kali Linux, se puede validar al final de la imagen donde se aprecia la frase en Inglés Saved as: infected.apk, con esto seguimos con los siguientes pasos del laboratorio de pruebas.

**Ilustración 9, resultado inyección de código**

**Comprobación de la detección en la página Virustotal[.]com**

El archivo creado anteriormente cómo infected.apk fue detectado por 15 motores de Antivirus el que se muestra en las imágenes posteriores, con esto demostramos que el laboratorio de pruebas se podrá ejecutar de manera exitosa en cualquier versión Linux, demostrando que es muy fácil poder crear un archivo malicioso malware.

**Ilustración 10, test de aplicación en pagina virustotal**

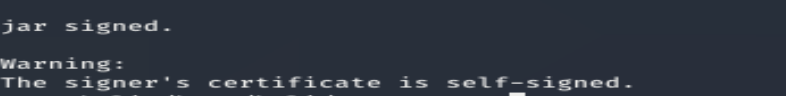
**Se crea almacén de claves con Keytool**

Cuando la aplicación infected.apk se creó de manera exitosa, se debió firmar para poder engañar a cualquier dispositivo y víctima, cómo se muestra en la imagen dieciséis, con el comando keytool ejecutado en Linux, se puede crear un almacén de claves para la siguiente acción que se realizara, sin este almacén no se podría firmar la aplicación, se utiliza el comando keytool agregando el nombre del almacén en este caso tesis.keystore, con un alisas nombrado cómo tesis, algoritmo utilizada RSA un tamaño de 2048 bits, y validación de 10000, cómo se muestra en ilustración que prosigue.

***Ilustración 11, comando creación claves de certificado***

**Firma el archivo con Jarsigner**

Ahora utilizando el almacén creado anteriormente con la aplicación keytool se procede a firmar la aplicación modificada, con el fin de no ser detectada cuando se instale, en este caso se deben pasar los parámetros alias de la ilustración dieciséis, se puede apreciar que este se realizó correctamente donde en la imagen se aprecia jar signed, que significa que la aplicación firmada.



**Ilustración 12, firma de apk**

Finalmente cómo se explicaba al principio de este punto, el fin objetivo fue la selección de sistemas para los siguientes laboratorios que se utilizaran en esta tesis, el cual se utilizó cualquier sistema base Linux, por eso se refuerza la utilización de este sistema operativo para pruebas, el tener una facilidad de utilización de cualquier herramienta, ejecución, entorno, rapidez, entre otros aspectos fundamentales para ser seleccionado para el entorno de pruebas.

5.2. Laboratorio de extracción de metadatos en aplicaciones Android

Este laboratorio es para la selección de metadatos para la identificación del malware de aplicaciones Android, para poder crear un conjunto de datos que puedan ser utilizados por un algoritmo ó por algoritmos de aprendizaje máquina. Además, se incluye la búsqueda de repositorios con aplicaciones malignas y benignas, las que se buscaron en distintas fuentes de internet. En este numeral se explican los pasos precisos que son utilizadas y realizados en la investigación, estos son de gran aporte para el éxito de la investigación, utilizando el software base Linux, que fue seleccionada en el punto anterior cómo sistema operativo base de pruebas y ejecución.

5.2.1 Extracción de fuentes aplicaciones Android malignas y benignas

Para desarrollar pruebas primarias para esta tesis se debió extraer datos de diferentes fuentes para extraer aplicaciones malignas y benignas, según el siguiente detalle:

**Fuentes benignas:** La cantidad de 74 aplicaciones descargadas de la página **www[.]apkpure[.]com**

**Fuentes malignas:** La cantidad de 74 aplicaciones descargas de la fuente **www[.]virusshare[.]com**

5.2.2 Clasificación de aplicaciones

Se testeo las aplicaciones en la página www[.]virustotal[.]com para verificar su estructura para clasificarlas cómo malware ó benigna.

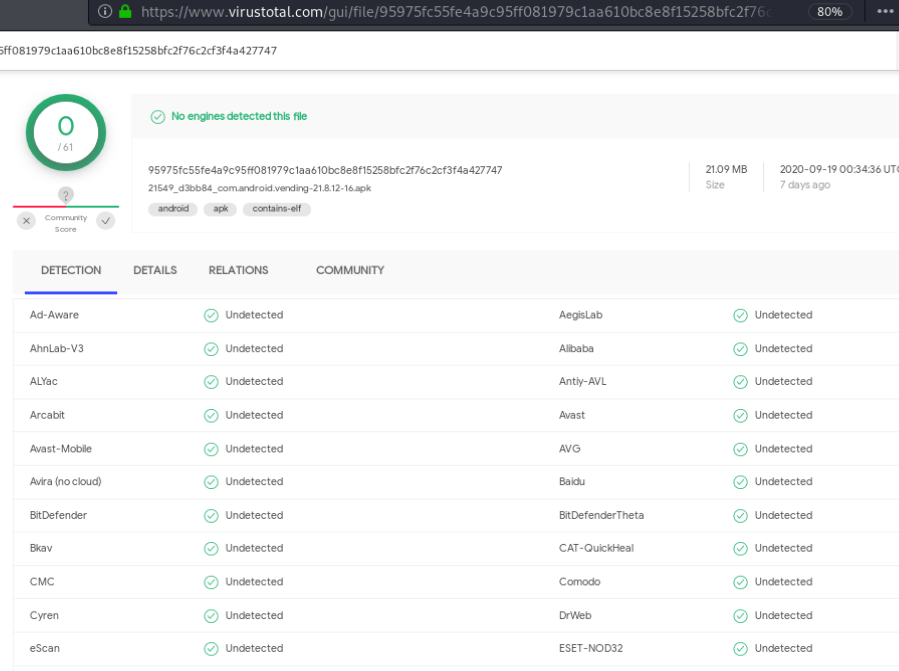
**Muestra de aplicación maligna:** Mediante la página web **www[.]virustotal[.]com** se revisa la estructura de la aplicaciones malignas, para verificar su carga y detección cómo aplicación maliciosa, en la imagen de muestra se verifica la carga maliciosa detectada en varios motores de Antivirus cómo **(Cyren, Ikarus, Microsoft, Truslook, ESET-NOD32, K7GW, Symantec Mobile Insight),** para poder catalogarla cómo malware en el directorio de Linux /APK del seleccionador de metadatos en aplicaciones Android.

En la imagen se puede apreciar que una aplicación del dataset es detectada por varios motores antivirus.



**Ilustración 13, test de pruebas dataset de aplicaciones malignas en pagina virustotal**

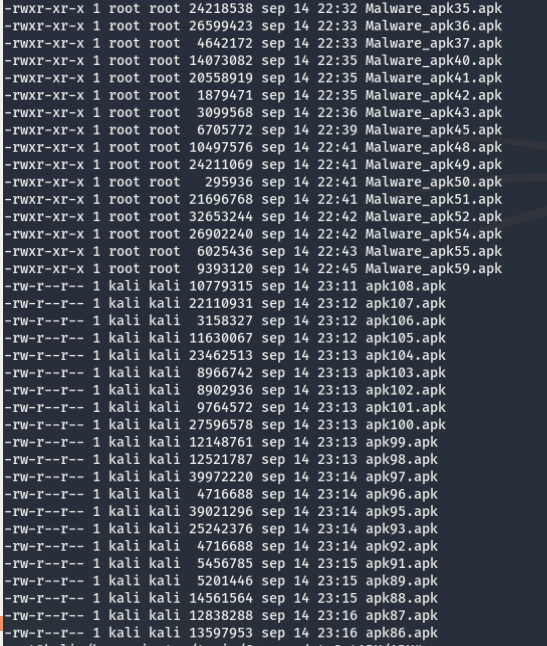
**Muestra de aplicación benigna:** Mediante la página web **www[.]virustotal[.]com** se revisa la estructura de la aplicaciones benignas que fueron descargadas del portal **www[.]apkpure[.]com**, para verificar su carga y detección cómo aplicación maliciosa, en la imagen de muestra se verifica la carga maliciosa NO detectada en todas las aplicaciones de este tipo, para poder catalogarla cómo aplicaciones benignas en el directorio de Linux **/APK** del seleccionador de metadatos aplicaciones python.



**Ilustración 14, test de pruebas base dataset aplicaciones benignas**

Las 148 aplicaciones se dejan en un repositorio Filesystem Linux y se renombran cómo **Malware\_apk\*** las malignas y las benignas cómo **apk\*.** Quedando en el repositorio **APK.**

**Directorio APK:** Este directorio de aplicaciones posee el conjunto de **148 apk’s (74 malignas y 74 benignas)** para sustraer y seleccionar metadatos de malware indicadas creando el archivo (.cvs) que será analizado en la aplicación Orange, estas fueron renombradas para que se puedan ir revisar mediante el script python **CreateDataset.py**



**Ilustración 15, muestra de archivos apk Android malignos y benignos**

5.2.3 Selección de metadatos

Está selección de metadatos se realizó conforme a las vulnerabilidades más recurrentes que se encuentran en las aplicaciones malware, las que son detectadas en las aplicaciones malignas en el dataset sustraído en el sitio **www[.]virusshare[.]com,** estos metadatos se seleccionaron de otras investigaciones (XuJiang, 2020) (Xin Su, 2020) (Zhuo Ma, 2020), cuyas investigaciones se basaron en la extracción de permisos peligrosos, permisos requeridos, localización de códigos peligrosos, dándole la forma de cómo seleccionar varios metadatos del archivo manifiesto (AndroidManifiest.xml) de las aplicaciones APK, para generar un dataset que se pueda validar con varios algoritmos de detección malware Android, utilizando Aprendizaje automático, Deep Learning, y cualquier técnica de análisis estático.

Se seleccionaron los metadatos de las aplicaciones (APK) para extraer y crear un dataset, conforme al siguiente recuadro resultado del mapeo sistemático:

**Tabla 17, metadatos seleccionadas para detección malware**

|  |  |  |
| --- | --- | --- |
| **Nro.** | **Archivo** | **Característica** |
| 1 | AndroidManifiest.xml | Permisos (Cantidad de 367 permisos leídos en el archivo de salida) |
| 2 | AndroidManifiest.xml | SDK <= 18 (Ultima versión estable versión 19) |
| 3 | AndroidManifiest.xml | MainActivity(Parámetro vulnerable a inyección de código) |
| 4 | AndroidManifiest.xml | Receivers (Receptor de transmisión vulnerable) |

* **Descripción de las metadatos extraídos y seleccionadas para el dataset**

**Permisos:** Los permisos extraídos son comparados con los contenidos en las aplicaciones malware, estas aplicaciones al contener permisos peligrosos tienen un alto riesgo de ser aplicaciones vulneradas, para esto se necesita la totalidad de los permisos que tienen las aplicaciones malware para hacer una comparación con las benignas, así pudimos estrenar el algoritmo seleccionado, posteriormente se pudo ingresar cualquier aplicación que se desconozca su resultado y con este entrenamiento poder entregar un resultado aproximado. La cantidad utilizada de permisos para esta característica es un total de 367 permisos que se encuentran en el dataset archivo de salida. Estos permisos contienen el nivel desde los más peligrosos hasta los que no mantienen ningún tipo de riesgo, se presentaron y se seleccionaron las siguientes metadatos:

**SDK <= 18:** Los dispositivos con plataformas anteriores a Android 4.4 (API nivel 19) utilizan una versión de webkit que presentan problemas de seguridad. La solución alternativa, si la aplicación se ejecuta en estos dispositivos se debe confirmar que los objetos WebView solo muestren contenido de confianza. Para asegurarte de que la aplicación no esté expuesta a posibles vulnerabilidades de SSL (secure sockets layer ó seguridad capa de transporte), usa el objeto de seguridad Provider actualizable, para actualizar el proveedor de seguridad con el fin de protegerse contra vulnerabilidades de SSL. Si la aplicación debe procesar contenido de la Web abierta, se debe proporcionar al propio procesador para poder mantenerlo actualizado con los últimos parches de seguridad (Developers Android, 2020).

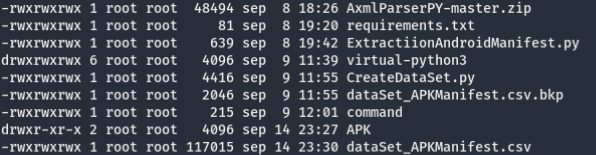
**MainActivity:** Este metadatodel manifiesto de aplicaciones Android, es que posee una declaración de actividad, si está declarada cómo MainActivity, es más fácil a un usuario mal intencionado inyectarle código malicioso, en los archivos de directorio encontrando con extensión “.smali”, tal como se muestran en item anteriores, encontrando en el archivo smali el método onCreate(), línea del código donde se le inyectara la invocación a un método (Developers Android, 2020).

