# Clasificación de ataques de intrusión por red

## Proyecto 1

Gustavo Méndez[a], Roberto Figueroa [b]

Departamento de Ing. CIencias de la computación y T.I

Universidad del Valle de Guatemala, Guatemala

[a] men18500@uvg.edu.gt, [b]fig18306@uvg.edu.gt

**Resumen— Este documento describe el proceso de implementación de dos modelos de aprendizaje de máquina: *Random Forest* y *Decision Tree*. Tras realizar la respectiva exploración, selección de características, balanceo, síntesis de características y preprocesamiento, los modelos se sometieron al mismo dataset. Ambos modelos lograron métricas aceptables, sin embargo, *Random Forest* fue quien logró los mejores puntajes.**

**Abstract— This document describes the implementation process of two machine learning algorithms: Random Forest and Decision Trees. After data exploration, feature selection, data balancing, feature generation and preprocessing, the models were trained. All of them achieved good metrics but Random Forest got the best results.**

***Palabras clave—Random Forest, Decision Tree, seguridad, ciberseguridad, ataques, detección, ciencia de datos.***

## I. Introducción

Hoy en día el Internet ha sido un ente no sólo de comunicación si no también ha sido una herramienta de transformación. Ha permitido reconfigurar la manera en que trabajamos, hacemos negocios, nos relacionamos y nos permite entretenimiento variado. Sin duda alguna esta herramienta es para muchos cotidiana. Es por ello que el incremento de usuarios y aplicaciones de esta red de redes trajo también vulnerabilidades y personas generalmente con intenciones no tan adecuadas, las han aprovechado para robar información y/o afectar el uso de cierto servicio y obtener activos de manera fraudulenta. Contar con arquitecturas robustas que brinden seguridad en los servicios que día con día se utilizan en la nube es mandatorio. Dentro de la seguridad implementada en los servidores de red, se pueden definir tres principales capas: seguridad a nivel de sistema operativo, seguridad a nivel de aplicaciones y seguridad de red siendo esta última a la que este proyecto se enfoca [8]. El conjunto de datos utilizado para el desarrollo de este proyecto fue obtenido del proyecto SIMARGL. La información recabada fue con base en la captura de ataques en una red diseñada para dicho propósito. El protocolo utilizado para tal fin fue Netflow en la versión 9.

## II. Marco Teórico

SIMARGL (*Secure Intelligent Methods for Advanced Recognition of Malware and Stegomalware*) es un consorcio de 14 miembros, de 7 países europeos. Los objetivos estratégicos consisten en proveer métodos efectivos para contrarrestar ataques de cibercrimen y proponer, implementar y validar métodos de aprendizaje de máquina para la detección de malware [2].

Para el desarrollo del conjunto de datos se utilizó una red diseñada para que fuese atacada. El protocolo utilizado fue Netflow. Netflow fue desarrollado por Cisco y es utilizado para recolectar y registrar todos los paquetes IP del tráfico saliente y entrante de routers o switches Cisco que permitan tener Netflow habilitado. Pasado un tiempo de desarrollo de este protocolo, se convirtió en un estándar que muchas empresas están implementando en sus productos, incluyendo Juniper (JFlow), Netgear (SFlow), Huawei (NetStream), entre otros. La manera en que opera es cuando un paquete entra en la interfaz del router o switch y que no ha sido visto anteriormente, se determina si es necesario enrutarlo o no. En caso se enrute, este se registra como una entrada en el *Cache Flow* o flujo de caché. Cada paquete que es enrutado se examina para obtener ciertos atributos que los diferencian de otros. Un flujo de IP registrado en el flujo de caché está compuesto principalmente de 5 hasta 7 atributos dentro de los cuales pueden ser:

- Dirección IP fuente
- Dirección IP destino
- Puerto de origen
- Protocolo de capa 3
- Clase de servicio
- Interfaz del router o switch

Aquellos paquetes que contengan los mismos atributos son agrupados formando así un *Flow* o flujo. Mientras se sigan enrutando paquetes que coincidan con otro que esté registrado se actualizan los campos como el contador de paquetes y bytes hasta que la conexión entre estas dos partes se culmine. Cuando la sesión expire o se culmine, los flujos son

exportados al *Netflow Collector* quien constantemente analiza y registra los flujos para futuras referencias [9].

