

# Malicious Event Detection

Rodrigo S. Agredo      Elkin Ramirez N.      David Martinez T.  
*rsierraa@unal.edu.co      elramirez@unal.edu.co      dasmartineztri@unal.edu.co*

Facultad de Ciencias  
 Departamento de Matemáticas  
 Universidad Nacional de Colombia.

Noviembre de 2023

Bogotá, Colombia.

Repositorio:

<https://github.com/rsierraa/Machine-Learning-for-Cybersecurity>

**Abstract**—Effective detection of malicious events in computing environments is an ongoing challenge in cybersecurity. This paper presents an innovative approach that combines the use of decision trees and context-based detection to improve the accuracy and efficiency of malicious event identification. The methodology focuses on the integration of relevant contextual information to strengthen the model's decision-making capabilities.

**Palabras claves**—Adware, Bots, Bugs, Ransomware, Rootkit, Spyware, Trojan horses, Viruses, Worms, Decision tree, Random forest.

## I. INTRODUCCIÓN: CIBERSEGURIDAD, DATOS Y APRENDIZAJE DE MÁQUINA

EN el ámbito de la ciberseguridad, la amenaza constante y evolutiva del malware ha generado la necesidad crítica de desarrollar enfoques avanzados para la detección y mitigación de eventos maliciosos. Este capítulo se sumerge en la exploración de estrategias innovadoras que combinan dos componentes clave: el poder de los árboles de decisión y la sensibilidad contextual para mejorar la capacidad de discernir entre actividades benignas y maliciosas.

El paisaje actual de ciberseguridad se enfrenta a diversas formas de malware, cada una con su conjunto único de actividades dañinas, desde adware y bots hasta ransomware y rootkits. Además, las inyecciones maliciosas, como las SQL injections y manipulaciones en redes inalámbricas, han ampliado la superficie de ataque. En este contexto, nuestro enfoque busca no solo identificar las amenazas conocidas, sino también adaptarse a nuevas variantes y técnicas evasivas.

En este trabajo, se implementa el capítulo 7 del libro Hands-On Machine Learning for Cybersecurity[1]. A lo largo de este capítulo, se explora un caso de uso práctico utilizando el conjunto de datos KDD Cup 1999, con el objetivo de distinguir entre conexiones de red benignas y maliciosas. Esta aplicación concreta nos permitirá evaluar la eficacia de la aplicación de árboles de decisión y la consideración del contexto en la detección de eventos maliciosos.

En última instancia, este capítulo pretende no solo ofrecer un análisis detallado de los enfoques propuestos, sino también proporcionar una perspectiva práctica y aplicada para aquellos inmersos en la seguridad cibernética y el aprendizaje automático.

## II. MARCO TEÓRICO

### A. Machine Learning en Ciberseguridad

Hoy día los protocolos tradicionales de prevención e identificación de amenazas están siendo rápidamente reemplazados por agentes inteligentes.

Utilizados para identificar nuevo malware, ataques zero-day y amenazas de persistencia avanzada (APT), estos nuevos productos de Machine Learning a manera de soluciones tecnológicas están tomando el control, en las áreas de User Behavior, Network Traffic, Endpoint Solutions y SIEM.

### B. Los Datos en el Machine Learning

Los datos son el alimento de los sistemas de aprendizaje que, mediante técnicas para detectar patrones, conocidas como "minería de datos" y otros procesos de preparación exhaustiva, convierten registros de eventos, incidentes, configuraciones, etc, tomados durante períodos específicos en contextos específicos, en modelos que son capaces de clasificar información de manera autónoma y, en última instancia, tomar o ayudar a tomar decisiones de una manera que un conjunto de seres humanos no podría tomar al menos a una velocidad remotamente cercana.

Dentro del mundo de la Ciencia de Datos, existen distintas maneras de estandarizar las muestras; hay muestras estructuradas y no estructuradas, marcadas y no marcadas (labelled and unlabelled); esto dependiendo de la organización y la clasificación previa que se le haya dado a los datos que las componen.