**Receivers:** Este metadato del manifiesto de aplicaciones Android, se encuentra receivers que es un receptor de transmisión, estos se registran para que ocurran eventos específicos, cuando ocurre dicho eventos, se invoca al receptor y realiza tareas, como por ejemplo mostrar un mensaje al usuario, que este puede ser utilizado por usuarios mal intencionados para realizar consultas no autorizadas (Srinivas, 2014).

5.2.4 Sistema Scripting Pyhton para extracción de metadatos

Se desarrollo un sistema scripting Python para la extracción de metadatos seleccionados, para luego crear un dataset en un archivo (CSV), con la siguiente estructura de repositorio. (jelvez, 2021)

La siguiente estructura del sistema scripting python cumple diferentes funciones, en las cuales tiene un archivo de ejecución CreateDataset.py que invoca a diferentes librerías de Pyhton para extraer del archivo AndroidManifiest.xml los metadatos seleccionas anteriormente, el cual después de seleccionarlas crear un archivo dataSet\_APKManifest.csv, para poder realizar el análisis en la aplicación Orange. Se muestra en la siguiente imagen la estructura de archivos de este script Python.



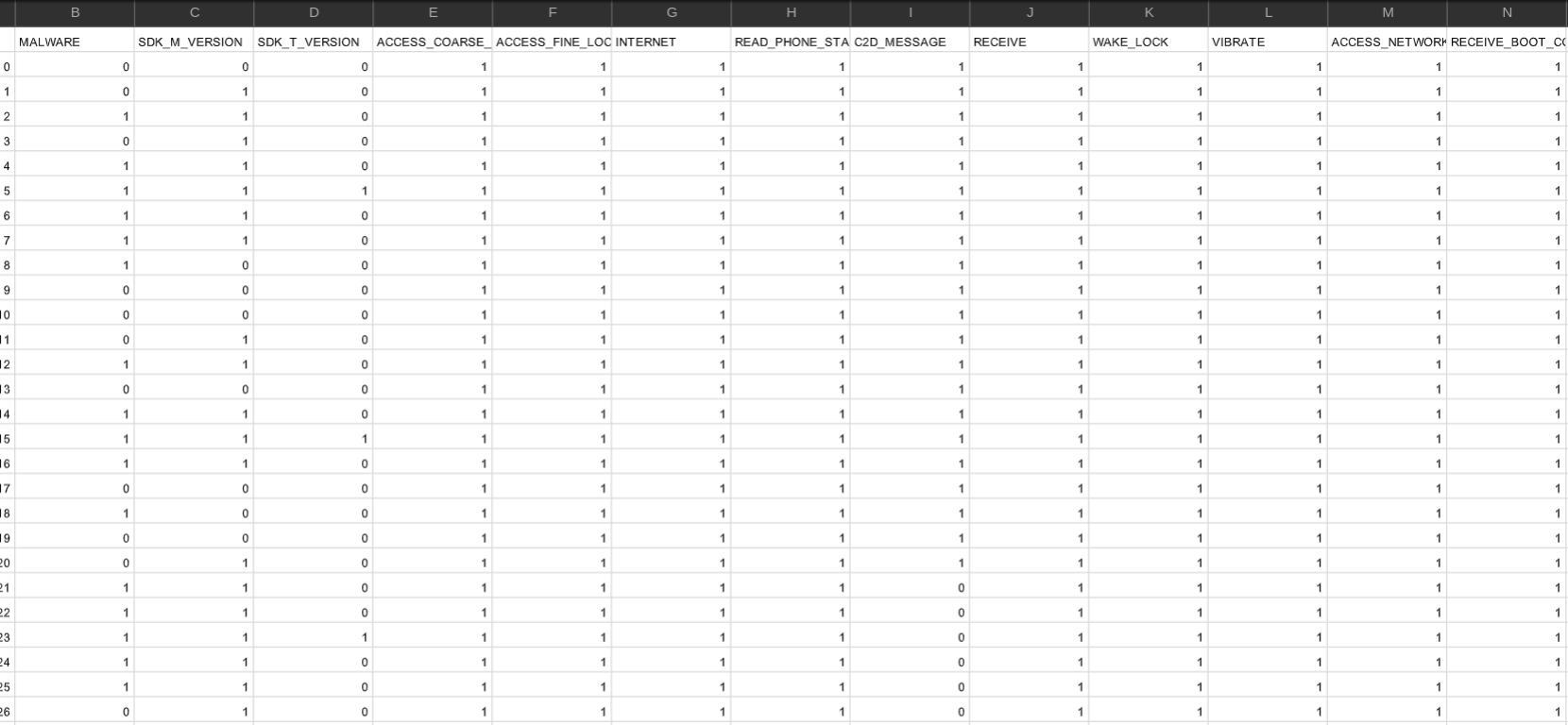
**Ilustración 16, estructura de archivos sistema extractor de metadatos**

**Estructura de archivos sistema scripting python:** La siguiente estructura de archivos es del repositorio del sistema python, el que sirve para la extracción de metadatos malware para Android, para lo cual se describe cada una de las funciones de los archivos en la siguiente tabla:

**Tabla 184, descripción script sistema extractor de metadatos**

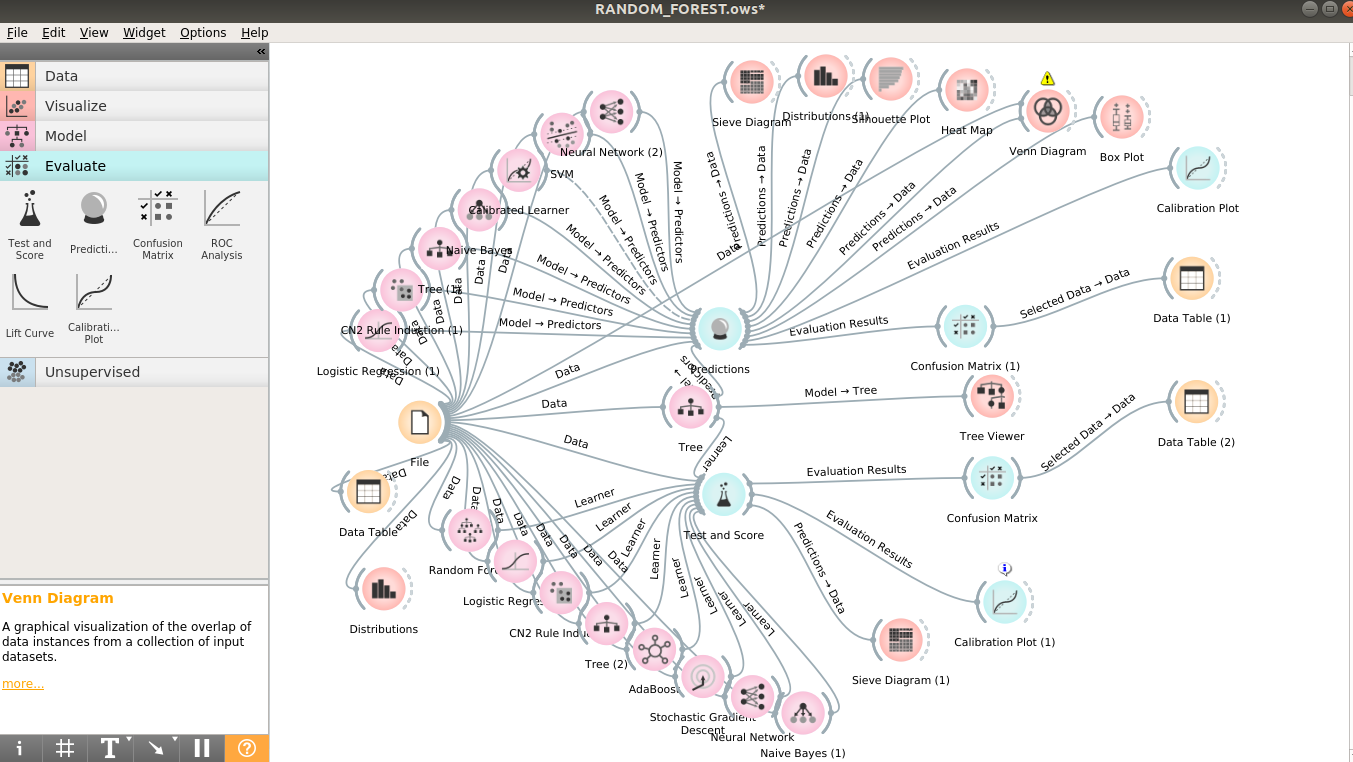
|  |  |  |
| --- | --- | --- |
| **Nro.** | **Archivo/Directorio** | **Característica** |
| 1 | AxmlParserPY-master.zip | Analizar archivos XML |
| 2 | requirements.txt | Requerimientos de instalación |
| 3 | ExtractiionAndroidManifest.py | Archivo de ejemplo |
| 4 | Virtual-python3 | Ambiente virtual |
| 5 | CreateDataset.py | Script principal |
| 6 | dataSet\_APKManifiest.csv.bkp | dataset respaldo |
| 7 | Command | Comando de ejemplo |
| 8 | APK | Directorios de APK’s |
| 9 | dataSet\_APKManifest.csv | archivo de salida |

**Estructura de archivo de salida del sistema scripting python:** Se adjunta imagen con un espacio del archivo dataSet\_APKManifest.csv (CSV) de salida, el que tiene la siguiente nomenclatura, si tiene la característica marcada escrita con el número uno en la celda del CSV., cómo se percibe en el extracto significa que si posee esa característica la aplicación Android. Para poder explicar de mejor manera este ejemplo podemos decir que el archivo de ejemplo seria predispuesto a tener metadatos de malware si el SDK es <= 18, MainActivity, y los permisos que tiene la aplicación, marcándose la celda con el número uno, todo esto aporta para poder crear un sistema de detección de malware Android, basado en Aprendizaje automático, el que será presentado en los puntos posteriores, cada línea de registro es una aplicación ingresada a la matriz del sistema de aprendizaje automático, entre más registros se obtiene mayor resultado de precisión. Lo ideal que este sistema este estrenado con muchos datos sobre los mil registros y cada vez que una aplicación pase por el algoritmo este detecte que metadatos fueron seleccionadas como preponderantes en las aplicaciones malignas. Para esto se presentó el siguiente archivo de ejemplo con sus estructura de salida del sistema scripting python.

***Ilustración 17, muestra archivo salida extracción metadatos***

5.2.5 Selección del algoritmo de aprendizaje automático para detección de malware

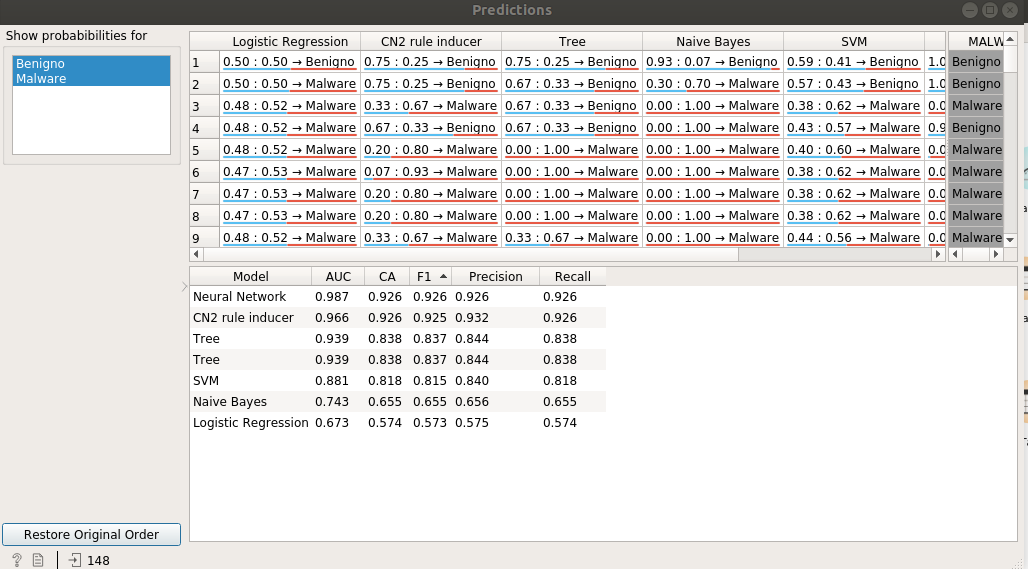
Se utiliza la herramienta Orange (Devashree Vaishnav, 2018) para realizar un modelado con todos los algoritmos y probar la precisión de estos. En esta imagen se pueden apreciar cada uno de los Algoritmos que fueron utilizados para realizar distintas pruebas y poder verificar el que más precisión tienen ante el conjunto de datos, esta imagen representa la cantidad de pruebas realizadas para llegar al validar el algoritmo con mayor precisión.