Dentro de las categorías que este trabajo se enfoca en identificar y clasificar están:

- Flujo normal
- SYN Scan
- DDoS R.U.D.Y
- DDos Slowloris

Un ataque DDoS es caracterizado por un intento explícito de prevenir el uso legítimo de un servicio. Este ataque despliega múltiples unidades de ataque para cierto objetivo o blanco. Una manera común de realizar este ataque es enviando un flujo de paquetes a una víctima, este flujo consume algunos recursos clave dejando así a las víctimas no acceder de manera legítima al servicio. Otra manera de realizar este ataque es enviando paquetes malformados que confunden a la aplicación o al protocolo de la máquina de la víctima forzándola a congelar y reiniciar la máquina [7].

Dentro de los ataques DDoS podemos nombrar al tipo Slowloris. Este ataque fue descubierto por Michal Zalewski y Adrian Ilarion en 2007. Se trata de un ataque implementado bajo HTTP con el verbo GET. Consiste en enviar un paquete malformado, para el caso de Michal y Adrian, enviaron un paquete con caracteres de nueva línea. Esto hace que el servidor mantenga la conexión esperando una terminación exitosa, sin embargo, esta nunca se notifica. Con esta idea se pueden emplear muchas peticiones de este tipo llegando a dejar al servidor sin puertos para conexiones legítimas y denegando su conexión. También podemos nombrar al ataque R.U.D.Y el cual es explotado bajo HTTP pero con el verbo POST. La conexión se emplea de manera similar a Slowloris solo que con la diferencia de que el atacante envía una petición POST sin definir el cuerpo del mensaje y enviándolo en pequeños trozos en intervalos de tiempo aleatorios. El servidor de nuevo espera la llegada del final del cuerpo del POST dejando la conexión abierta por tiempo indefinido llegando a saturar el servidor y denegando el servicio a conexiones legítimas [3].

SYN-Scan es una táctica que los hackers utilizan para determinar el estado de un puerto de comunicación sin establecer una conexión completa. Se basa en tratar de establecer conexión con un servidor bajo un determinado puerto enviando un paquete con la bandera SYN [5]. Esto se realiza con todos los puertos de un servidor. Si el servidor responde ACK quiere decir que el puerto está abierto, acto seguido el atacante envía un paquete con la bandera RST. Como resultado el servidor asume que hubo un error de conexión con el cliente dejando el puerto abierto cuando realmente esto no es cierto [1].

## III. Propuesta de metodología

Los datasets seleccionados para poder llevar a cabo el proyecto han sido las dos primeras partes del total de datos, contando con un total de cuatro clases a clasificar (las categorías de trabajo listadas en la sección anterior). El desarrollo actual para poder lograr el objetivo de implementar dos modelos de clasificación y detección de ataques constó de seis diferentes fases que fueron llevadas a cabo: análisis exploratorio, preprocesamiento de la data, selección de características o *features*, separación de datos de entrenamiento y pruebas, implementación de los modelos y la síntesis de los mismos por medio de métricas de evaluación.

### *A. Análisis Exploratorio*

Esta es la fase inicial y genérica que un *data scientist* debe realizar para poder empezar a relacionarse con la información. Sin embargo, la cantidad de datos para analizar alcanzaba los 12 millones de registros, algo que tiene un gran impacto para la memoria. Por defecto, Pandas (librería de Python) maneja todos los valores numéricos con el tipo de dato de mayor memoria (int64 y float64 por ejemplo). Por tanto, una tarea importante fue reducir los valores para ajustarlos a un tipo de dato que consumiera menos memoria: para el primer dataset se tuvo un 53.2% de reducción de memoria. Cada vez que se tenía que cargar un dataset, el uso de memoria era un factor clave para poder manejar más fácil y cómodamente los datos.

Se logró constatar que los dos datasets contenían las cuatro clases que sería útiles para clasificar el flujo de red y detectar anomalías o ataques: Normal flow, SYN Scan, DDoS por medio de R.U.D.Y y DDoS por Slowloris. Sin embargo, la data estaba demasiado desbalanceada, ya que el 53.8% de la data constaba de datos etiquetados como Normal flow y apenas el 7.1% como DDoS por Slowloris. Finalmente, se verificó la correlación entre variables logrando observar que muchas variables eran derivaciones de otras variables (milisegundos y segundos por ejemplo), por lo que se constató que esa redundancia de datos y desbalance tenía que tratarse en la siguiente fase.