### C. Fases del Aprendizaje de Máquina (Machine Learning)

- Fase de Análisis: Los datos recopilados son analizados para detectar patrones en ellos, que ayudan a definir qué características y parámetros serán relevantes para ser usados al entrenar el modelo.
- Fase de Entrenamiento: Los parámetros generados en la fase anterior se utilizan para crear el modelo de aprendizaje, que es el resultado de un proceso iterativo, donde a medida que las iteraciones avanzan, mejora la calidad de predicción.
- Fase de Pruebas: El modelo obtenido en la fase anterior son puestos a prueba; es entrenado con más datos y su

desempeño es evaluado con métricas. Se suele probar con datos que no fueron usados en la fase de entrenamientos para ver cómo se comporta.

- Fase de aplicación: El modelo ya mejorado, habiendo pasado por un control de calidad, es finalmente alimentado con datos reales del contexto real en el que se planea utilizar. Se dice que el modelo entra entonces al entorno de producción.

Las propiedades, colecciones, mapas, sustituciones y pre-determinados no son definidos directamente por el estándar AIML, sino más bien hacen parte del conjunto de parámetros que personalizan el producto final.

#### D. Tipos de Aprendizaje de Máquina

- Algoritmos de Aprendizaje Supervisado: El aprendizaje supervisado es aquel en el que se utiliza un conjunto de datos conocido para clasificar o predecir con datos en mano
- Algoritmos de Aprendizaje Sin Supervisión: La técnica consiste en no etiquetar los datos iniciales. Las conclusiones se dan procesando datos cuya estructura no se conoce de antemano.
- Aprendizaje por Refuerzo: es un tipo de programación dinámica en la que el software aprende de su entorno para producir una salida que maximice la recompensa.

#### E. Algoritmos en ML

- Máquinas de vectores soporte: Son algoritmos de aprendizaje supervisado que se usan en la clasificación lineal y no lineal. Estas crean un hiper plano en un espacio óptimo en espacios dimensionales altos.
- Redes bayesianas: Son modelos probabilísticos usados para realizar predicciones y ayuda en la toma de decisiones. Se basan en los principios de la teoría de la probabilidad.
- Árboles de decisión: Es una técnica de aprendizaje automático predictivo. Estos usan el análisis de decisiones e intentan predecir el valor objetivo, son implementaciones de problemas de clasificación.
- Random Forests: Son una extensión de los árboles de decisión. Para este caso, se usan en conjunto para realizar predicciones. Por ser un colectivo son más estables y fiables.
- Algoritmos genéticos: Son algoritmos meta heurísticos usados en problemas de optimización con y sin restricciones.

#### F. URL Spoofing [3]

URL (Uniform Resource Locator) es el formato utilizado universalmente en la Internet para representar la dirección de un recurso que se encuentra en ella (una página HTML, un documento CSS, una imagen, etc). [2]

La alteración de URLs (o URL spoofing) es una práctica fraudulenta que consiste en alterar la apariencia de una URL original que apunta al recurso oficial que se está buscando,

para redirigir de alguna manera a la víctima y hacerla acceder a algún recurso no legítimo diseñado para atacarle.

A veces es suficiente solo con dar click a una URL maliciosa para infectar un dispositivo. Otras, se trata de un ataque más sofisticado que conlleva más pasos y artefactos diseñados y puestos en marcha para obtener algún beneficio (robar una identidad, obtener credenciales, obtener información sensible, etc).

#### G. Recopilación y Análisis de los Datos

Dataset:

Las fuentes de datos provienen principalmente del informe de detección de intrusiones de DARPA de 1998. Programa de evaluación del MIT Lincoln Labs. Este conjunto de datos contiene una variedad de eventos de red que se han simulado en el entorno de la red militar. Los datos son TCP Dump que se ha acumulado desde la red de área local de un entorno de la Fuerza Aérea. Los datos están plagados de múltiples ataques.

