UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA DE SISTEMAS DE TELECOMUNICACIÓN

Trabajo Fin de Grado

IMPLEMENTACIÓN DE ALGORITMOS DE MACHINE LEARNING PARA DETECTAR TRÁFICO DE RED ILÍCITO

Autor: Iván Gómez Carretero

Tutor: Miguel Ángel Sicilia Urbán

Cotutor: Antonio Agustín Pastor Perales (Telefónica I+D)

TRIBUNAL:

Presidente:

Vocal 1º:

Vocal 2º:

FECHA:

1 ÍNDICE

2 RESUMEN

2.1 RESUMEN

La tecnología machine learning o aprendizaje automático tiene un potencial sumamente grande pudiéndose implementar en infinidad de casos. Nuestro caso que nos involucra es la ciberseguridad, el cual es un tema muy importante en la actualidad. Si no se trata con delicadeza estará en peligro la seguridad de todos los ciudadanos.

Este trabajo tendrá como enfoque detectar actividades maliciosas en la red mediante técnicas de machine learning para la detección de anomalías. El tráfico de red donde se producirán los actos ilícitos lo obtendremos en formato netflow, a estos datos le aplicaremos diferentes técnicas de procesamiento desarrolladas para una mejor asimilación (Big Data) y le aplicaremos diferentes algoritmos de machine learning no supervisados empleados para la detección de anomalías.

Una vez aplicado este proceso en diferentes pruebas de tráfico malicioso, obtendremos diferentes métricas para estudiar en cuál de ellos obtenemos mejores resultados de detección.

Palabras clave:

- Machine learning
- Ciberseguridad
- Netflow
- Detección anomalías
- Procesamiento datos

2.2 RESUMEN INGLÉS

The machine learning technology has an enormous potential and can be used in many cases. Our case that involves us is cybersecurity, which is a very important issue at present. If not treated delicately, the safety of all citizens will be in danger.

This work will focus on detecting malicious activities in the network using machine learning techniques for the detection of anomalies. The Illegal network traffic will be obtained in netflow format, to this data we will apply different processing techniques developed for a better assimilation (Big Data) and we will apply different non-supervised machine learning algorithms used for the detection of anomalies.

Once this process is applied to different malicious traffic tests, we will take different metrics to study in which we obtain better detection results.

3 INTRODUCCIÓN

El machine learning o aprendizaje automático en su definición más sencilla tiene como fin conquistar aquellos retos que las capacidades humanas y los sistemas estáticos no pueden alcanzar.

Se trata de una rama de la llamada inteligencia artificial basada en parámetros estadísticos y su función es aplicar los diferentes modelos y algoritmos que aprenden de los datos que tenemos para obtener cada vez mejores soluciones a un problema. Por tanto podemos decir que el objetivo principal de este proceso de aprendizaje es utilizar evidencias conocidas para poder crear una hipótesis con la que dar respuesta a nuevas situaciones no conocidas[1].

En los tiempos que corren la ciberseguridad es uno de los temas más importantes para la sociedad ya que acarrea pérdidas millonarias si no se gestiona correctamente. Para combatirlo se han desarrollado diferentes tecnologías, entre las que destaca la que acabamos de citar, el machine learning.

Con esta técnica se pretende poseer un elemento de choque contra estos ataques más dinámico, que consiga aprender por el mismo los diferentes ataques para después poder detectarlos.

La industria de la seguridad informática entra de lleno aquí para combatir a los cibercriminales ofreciendo la mayoría de proveedores de seguridad productos de machine learning[2].

Los métodos de Machine Learning desempeñan un papel clave en la construcción de perfiles de comportamientos normales en un sistema y los de detección de intrusiones. En general, los datos correspondientes a comportamientos usuales son fáciles de extraer y etiquetar, pero también necesitamos ejemplos de ataques, los cuales son más difíciles de conseguirán entornos reales. Adicionalmente, vivimos en un entorno de red cambiante, por lo que los tráficos que se pueden considerar como normales pueden llegar a ser tratados como anomalías en un futuro cercano[3].

Por tanto, la problemática en la que vamos a utilizar el machine learning es en la ciberseguridad como hemos comentado, intentaremos anticiparnos a los ciberdelincuentes detectando comportamientos anómalos mediante estas técnicas.

Para ello necesitaremos tráfico de red que generaremos en entornos controlados intentando que sean lo más real posible. Estos datos de tráfico generados para poder tratarlos necesitamos exportarlos de alguna manera legible, aquí entra en juego uno de los temas importantes de nuestro trabajo, el protocolo netflow. Este protocolo de red fue desarrollado por Cisco y permite la recolección de tráfico de red.

Este protocolo es importante ya que nos va a permitir exportar ese tráfico recolectado en diferentes formatos, en nuestro caso será en formato separados por comas (csv). Gracias a este procesamiento generaremos un dataset donde podremos realizar técnicas de procesamiento Big data y aplicar algoritmos de machine learning.