### B. *Preprocesamiento*

Como input de esta fase se describe un nuevo dataset concatenado, con un total de más de 12 millones de datos, cargados por *chunks* a memoria. Esta fase se subdivide en otras cuatro subfases:

1. Eliminación de columnas redundantes y correlacionadas

   De la fase anterior, se pudo constatar que existían *features* correlacionadas o derivadas

de otras features, cuyo significado en el mundo de los paquetes TCP es el mismo, muchas veces medidos en otra escala o encapsulados en datos binarios. Así que, de entrada, las siguientes columnas fueron removidas del dataset concatenado:

- BIFLOW_DIRECTION
- FIREWALL_EVENT
- FLOW_ACTIVE_TIMEOUT
- FLOW_INACTIVE_TIMEOUT
- FRAME_LENGTH
- MAX_IP_PKT_LEN
- MIN_IP_PKT_LEN
- SAMPLING_INTERVAL
- L7_PROTO_NAME
- FLOW_START_SEC
- FLOW_END_SEC
- TCP_FLAGS

2. *Encoding* y *feature derivation*

   Por supuesto, el encoding más importante consistió en aplicarlo en la *feature* llamada LABEL, que es la etiqueta deseada a predecir. Las siguientes *features* fueron trabajadas en esta fase:

   - LABEL (*OneHotEncode*)
   - PROTOCOL (Binario y luego *One Hot Encode*)
   - PROTOCOL_MAP (*One Hot Encode*)
   - TOS [*Type Of Service*] (Binario aplicando *Right Shift*, y luego *One Hot Encode*)

3. Balanceo de la data

   Esta fase consistía en reducir la brecha de porcentajes entre clases de etiquetas, así pudiendo trabajar con una muestra mucho más equilibrada. Para ello, es posible aplicar las siguientes técnicas: *SMOTE, Undersampling* y *Oversampling*. Las utilizadas fueron solamente las últimas dos dependiendo la clase a balancear:

   - *Undersampling* para Normal Flow y SYN Scan para poder llevarlo a la cantidad dada de datos etiquetados como DDoS por R.U.D.Y
   - *Oversampling* a DDoS por Slowloris, que era apenas representativo en el dataset con un 7.1% de datos.

   Finalmente, se obtiene un total de 9 millones de datos, listos para poder ser utilizados por los modelos. Sin embargo, para ello debemos normalizar la data, caso que se procede en la siguiente fase.

4. Normalización

   La normalización elimina la media y escala cada *feature* a la varianza de la unidad. Transforma los datos de tal manera que tiene una media de 0 y una desviación estándar de 1. En resumen, estandariza los datos. Las variables fueron normalizadas por medio de *Standard Scaler* (provisto por *Scikit Learn*). Este módulo se encarga de organizar los datos en una distribución normal estándar. Es bastante útil en tareas más orientadas a la clasificación que a la regresión.

Finalmente, se procede a guardar el dataset estandarizado y listo para poder utilizar en las siguientes fases.

### C. *Selección de features*

Esta fase se hizo en paralelo con la anterior, ya que involucra seleccionar aquellas características correlacionadas al menos en un 10-20%. Las características seleccionadas se muestran en la Tabla 1. Además, se utilizó criterio de investigación con base en los ataques para poder tomar una mejor decisión a la hora de elegir o no cierta característica.

TABLA 1. FEATURES SELECCIONADAS

| ***Feature*** | **Descripción** |
|---|---|
| FIRST_SWITCHED | *Timestamp* de aparición del primer flujo |
| FLOW_DURATION_MICROSECONDS | Duración del flujo en microsegundos |
| L4_DST_PORT | Puerto destino |
| L4_SRC_PORT | Puerto origen |
| LAST_SWITCHED | *Timestamp* de última aparición del flujo |
| TCP_WIN_MAX_IN | Métricas correspondientes al tamaño de la ventana TCP, máximos y mínimos para entrada y salida |
| TCP_WIN_MAX_OUT | |
| TCP_WIN_MIN_IN | |
| TCP_WIN_MIN_OUT | |