**Ilustración 18, test completo de pruebas algoritmos aplicación Orange**

5.2.6 Resultados de las pruebas con las predicciones

En esta prueba con el conjunto de datos acotado arrojó los resultados precisos sin editar la configuración de los algoritmos, donde obtuvieron más precisión los algoritmos **Neural Network (0.987), CN2 RULE inducer (0.966), Tree (0.939), SVM (0.881), y Naive Bayes (0.743),** los que no se realiza ajuste alguno en su configuraciones, obteniendo los resultados de la imagen que se adjunta.



**Ilustración 19, resultado de pruebas predictions**

5.2.7 Resultados previos de algoritmos para pruebas iníciales

**Matriz de confusión y predicción**

En la ilustración veinte se puede apreciar que tiene resultados de seis algoritmos, para poder elegir los dos que menor taza de fallos tengan, primero se demostraran los resultados en la siguiente descripción por algoritmos:

**Algoritmo Logistic Regression:** En la ilustración se puede apreciar el resultado de verdadero positivo 56.6%, verdaderos negativos 58.5%, falsos positivos 41.5% y falsos negativos 43.4%.

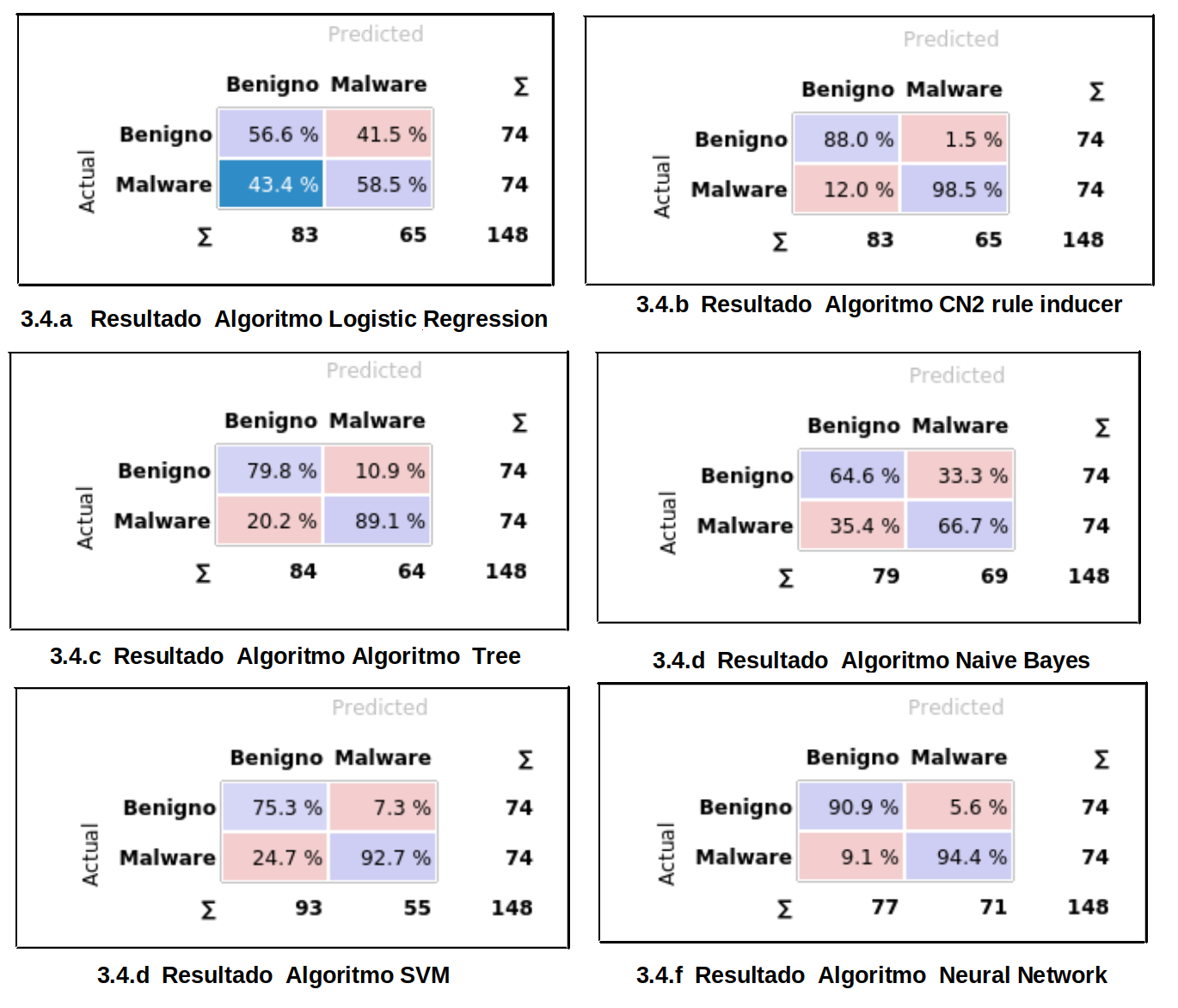
**Algoritmo CN2 rule inducer:** En la ilustración se puede apreciar el resultado de verdadero positivo 88.0%, verdaderos negativos 98.5%, falsos positivos 1.5% y falsos negativos 12.0%.

**Algoritmo Tree:** En la ilustración se puede apreciar el resultado de verdadero positivo 79.8%, verdaderos negativos 89.1%, falsos positivos 10.9% y falsos negativos 20.2%.

**Algoritmo Naive Bayes:** En la ilustración se puede apreciar el resultado de verdadero positivo 64.6%, verdaderos negativos 66.7%, falsos positivos 35.4% y falsos negativos 33.3%.

**Algoritmo SVM:** En la ilustración se puede apreciar el resultado de verdadero positivo 75.3%, verdaderos negativos 92.7%, falsos positivos 7.3% y falsos negativos 24.7%.

**Algoritmo Neural Network:** En la ilustración se puede apreciar el resultado de verdadero positivo 90.9%, verdaderos negativos 94.4%, falsos positivos 5.6% y falsos negativos 9.1%.

***Ilustración 20, resultado completo de pruebas predicción representado en matrices de***

***Confusión***

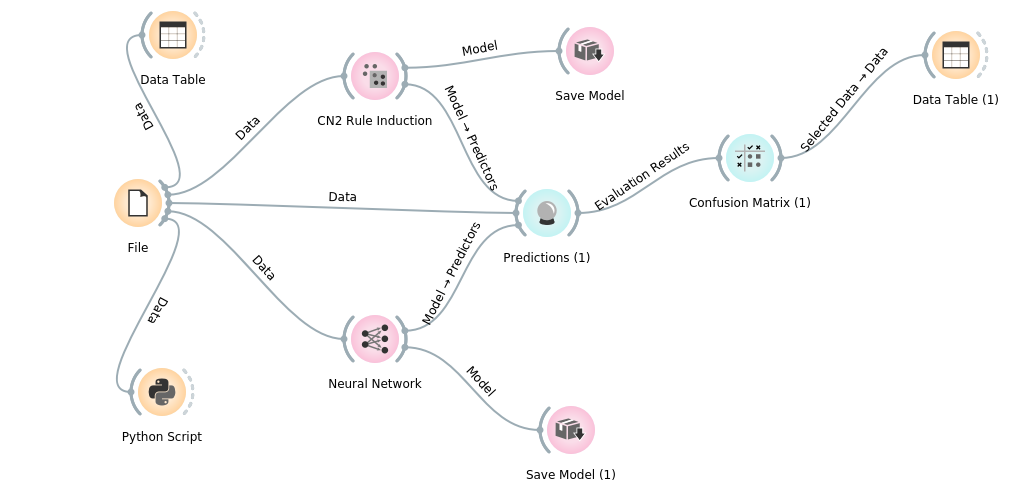
**Resumen:** Los datos revisados en las Matrices de confusión en los algoritmos ***(Logistic Regression, CN2 rule inducer, Tree, Naive Bayes, SVM, Neural Network),*** arrojaron como más exactos ***CN2 rule inducer y Neural Network***, representado en la ilustración veinte con una taza de ***fallos del 12.0% - 1.5%***

***Perteneciendo al algoritmo CN2 rule inducer*** y ***con un 9.1% - 5.6% perteneciendo al algoritmo Neural Network,*** cómo se representa en la imagen descriptiva adjunta para mayor descripción y entendimiento.

5.2.8 Selección de algoritmo con mayor precisión

* **Diagrama Neural Network y CN2 Induction**

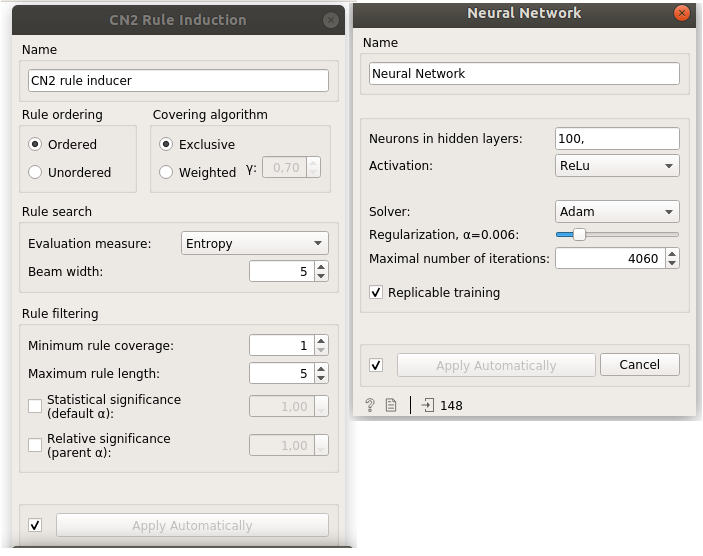
Después de la revisión de distintos algoritmos, se seleccionó los algoritmos con mayor precisión para las pruebas del dataset, creando el último modelo en Orange3 con los algoritmos seleccionados (CN2 Rule Induction, Neural Network) realizando ajustes a la configuración de cada algoritmo para alcanzar la máxima precisión.

****

**Ilustración 21, diagrama final Orange**

Este diagrama final es la selección de algoritmo con mayor aproximación realizada obtenida en el punto 5.2.5, seleccionando los algoritmos CN2 Rule Induction y Neural Network, se destacan los objetos del diagrama cómo input de datos (File), el tipo de selector (Predictions (1)), la matriz de evaluación (Confusion Matrix (1), para guardar el modelo (Save Model), tablas de visualización (Data Table), y para poder concatenar algún script de código (Python Script).

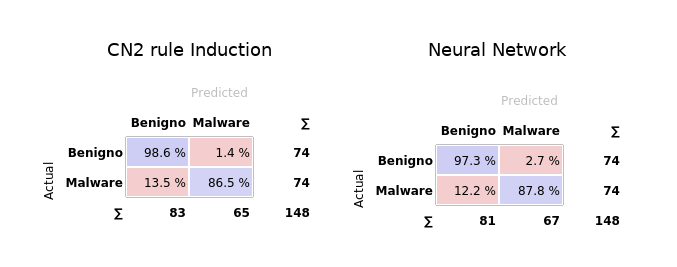
Se modifico los parámetros de la **Neural Network** dejando al solver Adam, con número de neuronas en capas ocultas 100, Activación Relu, y el número máximo de iteraciones 4060, el cual llegó a obtener una mayor precisión, y el otro algoritmo CN2 Rule Induction se movieron diferentes parámetros no modificando su precisión.



**Ilustración 22, configuración algoritmo seleccionados**

* **Matriz de confusión**

CN2 Rule Induction y Neural Network son los algoritmos con mayor precisión, se aprecia una falla de 13.5% y 1.4% de CN2 rule Induction y de un 12.2% y 2.7% de la Neural Network en el análisis estático.