A parte de para el caso de ciberseguridad que nos atrae, el machine learning se utiliza en infinidad de casos, por ejemplo, diferentes proveedores lo utilizan para recomendar sus productos, los servicios de correo electrónico lo usan para filtrar spam, aplicaciones tanto para

reconocimiento facial y de emociones como reconocimiento de voz, o las diferentes redes sociales lo utilizan para monitorizar opiniones de los diferentes usuarios entre otros usos[4].

3.1 OBJETIVOS

Con la realización del presente trabajo de fin de grado se quieren alcanzar los siguientes objetivos:

- Generar tanto tráfico de red normal de usuarios como tráfico malicioso para poder aplicar sobre ellos los algoritmos machine learning.

- Utilizar el protocolo de red NetFlow y aplicar las diferentes herramientas para su uso (nfdump, softflowd…). Con ello recolectaremos el tráfico que circula por la red en formato csv.

- Estudiar los diferentes algoritmos que se pueden aplicar en machine learning, entrando en detalle en los algoritmos empleados para la detección de anomalías, y elegir cuáles serán los más adecuados para un estudio comparativo.

- Desarrollar diferentes clases y funciones en Python que permitan una mayor facilidad al procesar los datos (Big data) que vayamos a pasarle al algoritmo.

- Realizar un estudio comparativo entre los algoritmos selecciones como más adecuados para la detección de anomalías en diferentes casos de ciberseguridad.

- Conseguir detectar tráfico anómalo de diferentes familias con unas tasas altas para algún algoritmo de detección de anomalías.

4 ESTADO DEL ARTE

4.1 MACHINE LEARNING

Se le llama machine learning a la detección automática de patrones significativos en un conjunto de datos. También se podría decir que se trata de convertir la experiencia en conocimiento. En los últimos años se ha convertido en una herramienta casi imprescindible cuando se requiere extraer información de un gran conjunto de datos[5].

Estamos rodeados de esta tecnología como podemos ver en los motores de búsqueda que aprenden nuestros mejores resultados, en el software anti-spam en los correos electrónicos, en transacciones de tarjetas de crédito segura, etc.

Una de las características más comunes del machine learning es su utilización cuando una persona humana no puede proporcionar una solución concreta, debido a la complejidad de los patrones desarrollados. Esto contrasta con el uso tradicional de los ordenadores, en el cual el usuario decide. Esto se puede conseguir gracias a que muchas de nuestras habilidades se adquieren a través del aprendizaje de nuestras experiencias, lo que se ha aplicado a las máquinas[6].

Esta tecnología tiene como motivo de su utilización la popularidad que ha adquirido en los últimos años, ya que como hemos comentado a partir de un conjunto de datos se puede aprender de ellos sin que existan técnicas de programación que den órdenes a la máquina para tratar esos datos. Para conseguir esto la máquina deberá detectar ciertos patrones en los datos que se escapan a la percepción humana. El fin de todo será a la hora de incluir nuevas grandes cantidades de datos, cuando la máquina deberá ofrecer resultados precisos en un tiempo corto a partir del aprendizaje adquirido[7].

4.1.1 Tipos de aprendizaje:

Existen diferentes tipos de aprendizajes a la hora de emplear los diferentes algoritmos[6]:

- Aprendizaje supervisado: Se basa en enseñar con anterioridad características o comportamientos a seguir en un campo objetivo para que en el futuro sea más fácil la predicción de ese campo con datos nuevos de entrada. Existen muchos ejemplos dispares entre ellos que ilustran este caso, desde por ejemplo detectar correos spam que han sido previamente etiquetados otros similares, o hasta predecir el valor de la luz en mes determinado a partir de ciertos datos proporcionados en otros meses.

- Aprendizaje no supervisado: En este caso se va a obtener resultados sin tener definidos patrones o comportamientos a seguir anteriormente, por lo que va a estar más enfocado a detectar patrones fuera de lo normal en nuestros datos. En este caso podemos citar como ejemplo el estudio que nos incumbe, en el cual detectamos tráfico de red anómalo.

4.1.2 Tipos algoritmos:

Hay infinidad de algoritmos que se pueden utilizar, pero destacamos los siguientes tipos más empleados[8]:

- Algoritmos predicción numérica

  Son algoritmos supervisados usados para predecir el valor de un campo numérico usando los valores de otros campos de ese evento. Destacan:

  - LinearRegression.
  - Lasso.
  - Ridge.
  - ElasticNet.
  - KernelRidge.
  - SDGRegressor (Stochastic gradient descent).
  - DecisionTreeRegressor.
  - RandomForestRegressor.
  - XGBRegressor (Gradient bosting).

- Algoritmo predicción categórica

  Son supervisados también y se utilizan para predecir el valor de un campo categórico usando los valores de otros campos de ese evento. Tenemos los siguientes:

  - LogisticRegression.
  - SVM (Suport vector machine).
  - BernoulliNB.
  - GaussianNB.
  - SGDClassifier(Stochastic gradient descent).
  - DecisionTreeClassifier.
  - RandomForestClassifier.

- Algoritmos previsión

  Igual que lo anteriores son algoritmos supervisados usados para predecir valores futuros dados valores pasados de una métrica (series de tiempo numéricas).