| | |
|---|---|
| TCP_WIN_MSS_IN | Métrica correspondiente al tamaño del segmento TCP de entrada |
| TCP_WIN_SCALE_OUT | Métrica correspondiente al tamaño de escala TCP de entrada y salida |
| TCP_WIN_SCALE_IN | |
| TOTAL_FLOWS_EXP | Número total de flujos exportados |
| tcp | Determina si el protocolo fue TCP, derivada de PROTOCOL_MAP |
| udp | Determina si el protocolo fue UDP, derivada de PROTOCOL_MAP |
| SRC_PKT_PRECEDENCE | Feature derivada de *Type Of Service*, que indica la importancia del paquete o datagrama de origen (así es como se descartan o no) |
| DST_PKT_PRECEDENCE | Feature derivada de *Type Of Service*, que indica la importancia del paquete o datagrama de destino (así es como se descartan o no) |
| URG | Banderas en las cabeceras de los paquetes TCP, derivada de TCP_FLAGS |
| ACK | |
| PSH | |
| RST | |
| SYN | |
| FIN | |
| **LABEL** | Nombre del tipo de ataque |

*D. Separación de datos*

Esta fase es primordial para poder establecer el volúmen de la data que se utilizará. Como buenas prácticas y, sugerido por el material inicial del proyecto, se tienen los siguientes porcentajes para esta separación: 55% entrenamiento, 15% validación y 30% pruebas. Esto corresponde a la siguiente cantidad de datos:

- Total observaciones: 9,107,788
- Observaciones entrenamiento: 5,009,283
- Observaciones de validación: 1,352,507
- Observaciones de pruebas: 2,745,998

*E. Implementación de modelos*

Para la implementación de modelos se utilizó la librería de Python llamada *Scikit Learn*. Esta librería ofrece diversos modelos de clasificación supervisados pero para este proyecto se tomaron en cuenta *Random Forest* y *Decision Tree* pues son modelos supervisados multiclase. Las *features* seleccionadas en su mayoría comprenden datos numéricos, por lo que esto también fue factor para la selección de los mismos.

Se tomó en cuenta *Random Forest* ya que es un modelo que permite dar una buena predicción sin ajustar tantos hiper-parámetros por lo que nos facilita la implementación y desarrollo sin dejar atrás la calidad con que clasifica y predice. Además, por la manera en que nuestra información está representada y planteada, surgió la duda de poder sobre-ajustar nuestro modelo pues al tener ciertas banderas y ciertos valores numéricos el modelo podría intentar imitar el comportamiento a tal modo de aprenderlo. *Random Forest* no permite esto pues en cada árbol aleatorio que genera selecciona un subconjunto de características aleatorias [6].

También se seleccionó el modelo *Decision Tree* pues representa una manera más sencilla que el modelo anterior. Además, por su naturaleza es un modelo que simplifica su entendimiento (no es tanto una caja negra), interpretación y visualización. También la preparación de la información no tiene que ser con mucho detalle pues este modelo permite analizar información numérica y categórica [4].

*F. Métricas*

Finalmente, las métricas básicas a obtener de estos modelos son las siguientes: matriz de confusión, *accuracy, precision, recall, F1 score*. Además, a manera de validar estos valores, se realizó el proceso de validación cruzada con *K-Folds* (con k=10) así como la graficación de la curva ROC para poder mostrar el rendimiento de los modelos de clasificación con base en todas sus clasificaciones (teniendo en cuenta los falsos positivos y los falsos negativos).

El resumen de dichas métricas puede observarse en la Tabla 2 y 3, además de poder observar las curvas ROC en los Gráficos 1 y 2 para cada modelo, respectivamente. Es importante resaltar que *Random Forest* presentó una curva que es bastante ideal, observada en aquellos clasificadores perfectos con un valor cercano a 1.0.