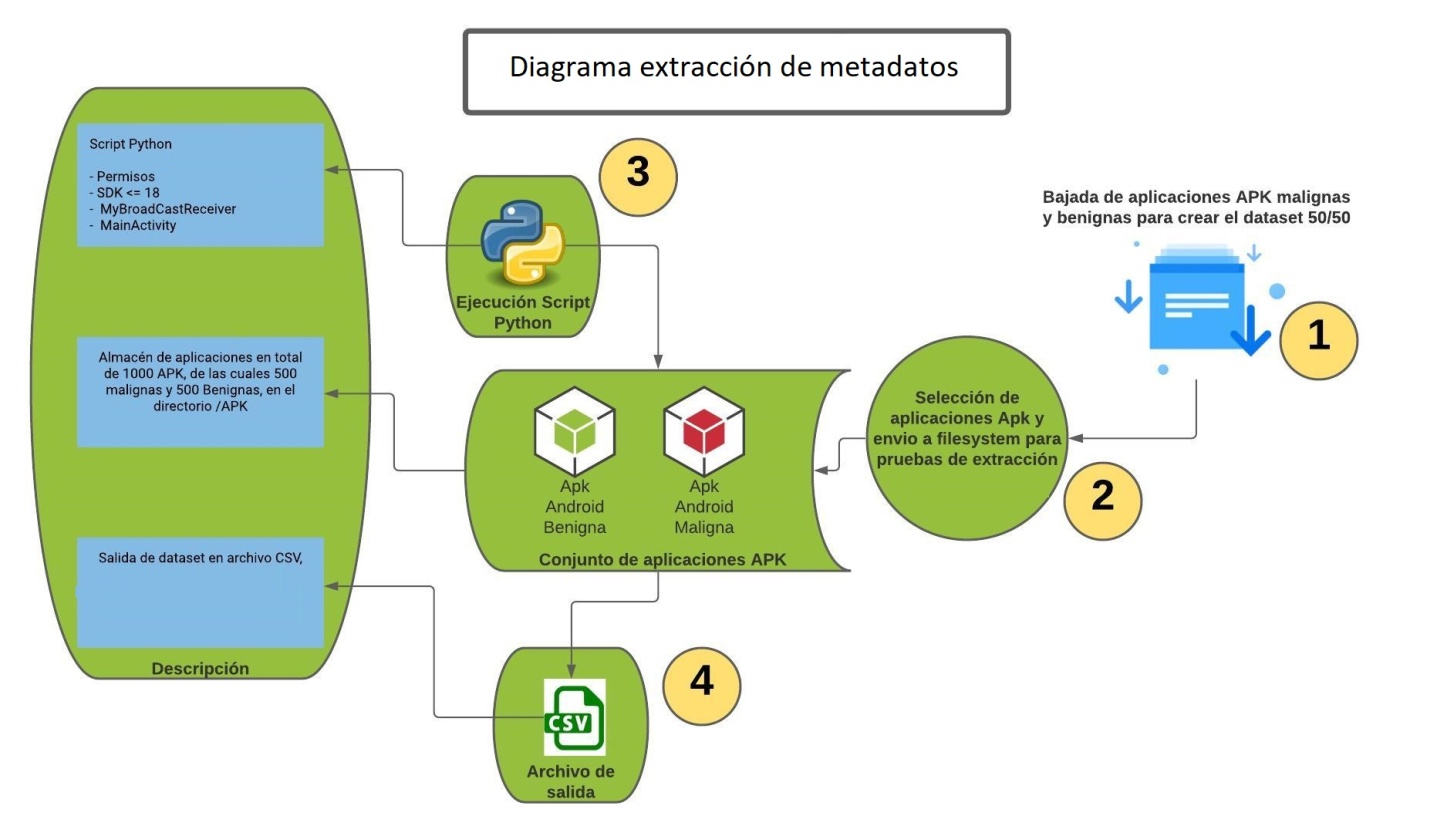
****

**Ilustración 23, resultado CN2 rule induction y Neural Network**

5.3. Construcción de método para detección de Malware

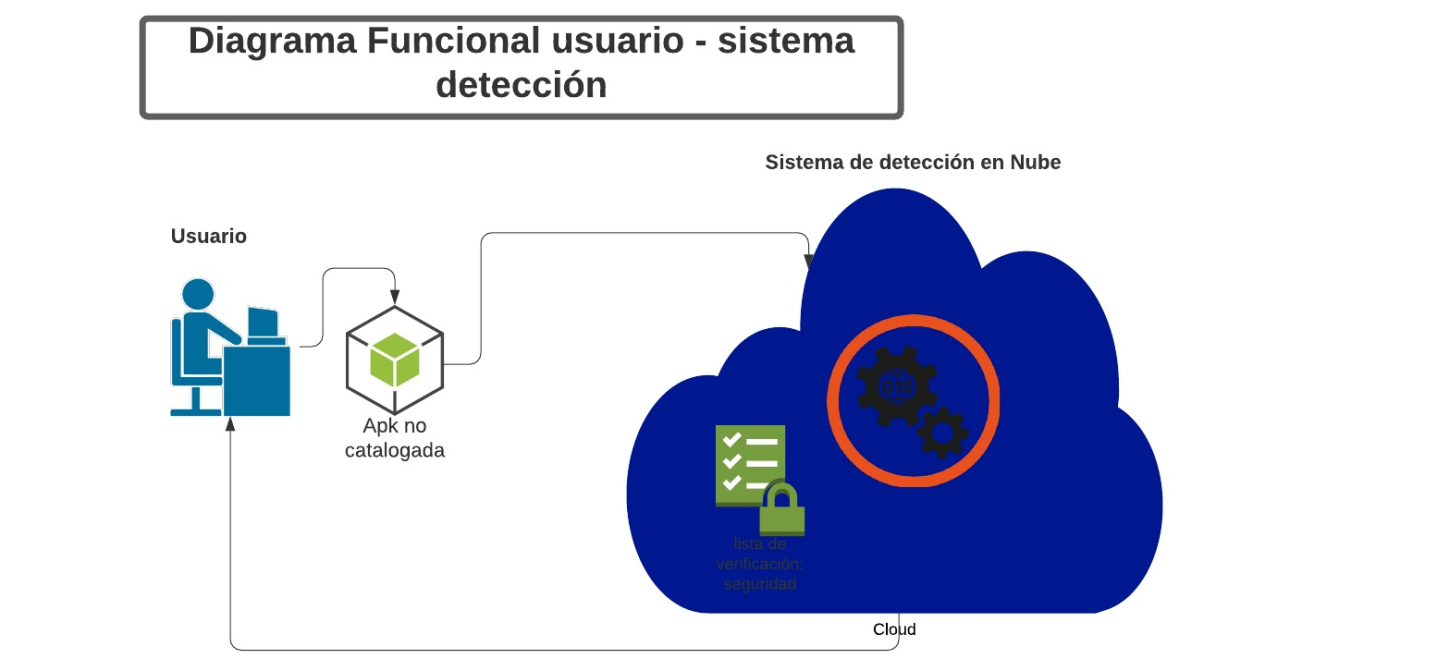
Los algoritmos de aprendizaje automático supervisados requieren de una fase de entrenamiento con muchas muestras para que el modelo diseñado pueda aprender de forma óptima y precisa, en este caso la detección de malware con variables estáticas. Este proceso se realizó de forma manual, previamente se tuvo identificadas estos metadatos estáticos de aplicaciones Android (APK) que ayudan a la identificación y/o detección de una aplicación malware, destacamos los pasos con los cuales se realizó el entrenamiento del modelo de Neural Network, los que se encuentran a continuación.

* Descarga muestras de aplicaciones malignas y benignas.
* Selección de aplicaciones APK malignas y benignas, descarte de otros formatos
* Envió de archivos al Filesystem APK.
* Ejecución del script Python para extracción de metadatos, permisos, MainActivity, SDK y receiver.
* Construcción del dataset en formato csv, con datos en 0 si la característica no está y con 1 si esta característica se encuentra en la aplicación.

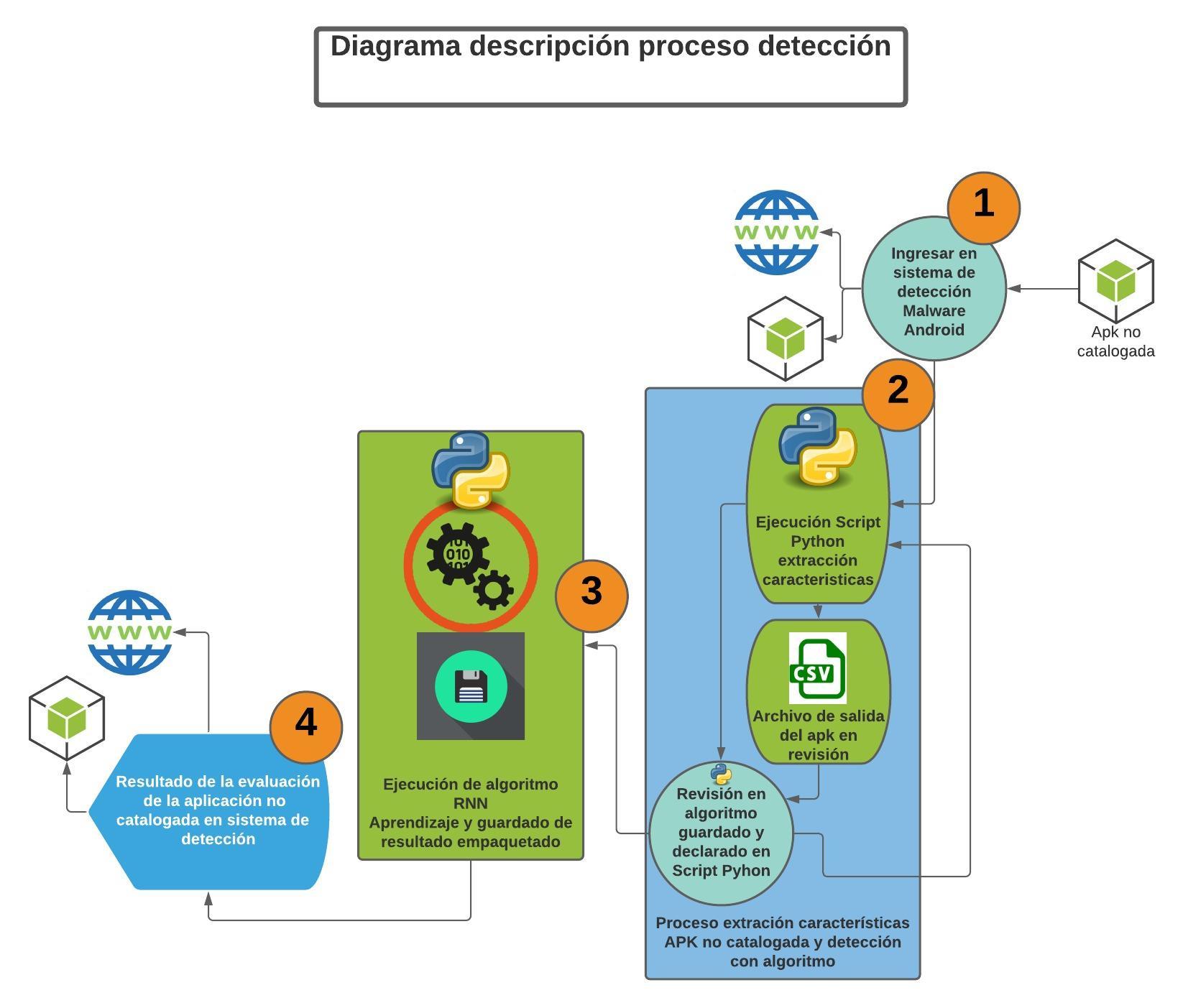


**Ilustración 24, diagrama sistema extracción de metadatos**

* Posterior al entrenamiento y obtener una presión aceptable para detectar malware en APK se procede a empaquetar el modelo entrenado.
* Una vez guardado este modelo se utilizó el mismo código del proceso 1), de la imagen, para extraer los metadatos de la siguiente aplicación que se debe analizar dejando ese registro en un archivo csv, para hacer la comparación con el modelo que logró aprender, en este proceso se podría crear una aplicación web ó una apk de Android que este corriendo en una Nube, con todos los protocolos de seguridad para que cualquier usuario, desarrollador, entidad, empresa, gobierno, etc., pueda subir sus aplicaciones al sistema y este entregue un resultado mostrando si esta aplicación tiene metadatos de malware, para esto se presenta este diagrama funcional donde se integra la Red Neuronal guardada en el primer paso y la ejecución de revisión de una aplicación APK no catalogada, igualmente en las empresas se podría integrar este sistema de detección a un pipeline de desarrollo con una aplicación que este analizando las aplicaciones Android para evitar desarrollos inseguros.



**Ilustración 25, diagrama funcional sistema detección**



**Ilustración 26, diagrama funcional sistema detección**

La Red Neuronal es ejecutada de manera local en una aplicación, todo este entorno es corrido bajo Linux, el algoritmo es codificado y se realiza la siguiente explicación de sus líneas.