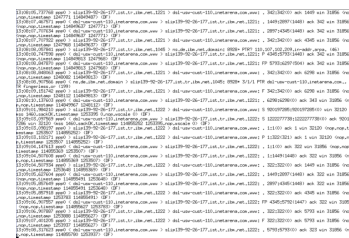


Fig. 1: [1]

Este conjunto de datos representa en su mayoría estas 4 categorías de ataques a red comúnmente conocidos:

- Ataques de denegación de servicio (DOS): una forma más avanzada de este ataque se llama ataque de denegación de servicio distribuido (DDoS).
- Ataques de fuerza bruta: son accesos no autorizados desde una máquina remota.
- Ataques Buffer overflow: son ataques a memoria que dan accesos no autorizados a privilegios de superusuario local (root).
- Ataques de reconocimiento: se ocupan de la vigilancia de sondeo y el escaneo de puertos.

Librerías importadas : Usamos paquetes de machine learning/ciencia de datos como numpy, sklearn, pandas y matplotlib para la visualización:

```
from time import time
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import cross_val_score
```

Fig. 2: [1]

Para implementar isolation forest, utilizamos el paquete sklearn.ensemble

```
from sklearn.ensemble import IsolationForest
```

Fig. 3: [1]

El siguiente código importa los paquetes relevantes y carga los datos (KDD):

```
from sklearn.metrics import roc_curve, auc
from sklearn.datasets import fetch_kddcup99
import matplotlib.pyplot as plt

dataset = fetch_kddcup99(subset=None, shuffle=True, percent=10=True)
# http://www.kdd.org/kdd-cup/view/kdd-cup-1999/tasks
X = dataset.data
y = dataset.target
```

Fig. 4: [1]

### Características de los datos:

Los datos almacenados en nuestro dataset conservan las siguientes características.

Feature name	Description	Type
duration	Length (number of seconds) of the connection	continuous
protocol_type	Type of the protocol, for example, tcp, udp, and so on	discrete
service	Network service on the destination, for example, http, telnet, and so on	discrete
succ_bytes	Number of data bytes from source to destination	continuous
dst_bytes	Number of data bytes from destination to source	continuous
flag	Normal or error status of the connection	discrete
land	1 if connection is from/to the same host/port; 0 otherwise	discrete
wrong_fragment	Number of wrong fragments	continuous
is_posit	Number of input packets	continuous

Fig. 5: [1]

En el dataset se representan como.

```
feature_cols = ['duration', 'protocol_type', 'service', 'flag',
                'succ_bytes', 'dst_bytes', 'land', 'wrong_fragment', 'urgent', 'hot',
                'num_failed_logins', 'logged_in', 'num_compromised', 'root_shell',
                'su_attempted', 'num_root', 'num_file_creations', 'num_shells',
                'num_access_files', 'num_outbound_cmds', 'is_host_login', 'is_guest_login',
                'count', 'srv_count', 'serror_rate', 'srv_serror_rate', 'error_rate',
                'srv_error_rate', 'same_srv_rate', 'diff_srv_rate', 'srv_diff_host_rate',
                'dst_host_count', 'dst_host_srv_count', 'dst_host_same_srv_rate',
                'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate',
                'dst_host_diff_src_port_rate', 'dst_host_error_rate',
                'dst_host_srv_error_rate', 'dst_host_error_rate',
                'dst_host_srv_error_rate']
X = pd.DataFrame(X, columns = feature_cols)
y = pd.Series(y)
X.head()
```

Fig. 6: [1]

El código anterior mostrará las primeras filas de la tabla con todos los nombres de las columnas. Luego convertimos las columnas en flotantes para un procesamiento eficiente.

```
for col in X.columns:
    try:
        X[col] = X[col].astype(float)
    except ValueError:
        pass
```

Fig. 7: [1]

Convertimos lo categórico en variables ficticias o indicadores:

```
X = pd.get_dummies(X, prefix=['protocol_type_', 'service_', 'flag_'],
                    drop_first=True)
X.head()
```

Fig. 8: [1]

Ya con todo esto es posible generar los recuentos.