TABLA 2. MÉTRICAS BÁSICAS POR MODELO

| | ***Random Forest*** | ***Decision Tree*** |
|---|---|---|
| *Accuracy* | 0.9999 | 0.9625 |
| *Precision* | 0.9999 | 0.9664 |
| *Recall* | 0.9999 | 0.9624 |
| *F1 score* | 0.9999 | 0.9620 |

TABLA 3. MÉTRICAS AUXILIARES POR MODELO

| | *Random Forest* | *Decision Tree* |
|---|---|---|
| *K-Fold validation* | $0.999 \pm 3.2570 \times 10^{-6}$ | $0.962 \pm 1.8485 \times 10^{-4}$ |
| Área en curva ROC | 1.00 | 0.92 |

GRÁFICO 1. CURVA ROC *RANDOM FOREST*

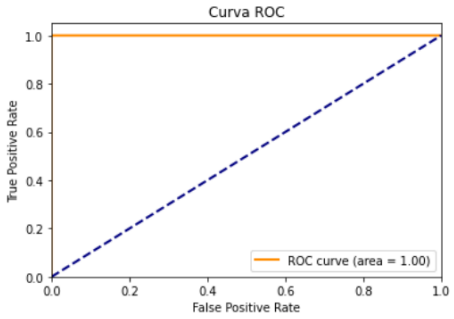
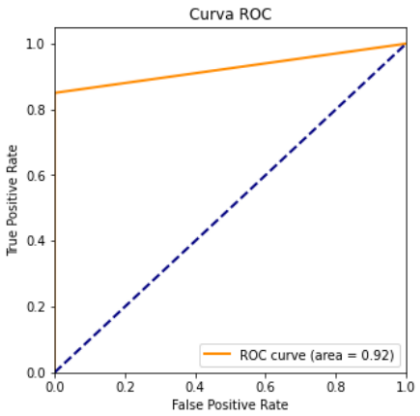



GRÁFICO 2. CURVA ROC *DECISION TREE*




## IV. DISCUSIÓN DE RESULTADOS

El procesamiento de información es una sección importante dentro de la línea de desarrollo de modelos de aprendizaje de máquina. Muchas veces se requieren diversos recursos computacionales para lograr dicha tarea. Para este proyecto los equipos elegidos para trabajar describen las siguientes características:

- Procesador Intel® Core™ i7-7500U CPU @ 2.70GHz × 4 , 16GB de RAM.
- Procesador Intel® Core™ i5-10400F CPU @ 4.30GHz × 6 , 16GB de RAM.

A pesar que los recursos disponibles no eran tan limitados, durante la etapa de exploración y procesamiento se tuvieron que realizar operaciones como carga por bloques y cambiar los tipos de datos que por defecto la librería Pandas le asigna a cada columna. Esto fue debido a que el conjunto de datos era muy grande (más de 12M de observaciones). Sin esto se hubiera tenido que considerar instancias o máquinas virtuales en la nube como las que ofrece *AWS SageMaker* o *Google Compute Engine* con el fin de poder tener la cantidad de RAM necesaria para alojar toda la información.

Otra alternativa hubiera sido cambiar la manera en que se procesa la información y realizar por pequeños bloques; de momento esto no fue necesario. En la fase de creación y entrenamiento de modelos no se tuvo mayor complicación. Para ello, se tomó en cuenta para procesar más rápido el parámetro *n_jobs* con el fin de poder acelerar el proceso y utilizar todos los núcleos disponibles en cada computadora.

En cuanto a los resultados obtenidos por los modelos se puede apreciar que *Random Forest* fue el algoritmo con mejor rendimiento. La selección y ordenamiento de características fue sin duda favorable para este modelo. Esto se valida con la validación cruzada *K-Folds* con k=10. En los 10 puntajes el modelo resolvió con la misma precisión que el modelo con data de entrenamiento y con una desviación de $3.2570x10^{-6}$ (ver Tabla 3). Por otro lado, las métricas obtenidas por el *Decision Tree* fueron también bastante buenas y realistas. Esto se puede observar en la validación cruzada *K-Folds* con k=10. El comportamiento general de ambos modelos se puede observar en las curvas ROC ya que estas nos permiten analizar el comportamiento del modelos tomando en cuenta la tasa de falsos positivos frente a falsos negativos. Ambos modelos cuentan con buenas métricas (ver Gráfico 1 y 2).