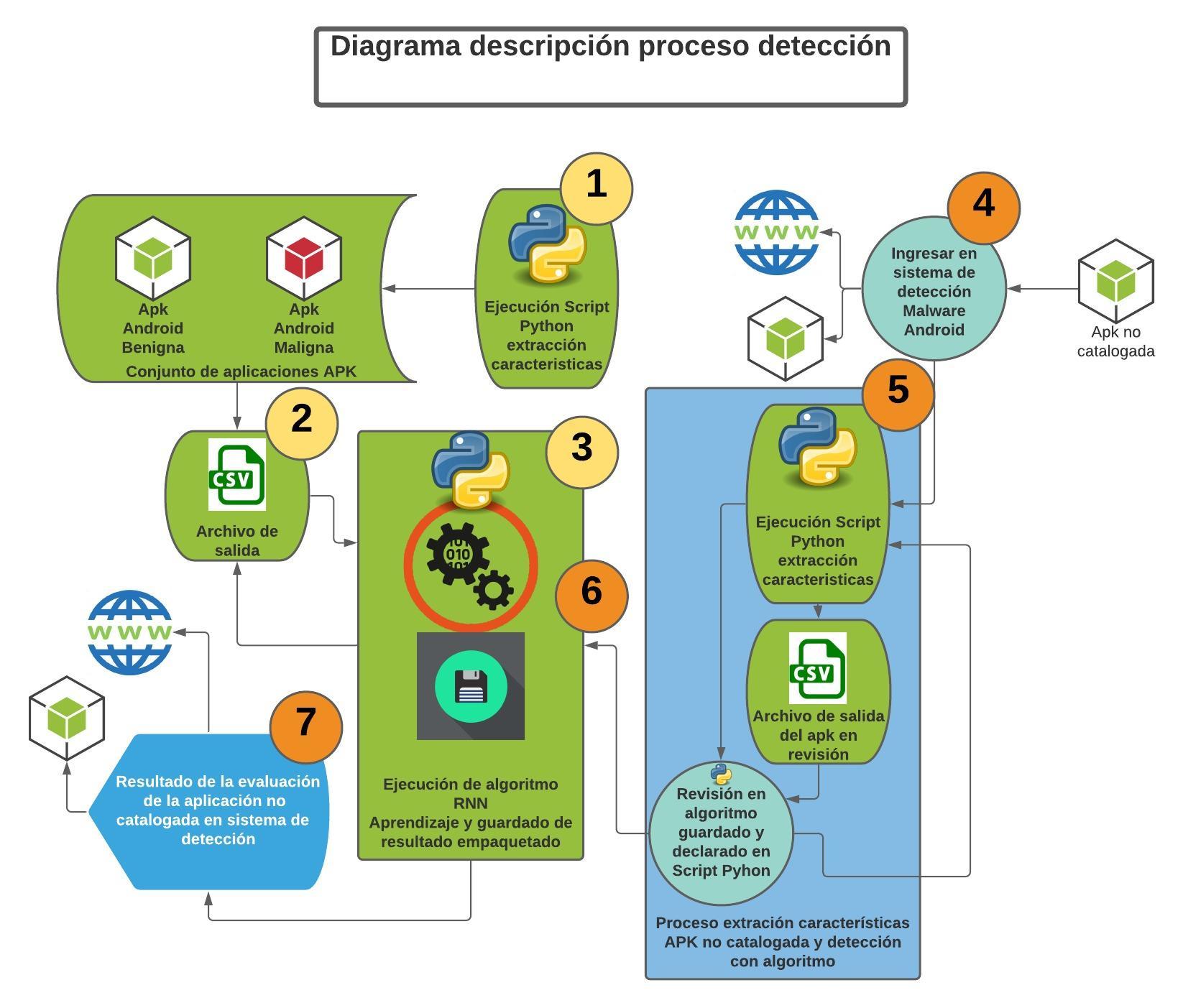
En la siguiente tabla se explica el código completo del algoritmo Red Neuronal utilizado, donde se explica cada línea en detalle del código, para que cualquier futuro trabajo se puedan integrar este algoritmo en cualquier sistema de extracción de metadatos, y detección de malware Android (Jelvez, Github, 2021).

**Tabla 19, descripción código algoritmo Red Neuronal**

|  |  |  |
| --- | --- | --- |
| **Línea** | **Código** | **Descripción** |
| 1 | Import tensorflow as tf | Carga de módulo Tensorflow, este es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, y desarrollado por Google para satisfacer sus necesidades de sistemas capaces de construir, entrenar redes neuronales para detectar, descifrando patrones, correlaciones, análogos al aprendizaje y razonamiento usados por los humanos |
| 2 | From tensorflow import keras | Importar módulo keras desde tensorflow, es una biblioteca de Redes Neuronales de Código Abierto escrita en Python. Es capaz de ejecutarse sobre TensorFlow, Microsoft CognitiveToolkit ó Theano. |
| 3 | Import numpy as np | Carga del módulo numpy, esta es una biblioteca para el lenguaje de programación Python que da soporte para crear vectores y matrices grandes multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel para operar con ellas. |
| 4 | import pandas | Carga de módulo Pandas, es una biblioteca de software escrita cómo extensión de NumPy para manipulación y análisis de datos para el lenguaje de programación Python. |
| 5 | From keras.models import Sequential | Carga de módulo Sequential desde keras, este es modelo es apropiado para una pila simple de capas donde cada capa tiene exactamente un tensor de entrada y un tensor de salida. |
| 6 | From keras.layers import Dense | Carga de módulo Dense desde keras.layers, este es la capa regular de la red neuronal profundamente conectada. Es la capa más común y de uso frecuente. La capa densa realiza la siguiente operación en la entrada y devuelve la salida. |
| 7 | Dataframe = pandas.read\_csv  ('/home/wilinosx/Escritorio  /Tesis/dataset/  dataSet\_APKManifest\_147\_0-1.csv') | Variable dataframe, con llamado a la función pandas.read\_csv(este concatena la librería pandas y función read de un archivo csv) la cual tiene el string de la ubicación del dataset anteriormente desarrollado. |
| 8 | RawData = dataframe.values | Se declara variable rawData para agregar la anterior variable con los valores del dataset |
| 9 | header=dataframe.head() | Se declara variable header, para llamar a la variable dataframe con el encabezado del archivo. |
| 10 | features = rawData[:,2:] | Se declara variable features, para llamar a la variable rawData con los valores separados en coma, desde la columna 2. |
| 11 | labels = rawData[:,1] | Se declara variable features, para llamar a la variable rawData con los valores separados en coma, desde la columna 2. |
| 12 | features=np.array  (features,dtype=float) | Se declara la variable features, para llamar np.array, con las características de dtype cómo campos flotantes. |
| 13 | model = Sequential() | Se declara variable model cómo sequencial, la que está declarada cómo módulo importado en la cabecera del script. |
| 14 | model.add(Dense(12, input\_dim=370, kernel\_initializer='uniform', activation='relu')) | A la variable model se le agrega a la capa densa 12 capas, un imput\_dim (número de dimensiones de las entidades). kernel\_initializer='uniform' cómo iniciador de capa uniforme, y activation='relu’ cómo activación del algoritmo cómo relu (rectificador de redes neuronales). |
| 15 | model.add(Dense(8, kernel\_initializer='uniform', activation='relu')) | Se le agrega a la variable model, capa de densa de 8, un iniciador uniforme, y activación CNN relu (Ejemplo de relu Cuando procesamos una imagen, cada capa de convolución debe capturar algún patrón en la imagen y pasarla en la siguiente capa de convolución. Los valores negativos no son importantes en el procesamiento de imágenes y, por lo tanto, se establecen en 0) |
| 16 | model.add(Dense(1, kernel\_initializer='uniform', activation='sigmoid')) | Se le agrega a la variable model, capa densa de 1, kernel\_initializer uniforme, la función de activación sigmoid (función matemática buena para representar una probabilidad, su dominio son todos los números reales, pero su rango es de 0 a 1.) |
| 17 | model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['precisión']) | Se agrega a la variable model.compile, la compilación del algoritmo CNN, cómo loss='binary\_crossentropy' (El propósito de las funciones de pérdida es calcular la cantidad que un modelo debe buscar minimizar durante el entrenamiento). Con el optimizador adam para compilar el algoritmo CNN adam (La optimización de Adam es un método de descenso de gradiente estocástico que se basa en la estimación adaptativa de momentos de primer y segundo orden). Finalmente, con (metrics=['precisión'])), Las funciones métricas son similares a las funciones de pérdida, excepto que los resultados de evaluar una métrica no se utilizan al entrenar el modelo. |
| 18 | model.fit(features, labels, epochs=4000, batch\_size=10) | A la variable model se le agrega el fit cómo ajuste, con las características, campos, y una cantidad de 4000 épocas, repeticiones que se realizarán para llegar a un óptimo del aprendizaje, finalmente un batch\_size de 10 (Ejemplo, supongamos que tiene 1050 muestras de entrenamiento y desea configurar un valor batch\_size igual a 100. El algoritmo toma las primeras 100 muestras (del 1 al 100) del conjunto de datos de entrenamiento y entrena la red) |
| 19 | scores = model.evaluate(features, labels) | Se agrega variable score para llamar a la variable model y evaluarla con sus características y campos. |
| 20 | print("%s: %.2f%%" % (model.metrics\_names[1], scores[1] \* 100)) | Se imprime el modelo.metrics\_name, puntos 1 x 100 en pantalla |

La descripción de cada línea detallada del algoritmo de Red Neuronal se puede describir primero donde se ingresan las librerías, posteriormente las variables, luego se crea el matriz de datos, y se imprime el resultado. Cómo se describe en este diagrama de proceso.

Finalmente, para el proceso de detección se realizan dos procesos en paralelo, en el cual se pueden unir en un solo diagrama explicativo.



**Ilustración 32, concatenación proceso aprendizaje y detección**

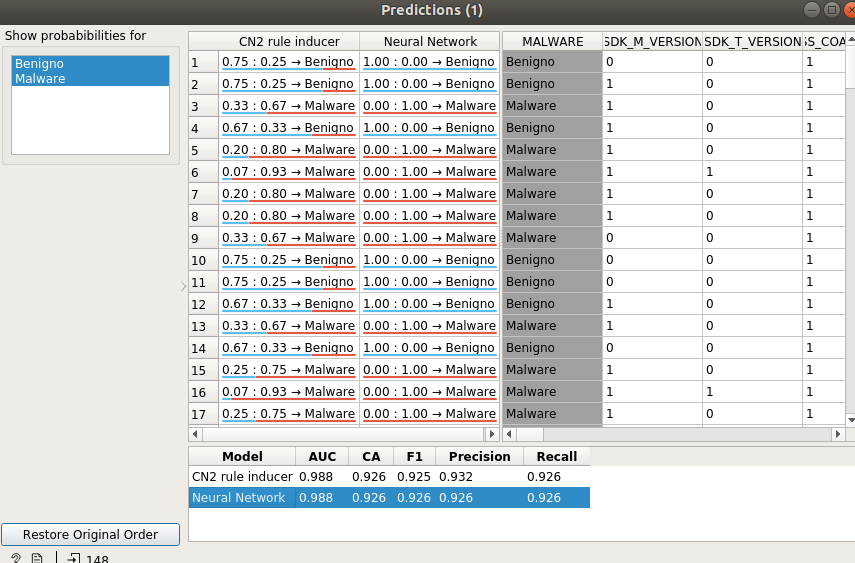
5.4. Evaluación del algoritmo

Se muestra los resultados de distintos análisis con distintos dataset (Jelvez, Github, 2021), para finalmente formar un tipo de conclusión de los resultados, para comparar y verificar la efectividad ó deficiencia de los algoritmos utilizados, en estos resultados se mostraran los resultados de la matriz de confusión precisando la cantidad de registros que fueron evaluados correctamente cómo igualmente los que no está evaluados de manera correcta, por ejemplo los FP que es la abreviatura de los falsos positivos.

Cómo se indicaba para esto se realizó con distintos conjuntos de datos que se extrajeron de dos fuentes que fueron Virusshare y University New Brunswick (UNB), en estas Fuentes se realizado igualmente un filtrado de información de datos seleccionando los archivos de aplicaciones compatibles con Android.

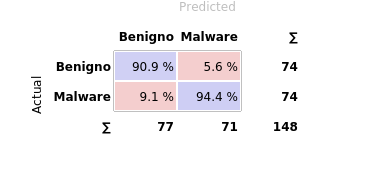
5.4.1 Ítems 148 aplicaciones dataset virusshare

* El algoritmo obtuvo una precisión de 0.988, con el dataset de virusshare con 148 aplicaciones, cómo se observa en la ilustración que se adjunta.