Al ejecutarse, el código anterior muestra alrededor de 5 filas × 115 columnas:

Ajustamos un árbol de clasificación con max\_profundidad = 7 en todos los datos de la siguiente manera

El resultado del ajuste del modelo anterior es el siguiente:

Análisis: Se implementó una técnica de minería de datos (KDD: Knowledge Discovery in Databases), en donde el objetivo fue aislar observaciones seleccionando features de la data al azar, y dividiendo en un valor entre el máximo y el mínimo. El resultado de este proceso es un árbol, donde el número de divisiones que llevan a aislar una muestra equivale a la longitud desde la raíz hasta el nodo terminal. Esta longitud, promediada en el conjunto de los random trees, es una medida de normalidad: Es nuestra función de decisión.

Se organizan los datos, dividiéndolos en train y test, y se aplican estándares como verificar la precisión de nulidad.

```
y.value_counts()
```

```
Out:
smurf.          280790
neptune.        107201
normal.         97278
back.           2203
satan.          1589
ipsweep.        1247
portsweep.      1040
warezclient.    1020
teardrop.       979
pod.            264
nmap.           231
guess_passwd.   53
buffer_overflow. 30
land.           21
warezmaster.    20
imap.           12
```

```
rootkit.        10
loadmodule.     9
ftp_write.      8
multihop.       7
phf.            4
perl.           3
spy.            2
dtype: int64
```

Fig. 9: [1]

```
from sklearn.tree import DecisionTreeClassifier, export_graphviz
treeclf = DecisionTreeClassifier(max_depth=7)
scores = cross_val_score(treeclf, X, y, scoring='accuracy', cv=5)
print np.mean(scores)
treeclf.fit(X, y)
```

Fig. 10: [1]

```
0.9955204407492013
```

Fig. 11: [1]

## H. Entrenamiento

El modelo fue entrenado con una base de datos del Defense Advances Research Projects Agency - DARPA conteniendo una gran variedad de eventos de red simulados en un entorno militar. Este conjunto de datos es analizado con árboles de decisión los cuales identifican los datos recolectados en los TCP Dump entre maliciosos y no maliciosos.

Existen dos categorías de árboles, estas van en función de su objetivo

- **Árbol de decisión de variable categórica:** Las variables categóricas, a diferencia de las numéricas, representan atributos que no pueden expresarse en términos de cantidades o números continuos. En lugar de ello, estas variables representan distintas categorías o grupos. Al emplear un árbol de decisión para analizar conjuntos de datos con variables categóricas, cada nodo de decisión se divide con base en una categoría específica. Esta división es guiada por preguntas del tipo "¿pertenece el dato a la categoría A o a la categoría B?". La respuesta a cada una de estas preguntas es binaria, es decir, sí o no. Esta estructura de decisión, representada por los nodos del árbol, permite clasificar eficientemente los datos en

función de las distintas categorías, lo que simplifica el proceso de toma de decisiones.

- **Árbol de decisión de variable continua:** las variables continuas representan atributos que pueden expresarse en términos de cantidades o números continuos. Cuando nos adentramos en el análisis de árboles de decisión que involucran variables continuas, la dinámica de toma de decisiones se modifica para adaptarse a la naturaleza numérica de estas características. En lugar de simplemente dividir los nodos en categorías, las divisiones se realizan a lo largo de un rango numérico. Cada nodo ahora se pregunta: "¿El valor de la variable continua está por encima o por debajo de un determinado umbral?". La respuesta sigue siendo binaria, pero el proceso de determinar la ubicación del umbral óptimo es crucial y se basa en criterios específicos como la ganancia de información o la reducción de la impureza de Gini.