En cuanto a las métricas de *precisión* y *recall* se refiere, se podría considerar que la métrica de precisión puede funcionar bastante bien pues se desea que un ataque sea realmente identificado como tal. Quizás se puedan filtrar ataques etiquetados como flujo normal pero como la mayoría de estos ataques funcionan bajo peticiones en masa, no habría problema. Además, si se considera la métrica de *recall,* el

modelo sería muy sensible dejando no pasar tráfico que fuese legítimo por lo que la calidad en el servicio decaería. Este escenario es favorable o no, dependiendo del contexto o políticas de red que se prefieran en un entorno real.

Sin duda, el modelo de ensamble mejoró el rendimiento para predecir ataques ya que *Random Forest* está compuesto de múltiples árboles. A pesar de buscar alternativas con costo computacional menor, mucho más fáciles de comprender y que no se vean afectadas por outliers, *Decision Tree* tuvo métricas aceptables a pesar de no ser el mejor modelo. Para futuras implementaciones, se propone trabajar con otros modelos mucho más costosos computacionalmente como lo sería un *Support Vector Classifier* o *Gradient Boosted Decision Trees* (GBDT). El código escrito en Python puede ser encontrado en el siguiente repositorio de GitHub: https://github.com/gusmendez99/sds-project-1

## Conclusiones

- *Random Forest* fue el mejor modelo para predecir ataques de intrusión por red con una precisión de 0.9999, esto fue validado con validación cruzada *K-folds* con k=10 obteniendo puntajes muy cercanos al rendimiento actual.
- La selección de características y preprocesado de la información fue en lo que más tiempo se invirtió, sin embargo, valió la pena ya que los modelos lograron comprender el conjunto de datos preprocesado con mayor facilidad logrando métricas favorables.
- Dentro de la toma de decisiones respecto a la selección de características se tomó en cuenta no solo la parte numérica, si no también el expertaje que se tiene respecto al tema de ataques de red logrando combinar ambas para tomar la mejor alternativa.
- Los ataques de red en la mayoría de casos son parte de ineficiencia en los protocolos, es por eso que es necesario implementar medidas que compensen estas vulnerabilidades con el fin de poder mitigar o reducir los ataques. Una herramienta de detección es una buena opción para tal fin.
- Se debe tomar en cuenta que los ataques van surgiendo día con día y para obtener un modelo real estos debería actualizarse constantemente.

## Referencias


[1] Balram, S., y Wiscy, M. (2008, November). Detection of TCP SYN scanning using packet counts and neural network. In *2008 IEEE International Conference on Signal Image Technology and Internet Based Systems* (pp. 646-649). IEEE.

[2] Chorás, M., Keller, J., Armin, J., Ariu, D., & Schmidt, N. (2020). About the project. SIMARGL. Recuperado 24 de febrero de 2022, de https://simargl.eu/about/project

[3] Damon, E., Dale, J., Laron, E., Mache, J., Land, N., y Weiss, R. (2012). Hands-on denial of service lab exercises using SlowLoris and RUDY. In *Proceedings of the 2012 Information Security Curriculum Development Conference* (*InfoSecCD '12*). Association for Computing Machinery, New York, NY, USA, 21–29. DOI:https://doi.org/10.1145/2390317.2390321

[4] Gupta, P. (2018, 20 junio). Decision Trees in Machine Learning - Towards Data Science. Medium. Recuperado 24 de febrero de 2022, de https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052

[5] Hanna, K. T. (2021, 22 julio). SYN scanning. SearchNetworking. Recuperado 24 de febrero de 2022, de https://www.techtarget.com/searchnetworking/definition/SYN-scanning

[6] Mbaabu, O. (2020, 11 diciembre). Introduction to Random Forest in Machine Learning. Engineering Education (EngEd) Program Section. Recuperado 24 de febrero de 2022, de https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/

[7] Mirkovic, J. y Reiher, P. (2004). A taxonomy of DDoS attack and DDoS defense mechanisms. *SIGCOMM Comput. Commun. Rev.* 34, 2 (April 2004), 39–53. DOI:https://doi.org/10.1145/997150.997156

[8] N.I.B. (2021). Protect your business online. NI Business. Recuperado 24 de febrero de 2022, de https://www.nibusinessinfo.co.uk/content/server-security

[9] Parker, J. (2021). Netflow - What is it and how does it Work? A Complete Guide. PC & Network Downloads - PCWDLD.Com. Recuperado 24 de febrero de 2022, de https://www.pcwdld.com/what-is-netflow