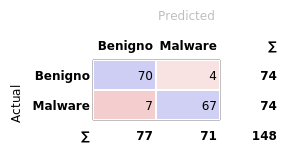
**Ilustración 273, resultado 148 apk dataset virusshare**

* Matriz confusión de 148 registros en porcentaje se demuestra, con resultado de 90.9% cómo verdaderos positivos (VP), 94.4% verdaderos negativos (VN), 9.1% de falsos positivos (FP) y finalmente 5.6% de falsos negativos (FN).



**Ilustración 34, matriz 148 registros porcentaje con dataset virusshare**

* Matriz confusión 148 registros de instancias, con resultado de 70 cómo verdaderos positivos (VP), 67 verdaderos negativos (VN), 7 de falsos positivos (FP) y finalmente 4 con falsos negativos (FN).

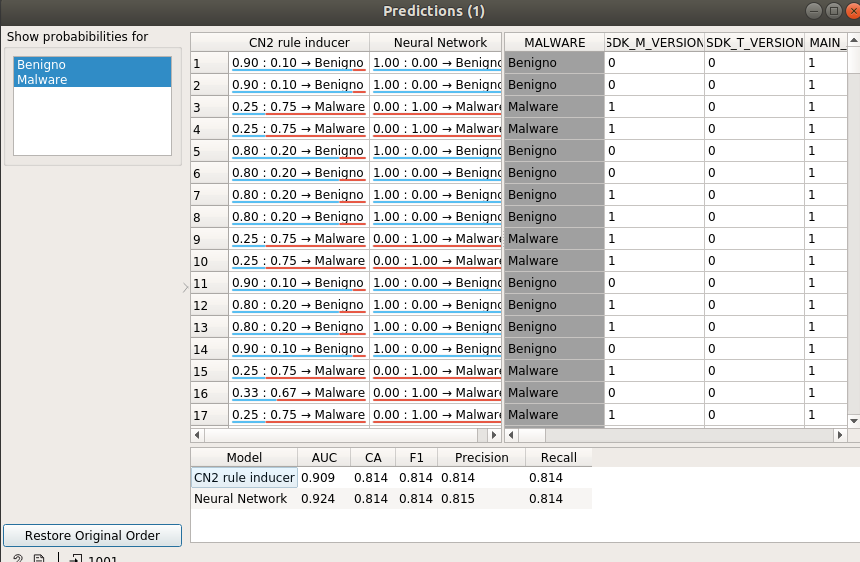


**Ilustración 35, matriz 148 instancias con dataset virusshare**

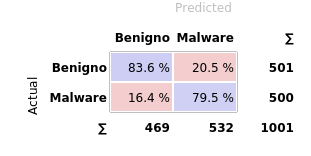
* En la precisión de realiza el cálculo de la suma de verdaderos positivos más verdaderos negativos, dividido por la suma de verdaderos positivos más falsos positivos, falsos negativos y verdaderos negativos, dejando un resultado de 0.926.

**Precisión = (70+67)/(70+7+4+67) = 0.926**

5.4.2 Ítems 1000 aplicaciones dataset virusshare

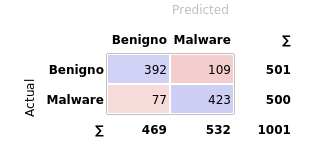
* El algoritmo obtuvo una precisión de 0.924, con el dataset de virusshare de 1000 aplicaciones, cómo se observa en la ilustración que sigue, con el algoritmo seleccionado Neural Network.

**Ilustración 36, resultado precisión 1000 registros virusshare**

* Matriz de confusión porcentajes de 1000 registros de Virusshare, con resultado de 83.6% cómo verdaderos positivos (VP), 79.5% verdaderos negativos (VN), 20.5% de falsos positivos (FP) y finalmente 16.4% de falsos negativos (FN).

**Ilustración 37, matriz confusión porcentaje con dataset virusshare**

* Matriz de confusión instancias de 1000 registros de Virusshare, con resultado de 392 cómo verdaderos positivos (VP), 423 verdaderos negativos (VN), 109 de falsos positivos (FP) y finalmente 77 aplicaciones de falsos negativos (FN).

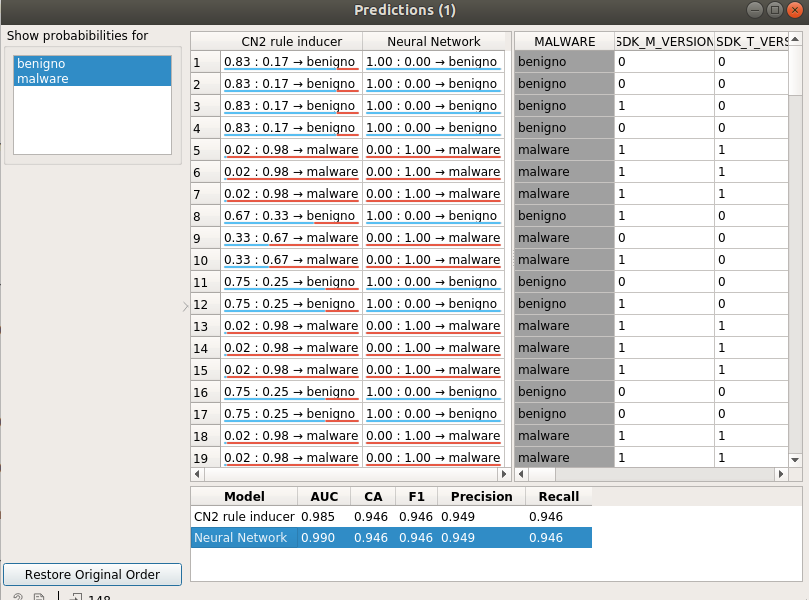


**Ilustración 38, matriz confusión instancias con dataset virusshare**

* En la precisión de realiza el cálculo de la suma de verdaderos positivos más verdaderos negativos, dividido por la suma de verdaderos positivos más falsos positivos, falsos negativos y verdaderos negativos, dejando un resultado de 0.814.

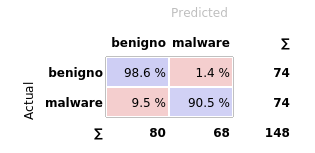
**Precisión = (392+423)/(392+77+109+423) = 0.814**

5.4.3 Item 148 aplicaciones dataset de fuente University New Brunswick(UNB)

Con el dataset UNB (UNB, 2017)el algoritmo seleccionado entregó precisión del 0.990.

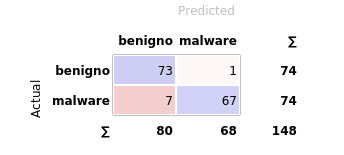
**Ilustración 39, resultado precisión 148 registros UNB**

* Matriz de confusión porcentajes de 148 registros de la fuente UNB, con resultado de 98.6% cómo verdaderos positivos (VP), 90.5% verdaderos negativos (VN), 1.4% de falsos positivos (FP) y finalmente 9.5% de falsos negativos (FN).



**Ilustración 40, matriz confusión porcentaje con dataset UNB**

* Matriz de confusión con instancias de 148 por fuente UNB, con resultado de 73 cómo verdaderos positivos (VP), 63 verdaderos negativos (VN), 1 de falsos positivos (FP) y finalmente 7 de falsos negativos (FN).



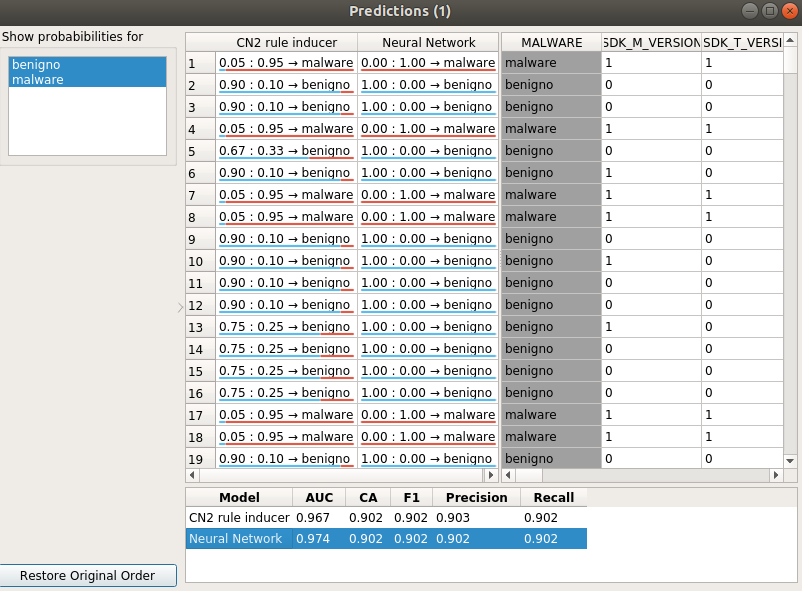
**Ilustración 41, matriz confusión instancias con dataset UNB**

* En la precisión de realiza el cálculo de la suma de verdaderos positivos más verdaderos negativos, dividido por la suma de verdaderos positivos más falsos positivos, falsos negativos y verdaderos negativos, dejando un resultado de 0.946.

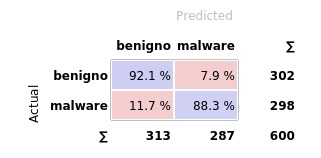
**Precisión = (73+67)/(73+7+1+67) = 0.946**

5.4.4 Item de 600 aplicaciones dataset UNB

* Este dataset tiene una precisión del 0.974, con el algoritmo seleccionado de Neural Network, con dataset UNB de 148 registros

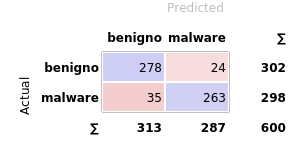


**Ilustración 42, resultados precisión 600 registros con dataset UNB**

* Matriz de confusión porcentajes con 600 registros de la fuente de datos UNB, con resultado de 92.1% cómo verdaderos positivos (VP), 88.3% de verdaderos negativos (VN), 7.9% de falsos positivos (FP) y finalmente 11.7% de falsos negativos (FN).

**Ilustración 43, matriz confusión porcentajes con dataset UNB**

* Matriz de confusión instancias con 600 registros de la fuente UNB, con resultado de 278 cómo verdaderos positivos (VP), 263 verdaderos negativos (VN), 24 de falsos positivos (FP) y finalmente 35 de falsos negativos (FN).

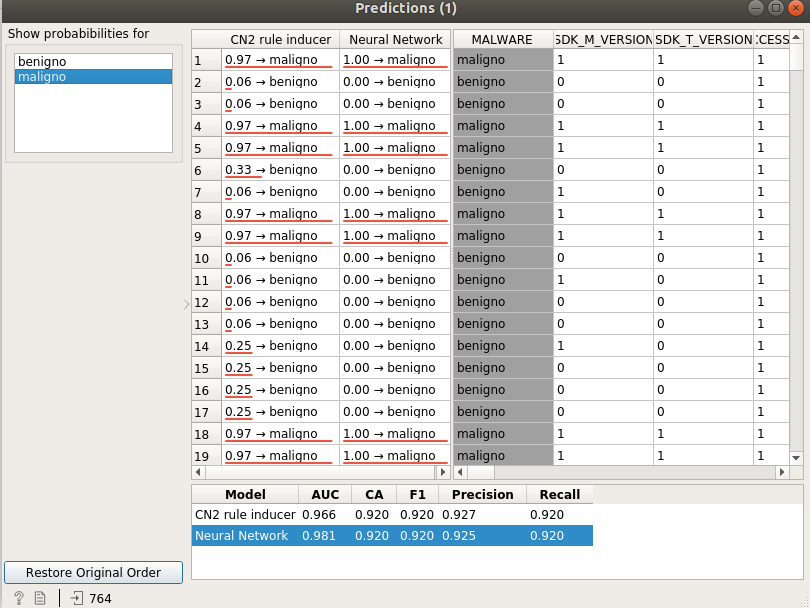


**Ilustración 44, matriz confusión instancias con dataset UNB**

* En la precisión de realiza el cálculo de la suma de verdaderos positivos más verdaderos negativos, dividido por la suma de verdaderos positivos más falsos positivos, falsos negativos y verdaderos negativos, dejando un resultado de 0.902.