### I. Coeficiente de Gini

La reducción de la impureza de Gini es utilizada por el algoritmo de árboles de decisión para evaluar qué división resulta en una mayor homogeneidad en los nodos hijos. Una reducción mayor indica una mejor división, ya que se está logrando una mayor separación entre las clases en los nodos resultantes. En cada paso, el árbol seleccionará la división que maximice esta reducción de la impureza de Gini. Este proceso se repite recursivamente durante la construcción del árbol hasta que se cumple un criterio de parada.

$$Gini(D) = 1 - \sum_{i=1}^c p_i^2$$

Donde:

- $D$  es el conjunto de datos del nodo.
- $c$  es el numero de clases en el conjunto de datos.
- $p_i$  es la proporción de elementos en el conjunto de datos que pertenecen a la clase  $i$ .

En el siguiente caso se usará un clasificador de árbol de clasificación. El objeto que se definió anteriormente recibe los

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=7,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False, random_state=None,
splitter='best')
```

Fig. 12: [1]

siguientes parámetros:

- **class\_weight:** Define el peso de las clases en el modelo. Si es None, todas las clases tienen el mismo peso.
- **criterion:** Especifica la función para medir la calidad de una división. En este caso, se utiliza la impureza de Gini, ya que el valor es 'gini'.
- **max\_depth:** Establece la profundidad máxima del árbol. En este caso, la profundidad máxima es 7, lo que significa que el árbol se construirá hasta una profundidad máxima de 7 niveles.

- **max\_features:** Indica el número de características a considerar al buscar la mejor división. Si es None, se consideran todas las características.
- **max\_leaf\_nodes:** Limita el número máximo de nodos hoja en el árbol. Si es None, no hay límite.
- **min\_impurity\_decrease:** Requiere una cierta cantidad de disminución de la impureza para dividir un nodo.
- **min\_impurity\_split:** Este parámetro ya no se utiliza y se mantiene por razones de compatibilidad. En versiones más recientes de scikit-learn, se ha eliminado.
- **min\_samples\_leaf:** Establece el número mínimo de muestras requeridas para estar en un nodo hoja.
- **min\_samples\_split:** Establece el número mínimo de muestras requeridas para dividir un nodo interno.
- **min\_weight\_fraction\_leaf:** La fracción mínima ponderada de la suma total de pesos (de todas las muestras de entrada) requerida para estar en un nodo hoja.
- **presort:** Este parámetro ya no se utiliza y se mantiene por razones de compatibilidad.
- **random\_state:** Si se proporciona un valor, asegura reproducibilidad.
- **splitter:** Especifica la estrategia utilizada para elegir la división en cada nodo. En este caso, se utiliza 'best', lo que significa que la mejor división se elige según la impureza.

Finalmente se realiza la ejecución del siguiente comando:

```
pd.DataFrame({'feature':X.columns,
'importance':treeclf.feature_importances_}).sort_values('importance',
ascending=False).head(10)
```

Fig. 13: [1]

Este comando crea un DataFrame que muestra las 10 características con la mayor importancia, ordenadas de mayor a menor importancia, a partir del modelo de árbol de decisión anteriormente mencionado.

El DataFrame obtenido es el siguiente:

Número	Característica	Importancia
20	srv_count	0.633722
25	same_srv_rate	0.341769
9	num_compromised	0.013613
31	dst_host_diff_srv_rate	0.010738
1	src_bytes	0.000158
85	service_red_i	0.000000
84	service_private	0.000000
83	service_printer	0.000000
82	service_pop_3	0.000000
75	service_netstat	0.000000

### J. Pruebas

Detección de Anomalías con KDD: Usamos datos binarios donde 1 representa un ataque no-normal, y verificamos la precisión.

El output de los cinco primeros elementos se ve así:

Organizamos los datos, declaramos el modelo IsolationForest y ejecutamos model fit

Hacemos una predicción:



```

0.1
[[1190 15]
 [ 15 30]]

0.2
[[1201 4]
 [ 17 28]]

0.3
[[1204 1]
 [ 22 23]]

0.4
[[1205 0]
 [ 25 20]]

0.5
[[1205 0]
 [ 27 18]]

0.6
[[1205 0]
 [ 28 17]]

0.7
[[1205 0]
 [ 29 16]]

0.8
[[1205 0]
 [ 29 16]]

```

Fig. 27: [1]

[illegible]

Fig. 28: [1]

```

treeuid = CreateFromIndex(index_name, begin);
tree_pipe = Pipeline::Init("root", root, "Node()", treeuid);
root = CreateValueFromIndex(index_name, root_name);
source = CreateValueFromIndex(index_name, source_name, root, tree_pipe, 0, 0);
graph = Graph::Create(index_name, tree_pipe, 0, 0);
tree_pipe.Finish(0, 0);
export_graphviz(tree_pipe.steps[1][1], root, "tree_graphviz.dot",
    "tree_graphviz.png");

```

Fig. 29: [1]

The tree diagram below shows how the decision logic for maliciousness detection works.

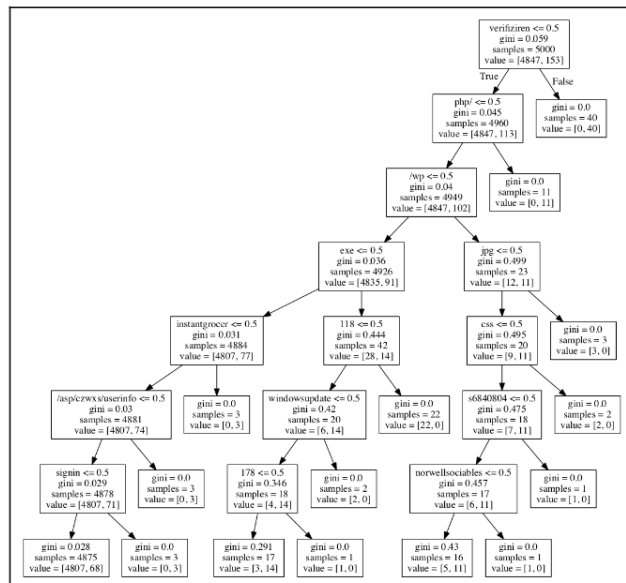


Fig. 30: Figura 1: Diagrama que muestra cómo funciona la lógica de decisión de maliciosidad de nuestro modelo clasificador [1]

### III. ANÁLISIS Y DESARROLLO

Este trabajo consiste en la implementación de un modelo clasificador de actividad maliciosa en redes de computadores.

y de URLs, mediante el análisis, procesamiento y entrenamiento basado en muestras de tráfico TCP.

## IV. CONCLUSIONES

Se estudiaron diferentes tipos de datos maliciosos junto con inyecciones maliciosas en sensores inalámbricos. También se abordaron los diferentes tipos de árboles de decisión, que incluyeron árboles de decisión para variables categóricas y continuas. Se concluyó con una aplicación en la detección de URLs maliciosas mediante árboles de decisión.

Lo que sigue, y está un nivel más allá del alcance de este capítulo en el libro [1] es aprender cómo se pueden atrapar a los impostores y hackers con las manos en la masa.

## REFERENCIAS

- [1] Halder, S., & Ozdemir, S. (2018). *Hands-On Machine Learning for Cybersecurity* (1.<sup>st</sup> ed., pp. 159–182). Birmingham: Packt Publishing.
- [2] Mozilla Foundation *What is a URL? MDN Web Docs* Available at: <https://developer.mozilla.org/en-US/docs/Learn/>
- [3] NordVPN Foundation *Web DocsURL Spoofing: Definition and explanation* Available at: <https://nordvpn.com/es/blog/url-spoofing/>