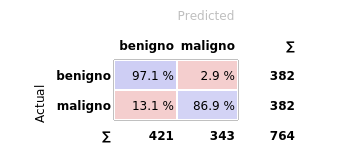
**Precisión = (278+263)/(278+35+24+263) = 0.902**

5.4.5 Item de 764 aplicaciones Dataset UNB

* Este dataset tiene una precisión del 0.981, con el algoritmo seleccionado de Neural Network, con dataset UNB de 148 registros

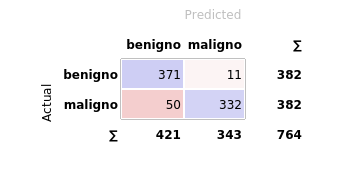
**Ilustración 45, resultado precisión 764 registros con dataset UNB**

* Matriz de confusión porcentajes con 764 registros de la fuente UNB, con resultado de 97.1% cómo verdaderos positivos (VP), 86.9% verdaderos negativos (VN), 2.9% de falsos positivos (FP) y finalmente 13.1% de falsos negativos (FN).



**Ilustración 46, matriz confusión porcentajes con dataset UNB de 148 registros**

* Matriz de confusión instancias con 764 registros de la fuente UNB, con resultado de 371 cómo verdaderos positivos (VP), 332 verdaderos negativos (VN), 11 de falsos positivos (FP) y finalmente 50 de falsos negativos (FN).



**Ilustración 287, matriz confusión instancias con dataset UNB**

* En la precisión de realiza el cálculo de la suma de verdaderos positivos más verdaderos negativos, dividido por la suma de verdaderos positivos más falsos positivos, falsos negativos y verdaderos negativos, dejando un resultado de 0.902.

**Precisión = (371+332)/(371+50+11+332) = 0.920**

5.4.6 Tabla comparativa de resultados

Los distintos dataset con el algoritmo seleccionado dieron una predicción por sobre del 0.90, existe una diferencia entre los primeros dataset que se sustrajeron desde la fuente virusshare la cual el tipo de aplicaciones malignas se pudo sustraer más cantidades que UNB, por la cantidad de aplicaciones que no eran Android, la precisión que se llegó obtener en porcentaje general de los distintos dataset en precisión del 0.9016, en sumatoria de todas las pruebas en los distintos dataset la precisión más alta fue 0.946 con el dataset del UNB con 148 aplicaciones. En esta comparación y estadística se logra apreciar que con los datos de UNB se logra una mayor precisión. En AUC (Área bajo la curva ROC, AUC mide toda el área bidimensional por debajo de la curva ROC completa), el de mayor precisión es el mismo dataset con 0.990 demostrando estos resultados en la siguiente tabla.

**Tabla 20, comparativa resultados de distintos dataset**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nro.** | **Fuente dataset** | **Cantidad APK** | **AUC** | **Precisión** |
| 1 | virusshare | 148 | 0.988 | 0.926 |
| 2 | Virusshare | 1000 | 0.924 | 0.814 |
| 3 | UNB | 148 | 0.990 | 0.946 |
| 4 | UNB | 600 | 0.974 | 0.902 |
| 5 | UNB | 764 | 0.981 | 0.920 |

CAPÍTULO VI CONCLUSIONES

Uno de los proyectos era identificar los metadatos utilizados para detectar malware utilizando aprendizaje automático y análisis estático, cumpliéndose los objetivos al utilizar estos metadatos del manifiesto de Android que se extrajeron y las que fueron para recordar fueron los permisos, MainActivity, SDK y receiver, entre otras más que se podrían agregar en los futuros trabajos, haciendo este método más robusto.

Según los objetivos específicos propuestos en esta investigación uno de ellos era diseñar un método de detección de malware en aplicaciones Android basado en aprendizaje automático y análisis estático, lo cual se concluye en este método de extracción de metadatos específicos utilizados por los malwares, las cuales se logró extraer mediante análisis estático con el sistema scripting python, el que fue de gran utilidad.

Validar la eficiencia del método diseñado utilizando un conjunto de datos de malware Android y los resultados de la medición con métricas de aprendizaje automático, esta investigación se valida en la conclusión anterior, donde da un porcentaje de precisión mayor al 90%, porque la calidad de los datos se extrajo de fuentes que sirven para investigaciones de esta índole.

Diseñar un método para la detección de malware, basado en análisis estático, extrayendo los metadatos y comportamientos propios del malware, el cuál este pueda evaluar los metadatos propios de aplicaciones Android y se pueda hacer una comparación de las familias de malware usando un algoritmo de aprendizaje automático, se entiende este objetivo cumplido con todos los anteriores, cómo los que fueron la creación del método con extracción de metadatos del manifiesto, mediante extracción y análisis estático, comparando estos metadatos con distintas clases de malware Android que se extrajeron de las fuentes Virusshere y UNB.

Se logró apreciar que entre mayor cantidad de datos la precisión puede variar más del porcentaje óptimo, aun así en los datos de UNB entre los registros de 600 y 764 este último tiene una mayor presión, se concluye que depende de la calidad de los datos se refleja mayor precisión, cómo también si las aplicaciones tienen mayor detalle en las metadatos extraíbles se puede obtener un mejor resultado, no así entre más cantidad de aplicaciones, concluyendo finalmente que deben ser ambos factores integrados, calidad de los datos y cantidad para obtener mayores precisiones en los resultados finales.

Se podrían haber integrado más metadatos de identificación del malware con el sistema scripting Python, esto se podría hacer de manera incremental integrando más librerías en el sistema que extrae metadatos Android para robustecer el sistema, pero se requiere un tiempo de investigación donde se debe estudiar el plazo para poder detectar nuevos metadatos, y se debería incluir los datos día cero, esto último se entiende de las nuevas variantes de malware, vulnerabilidades y errores de fabrica de las aplicaciones.

Existe un potente material para continuar investigando sobre Android y detección malware en internet, motivado por crecimiento de la plataforma y además de la integración del sistema operativo en distintos equipos de IoT, debido a esto el factor de ataque es más amplio y las técnicas de ataque son variadas, motivo de esto se debe seguir la investigación en detección de malware en Android utilizando Inteligencia Artificial, con sus derivados de Aprendizaje automático y Deep learning conocido cómo aprendizaje profundo, este último campo ha sido ampliamente explotado estos últimos años por su Precisión en distintos campos inclusive en la medicina.

Con el avance de la tecnología en, vehículos autónomos, IoT, drones, industria aeroespacial, medicina, etc., y cómo se ha visto en este último tiempo por ejemplo de automóviles con capacidades de volar, todas estas tecnologías integradas con dispositivos Android, el riesgo de un ataque es más amplio, por eso se hace necesario crear sistemas de detección avanzados robustos que puedan detener ataques a estas nuevas integraciones, imagínese un vehículo volador autónomo, nave de guerra, con navegación remota ó autónoma secuestrada por hackers, el impacto que nos podría provocar este riesgo en la seguridad mundial.

Los objetivos relacionados con el trabajo se llevaron a cabo de manera óptima, se cumple con ellos, después de un mapeo sistemático, la búsqueda de trabajos relacionados, mapeo de información, se logró extraer lo más importante de cada uno de ellos, igual se integra un trabajo investigativo con aplicativo, utilizando laboratorios personales lo que hace más interesante poder seguir con la investigación en este campo, debido la cantidad de dispositivos que podríamos abarcar.

Trabajos futuros:

En este trabajo de tesis, se logra con metadatos extraídas del archivo Manifiest de las aplicaciones Android, creando un dataset para crear con Redes Neuronales el algoritmo de aprendizaje máquina, se puede agregar secuencialmente más metadatos para extraer de la aplicación Android, se podría extraer metadatos no solo del Manifiest sino del propio código, certificado, llamadas API, entre otras más que la detección sea más robusta a la hora de aplicarla, incluso hasta el comportamiento de las aplicaciones con un estudio durante un tiempo determinado, correlacionando por ejemplo transacciones, días y horas de alta demanda, donde el sistema pueda determinar un comportamiento anómalo en la plataforma.

Una vez obteniendo un modelo de aprendizaje cómo el de esta tesis, se puede crear un sistema igual a las ilustraciones propias de la tesis, sea en un sistema web ó aplicación móvil.

El ideal de esta investigación, es ir agregando más metadatos extraíbles, como por ejemplo los día 0 constantemente a las selección de metadatos, y que este sistema se alimente de distintas fuentes, actualice el sistema, cargue la característica y el script de selección agregue el nuevo atributo, para finalmente mantenerlo sincronizado a los nuevos ataques, malwares, vulnerabilidades, etc.

Para lograr diseñar y desarrollar un sistema de detección se podría hacer con alguna plataforma amigable al usuario, la que destacamos cómo ejemplo podría ser una aplicación web ó Android.

**Aplicación web:** En una plataforma nube con todas las medidas de seguridad disponible a todos los usuarios para subir aplicaciones para que puedan ser detectadas, y que esta se alimente constantemente con nuevos metadatos ingresadas de distintas fuente, y que podría ser de utilidad a las desarrolladores en sus ciclos de desarrollo seguro integrado a su pipeline.

**Aplicación Android:** Aplicación que pueda ser instalada en el dispositivo y consulte el resto de los paquetes instalados en el dispositivo y detecte posibles aplicaciones vulnerables.

Además, de crear el sistema se puede agregar un módulo donde el usuario pueda agregar los metadatos a detectar cómo vulnerables y este automáticamente actualice el sistema para que detecte esa característica y a la agregue al modelo constantemente.

Este sistema se podría integrar con distintos dispositivos de IoT, donde cada característica seleccionada en el dataset descarte el uso de una aplicación por poseer un alto riesgo a ser utilizada a futuro como malware, además de entregar una fecha aproximada de explotación y tiempo de compromiso del sistema, se puede igual integrarse a un sistema de análisis dinámico con las mismos metadatos tipo Sandbox, en tanto los dos sistemas sean uno solo integrado con distintos dispositivos y estén alertando aplicaciones ó intrusos en la red de dispositivos, comunicándose a través de algún protocolo especial diseñado para este sistema, ideal sería inventar uno, este sistema debería ser constantemente alimentando con nuevos metadatos, no solo del manifiesto sino también del certificado, Api, código, imágenes, todas las que se puedan integrar al sistema para su robustez.

Se podría integrar el máximo de seguridad en todas las nuevas tecnologías, especialmente en las que indicamos más arriba las que indica de vehículos autónomos, máquinas de guerra, entre otras tecnologías críticas, el ideal sería ingeniar un sistema en el cual pueda integrar todas las máquinas de alto valor para las organizaciones y diseñar un sistema de alerta notificación, bloqueo y reporte de malware en estos equipos, abarcando en total el análisis de comportamiento, análisis estático, notificación, reporte programado, bloqueo, alerta de posibles ataques, y creación de listas negras cómo blancas en el sistema.

Bibliografía

1. Carton, B., Mongardini, M. J., & Li, Y. (2018). A New Smartphone for Every Fifth Personon Earth: Quantifying the New TechCycle. International Monetary Fund.
2. S. O'Dea. (2021). Number of smartphone users world wide from 2016 to 2021 .01/01/2021, de Statista Sitio web: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/
3. G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and A. Ribagorda, Evolution, detection and analysis of malware for smart devices, IEEE Communications Surveys & Tutorials, vol. 16, no. 2, pp. 961-987, 2014.
4. S. O'Dea. (2020). Share of global smartphone shipments by operating system from 2014 to2023 .01-01-2021, de Statista Sitio web: https://www.statista.com/statistics/272307/market-share-forecast-for-smartphone-operating-systems/
5. Kouliaridis, V., Barmpatsalou, K., Kambourakis, G., & Chen, S. (2020). A survey on mobile malware detection techniques. IEICE Transaction son Information and Systems, 103(2), 204-211.
6. Morales-Ortega, S., Escamilla-Ambrosio, P.J., Rodríguez-Mota, A., Aguirre-Anaya, E. (2015). Detección de malware en teléfonos inteligentes con sistema operativo Android: Estado del arte, Revista de Ciencia e Ingeniería del Instituto Tecnológico Superior de Coatzacoalcos, Año 2, No. 2, enero-diciembre, pp. 185-190.
7. Morales-Ortega, S., Escamilla-Ambrosio, P.J., Rodríguez-Mota, A., Coronado-De-Alba, L.D. (2016). Native Malware Detection in Smartphones with Android OS Using Static Analysis, Feature Selection and Ensemble Classifiers, Proceedings of the 11th IEEE International Conferenceon Malicious and Unwanted Software (MALWARE 2016), Oct-18-21, Fajardo, Puerto Rico, USA, pp. 67-74.
8. Rodríguez-Mota, A., Escamilla-Ambrosio, P.J., Happa, J., Aguirre-Anaya, E. (2016). GARMDROID: IoT potential security threats analysis through the inference of Android applications hardware feature requirements, Proceedings of the EAI International Conference on Future Internet and Internet of Things Applications, May 25-27, Puebla, Puebla, México
9. Inzaurralde, M., Isi, J., &Garderes, J. (2006). Telefonía celular.Recuperado el viernes, 25.
10. Polanco, K. M., &Taibo, J. L. B. (2011). “Android” el sistema operativo de Google para dispositivos móviles. Negotium: revista de ciencias gerenciales, 7(19), 79-96.
11. Gastañaga, I., Gibellini, F. A., Ruhl, A. L., Frias, P., Parisi, G., Zea Cárdenas, M., ... & Olmedo, P. (2019). Laboratorio de análisis de malwares para fines educativos. In XIV Congreso Nacional de Tecnología en Educación y Educación en Tecnología (TE&ET 2019),(Universidad Nacional de San Luis, 1 y 2 de julio de 2019).
12. Hinestroza Ramírez, D. (2018). El Aprendizaje automático a través de los tiempos, y los aportes a la humanidad (Doctoral dissertation, Universidad Libre Seccional Pereira).
13. Bustos, D. (2018). Malware para dispositivos móviles (Android).Trabajo de grado/Universidad Tecnológica de Chile). Recuperado de: https://premios. eset-la. com/universitario/pdf/malaware\_para\_dispositivos\_moviles\_android. Pdf.
14. Huilcamaigua Pazuña, S. R. (2017).Aplicación de una metodología para el análisis de los efectos de malware en dispositivos móviles con sistema operativo Android en el Ecuador (Bachelor'sthesis, Quito, 2017.).
15. Williams, M. (2017). Análisis Estático de programas móviles. Estados Unidos: Universidad Stanford
16. Haisheng Yan, Lingling Peng. (18 de Abril de 2018). Detección de malware de Android basado en una súper red evolutiva. Portal federal de ciencia. AIP Conference Proceedings
17. Hiroya KATO, Shuichiro HARUTA, Iwao SASASE. (1 de Febrero de 2020). Android Malware Detection Scheme Based on Level of SSL Server Certificate. Portal Federal de Ciencia. J-STAGE
18. Chebyshev, V. (2019). Kaspersky Lab: la cantidad de ataques con malware móvil se duplicó en 2018. Recuperado 25 de mayo de 2020, de Kaspersky website: <https://latam.kaspersky.com/about/press-releases/2019_kaspersky-lab-la-cantidad-de-ataques-con-malware-m-vil-se-duplic-en-2018>
19. Bustos, D. (2018). Malware para dispositivos móviles (Android). Trabajo de grado/Universidad Tecnológica de Chile). Recuperado de: https://premios. eset-la. com/universitario/pdf/malaware\_para\_dispositivos\_moviles\_android. Pdf.
20. Alpaydin, E. (2020). Introduction to machine learning. MIT press.
21. Alpaydin, E. Introduction to Machine Learning. MIT Press. (2014).
22. XuJiang, Baolei Mao, Jun Guan, Xingli Huang. (2020, 25 Enero). Android Malware Detection Using Fine-Grained Features. Hindawi. <https://www.hindawi.com/journals/sp/2020/5190138/>
23. Xin Su, Weiqi Shi, Xilong Qu, Yi Zheng & Xuchong Liu. (2020, 3 Enero). DroidDeep: using Deep Belief Network to characterize and detect android malware. Springer Link. <https://link.springer.com/article/10.1007/s00500-019-04589-w#Bib1>
24. Zhuo Ma, Haoran Ge, Zhuzhu Wang, Yang Liu, Ximeng Liu. (2020, 10 febrero). Droidetec: Android Malware Detection and Malicious Code Localization through Deep Learning. Cornell University. <https://arxiv.org/abs/2002.03594>
25. Developers Android. (2020, 15 febrero). Sugerencias de seguridad | Desarrolladores de Android. Android Developers. https://developer.android.com/training/articles/security-tips?hl=es-419
26. Developers Android. (2020, 15 febrero). Sugerencias de seguridad | Desarrolladores de Android. Android Developers. https://developer.android.com/training/articles/security-tips?hl=es-419
27. Kitchenham, B., (2004) Procedures for Performing Systematic Reviews, TR/SE-0401, Keele University.
28. XuJiang, Baolei Mao, Jun Guan, y Xingli Huang. (2019). Detección de malware de Android con funciones específicas. 01-07-2020, de Hindawi programación científica Sitio web: <https://www.hindawi.com/journals/sp/2020/5190138/>
29. Manel Jerbi, Zaineb Chelly Dagdia, Slim Bechikh, Lamjed Ben Dijo. (2020). Sobre el uso de patrones maliciosos artificiales para la detección de malware de Android. 01-07-2020, de Sciencie Direct Sitio web: <https://www.sciencedirect.com/science/article/pii/S0167404818309994>
30. Pektaş, A., & Acarman, T. (2019, 23 marzo). Deep learning for effective Android malware detection using API call graph embeddings. Soft Computing. <https://link.springer.com/article/10.1007/s00500-019-03940-5?error=cookies_not_supported&code=3d9a02f8-4c55-4e40-a1b4-ead01f6a6ad2>
31. Ma, Z. (2020, 10 febrero). Droidetec: Android Malware Detection and Malicious Code Localization through Deep Learning. arXiv.org. <https://arxiv.org/abs/2002.03594>
32. Zhongru Ren, Haomin Wu, Qian Ning, Iftikhar Hussain, Bingcai Chen. (2020, 15 Abril). End-to-end malware detection for android IoT devices using Deep learning. ScienceDirect. <https://www.sciencedirect.com/science/article/abs/pii/S1570870519310984>
33. Ananya, A., Aswathy, A., Amal, T. R., Swathy, P. G., Vinod, P., & Mohammad, S. (2020, 13 Enero). SysDroid: a dynamic ML-based android malware analyzer using system call traces. Springer Link. <https://link.springer.com/article/10.1007/s10586-019-03045-6?error=cookies_not_supported&code=729af90b-447b-443d-a3b0-1e9fa00cc3b7>
34. Shanshan Wang ,Zhenxiang Chen ,Qiben Yan ,Bo Yang, Zhongtian Jia. (2019, 1 mayo). A mobile malware detection method using behavior features in network traffic. ScienceDirect. <https://www.sciencedirect.com/science/article/abs/pii/S1084804518304028>
35. Su, X., Li, X., Liu, X., Li, K. C., & Liang, W. (2020, 19 Abril). DroidPortrait: Android Malware Portrait Construction Based on Multidimensional Behavior Analysis. MDPI. <https://www.mdpi.com/2076-3417/10/11/3978>
36. Yerima, S. Y y Alzaylaee. (2020, 19 junio). Detección de botnets móviles: un enfoque de aprendizaje profundo utilizando redes neuronales convolucionales. DE MONTFORT UNIVERSITY. <https://146.227.160.144/handle/2086/19818>
37. M Wadkar, F Di Troia, M Stamp. (2020, 1 Abril). Detecting malware Evolution using support vector machines. ScienceDirect. <https://www.sciencedirect.com/science/article/abs/pii/S0957417419307390>
38. Salah, A. (2020, 28 Abril). A Lightweight Android Malware Classifier Using Novel Feature Selection Methods. MDPI. <https://www.mdpi.com/2073-8994/12/5/858>
39. R Mateless, D Rejabek, ó Margalit. (2020, 1 septiembre). Decompiled APK based malicious code classification.ScienceDirect. <https://www.sciencedirect.com/science/article/abs/pii/S0167739X19325129>
40. Maigida, A. M., Abdulhamid, S. M., Olalere, M., Alhassan, J. K., Chiroma, H., & Dada, E. G. (2019, 3 mayo). System at icliterature review and metadata analysis of ransomware attacks and detection mechanisms. Springer Link. <https://link.springer.com/article/10.1007/s40860-019-00080-3?error=cookies_not_supported&code=c42826b9-89a2-465e-970e-150fd233fc99>
41. Zhou, Q., Feng, F., Shen, Z., Zhou, R., Hsieh, M., & Li, K. (2018, 11 Agosto). A novel approach for mobile malware classification and detection in Android systems. Springer Link. <https://link.springer.com/article/10.1007/s11042-018-6498-z?error=cookies_not_supported&code=1fa15f1d-381e-4b17-90a8-0ef66b3723b6>
42. C Zhao, C Wang, W Zheng. (2019, 16 Octubre). Familial Clustering for Weakly-Labeled Android Malware Using Hybrid Representation Learning - IEEE Journals& Magazine. IEEE Xplore. <https://ieeexplore.ieee.org/abstract/document/8871171>
43. Kedziora, M. (2019, 1 Enero). Malware Detection Using Machine Learning Algorithms and Reverse Engineering of Android Java Code by Michal Kedziora, Paulina Gawin, Michal Szczepanik, Ireneusz Jozwiak :: SSRN. SSRN. <https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3328497>
44. Oğuz Emre Kural; Durmuş Özkan Şahin; Seda tAkleylek; ErdalKılıç T. (2019, 1 Septiembre). Permission Weighting Approaches in Permission Based Android Malware Detection - IEEE Conference Publication. IEEE Xplore. <https://ieeexplore.ieee.org/abstract/document/8907187>
45. Abdullah, Z. (2020, 22 Enero). Android Ransomware Detection Based on Dynamic Obtained Features. Springer Link. <https://link.springer.com/chapter/10.1007/978-3-030-36056-6_12>
46. Hendra Saputra, Setio Basuki, Mahar Faiqurahman. (2020, 15 enero). Algoritmos de clasificación. researchgate. <https://www.researchgate.net/publication/338624768_Classification_Algorithms>
47. Joseph Yisa Ndagi, John Alhassan. (2019, 12 diciembre). Algoritmos de clasificación de aprendizaje automático para adware en dispositivos Android: una evaluación y análisis comparativos. researchgate. <https://www.researchgate.net/publication/340119278_Machine_Learning_Classification_Algorithms_for_Adware_in_Android_Devices_A_Comparative_Evaluation_and_Analysis>
48. Pei, X. (2020, 1 Enero). Combining multi-features with a neural joint model for Android malware detection - IOS Press. IOS Press. <https://content.iospress.com/articles/journal-of-intelligent-and-fuzzy-systems/ifs190888>
49. J Gajrani, M Tripathi, V Laxmi, G Somani. (2020, 1 mayo). Vulvet: Verificación de vulnerabilidades en aplicaciones de Android para frustrar la explotación. Digital Library. <https://dl.acm.org/doi/abs/10.1145/3376121>
50. Elsabagh, M. (2020). FIRMSCOPE: Automatic Uncovering of Privilege-Escalation Vulnerabilities in Pre-Installed Apps in Android Firmware | USENIX. 29 TH USENIX SECURITY SYMPOSIUM. <https://www.usenix.org/conference/usenixsecurity20/presentation/elsabagh>
51. J Qin, H Zhang, J Guo, S Wang, Q Wen, Y Shi. (2020). Vulnerability Detection on Android Apps–Inspired by Case Study on Vulnerability Related With Web Functions. IEE EXPLORER. <https://ieeexplore.ieee.org/abstract/document/9102313>
52. SM Shambra. (2020, 22 abril). Automated Vulnerability Assessment of Mobile Device Vulnerabilities. MISSISSIPPI STATE UNIVERSITY. <https://ir.library.msstate.edu/handle/11668/16917>
53. A Alanda , D Satria , HA Mooduto , B Kurniawan. (2019, 1 noviembre). Pruebas de penetración de seguridad de aplicaciones móviles basadas en OWASP. Iop science. <https://iopscience.iop.org/article/10.1088/1757-899X/846/1/012036/meta>
54. Y Wang, X Liu, W Mao, W Wang. (2019, junio). DCDroid: detección automatizada de vulnerabilidades de verificación de certificados SSL/TLS en aplicaciones de Android. Digital Library. <https://dl.acm.org/doi/abs/10.1145/3321408.3326665>
55. SA Gorski III, W Enck. (2019, Mayo). ARF: identificación de vulnerabilidades de re delegación en los servicios del sistema Android. Digital Library. <https://dl.acm.org/doi/abs/10.1145/3317549.3319725>
56. D Wu, D Gao, EKT Cheng, Y Cao, J Jiang. (2019, junio). Hacia la comprensión de las vulnerabilidades del sistema Android: técnicas e ideas. Digital Library. <https://dl.acm.org/doi/abs/10.1145/3321705.3329831>
57. Z Namrud, S Kpodjedo, C Talhi. (2019, noviembre). AndroVul: un repositorio para vulnerabilidades de seguridad de Android. Digital Library. <https://dl.acm.org/doi/abs/10.5555/3370272.3370279>