  - ARIMA (Autoregressive integrated moving average).

- Algoritmos detección anomalía

  En este caso son algoritmos no supervisados y se emplean cuando se quiere encontrar eventos que contienes combinaciones inusuales de valores. Entre los más importantes tenemos:

  - GaussianMixture
  - BGM (Bayesian Gaussian Mixture)
  - KDE (Kernel Density)
  - EllipticEnvelope. Scikit. Outlier detection.
  - OneClassSVM (One class suport vector machine).
  - LOF (local outlier factor).

- IForest (Isolation Forest).

- Algoritmos de cluster numérico

  Son no supervisados y se utilizan para predecir diferentes eventos en varios clusters numéricos.

  - K-means.
  - KNeighbors.
  - DBSCAN (Density-Based Spatial clustering of applications with noise).
  - Birch.
  - SpectralClustering.

4.2 NETFLOW

Netflow es un protocolo de red que permite recolectar información del tráfico IP que circula por la red. Fue creado por Cisco y es soportado por todos los switches y routers Cisco. Como veremos más adelante se han desarrollado diferentes herramientas para que fabricantes (Juniper, Enterasys Switches …) o S.O (Linux, FreeBSD …) puedan soportar netflow[9].

Este protocolo abierto es muy útil para los técnicos de red, ya que aparte de recolectar el tráfico de diferentes dispositivos permite monitorizar y representar el tráfico de red en tiempo real.

Este tráfico se exportará en los llamados flujos, que es una secuencia unidireccional de paquetes con ciertas características comunes (IP origen, IP destino, protocolo, puertos…). Dependiendo de la versión de netflow estos campos variaran como veremos más adelante.

Las versiones de netflow desarrolladas han sido:

- Netflow v1
- Netflow v5: Versión más utilizada.
- Netflow v6
- Netflow v7
- Netflow v8
- Netflow v9: Será la versión que utilicemos ya que nos la dará la plantilla nfdump.

Unos de los aspectos a destacar en netflow y en nuestro caso afectará a la hora de detectar anomalías es que los flujos netflow no contienen información sobre los usuarios sino solo datos de la conexión, únicamente llegan hasta la capa 3 en el sistema OSI por así decirlo. Gracias a esto evitaremos problemas relacionados con la privacidad de los usuarios.

Por tanto en resumen podemos decir que el protocolo netflow tiene una gran utilidad en seguridad informática, tanto en la detección en tiempo real de ataques y anomalías como en la investigación de incidentes (análisis forense)[10].

Respecto a netflow existen diferentes versiones creadas hasta la fecha como hemos comentado antes, entre las que destacamos las siguientes:

4.2.1 Netflow v1

Es la primera versión desarrollada por Cisco[11].

Paquete:

CABECERA PAQUETE
Registro conjunto de flujos

- Formato cabecera del paquete:

BYTES	CAMPO	DESCRIPCIÓN
0-1	version	Versión formato NetFlow.
2-3	count	Número de conjunto de flujos exportados.
4-7	sys_uptime	Hora en milisegundos desde que se inició la exportación del paquete.
8-11	unix_secs	Cantidad de segundos desde 0000 UTC 1970.
12-16	Unix_nsecs	Cantidad de nanosegundos residuales desde 0000 UTC 1970.

- Formato registro conjunto de flujos:

CAMPO	DESCRIPCIÓN
srcaddr	Dirección IP origen.
dstaddr	Dirección IP destino.
nexthop	Dirección IP siguiente router próximo salto.
input	Interfaz entrada.
output	Interfaz salida.
dPkts	Paquetes en el flujo.
dOctets	Número bytes en los paquetes del flujo.
first	Fecha inicio del flujo
last	Fecha fin del flujo.
srcport	Puerto origen.
dstport	Puerto destino.
pad1	Bytes no utilizados.
prot	Protocolo.
tos	Tipo servicio IP.
flags	Flags utilizados.
pad2	Bytes no utilizados.
reserved	Bits reservados

4.2.2 Netflow v5

Fue de las versiones que más éxito tuvo antes de dar entrada a la versión 9[12].

Paquete:

CABECERA PAQUETE
Registro conjunto de flujos

- Formato cabecera del paquete:

BYTES	CAMPO	DESCRIPCIÓN
0-1	version	Versión formato NetFlow.
2-3	count	Número de conjunto de flujos exportados.
4-7	sys_uptime	Hora en milisegundos desde que se inició la exportación del paquete.
8-11	unix_secs	Cantidad de segundos desde 0000 UTC 1970.
12-15	Unix_nsecs	Cantidad de nanosegundos residuales desde 0000 UTC 1970.
16-19	package_sequence	Contador de secuencias de todos los paquetes exportados enviados por el dispositivo.
20	engine_type	Tipo máquina.
21	engine_id	ID máquina.
22-23	sampling_interval	Modo y valor del intervalo de muestreo.

- Formato registro conjunto de flujos:

CAMPO	DESCRIPCIÓN
srcaddr	Dirección IP origen.
dstaddr	Dirección IP destino.
nexthop	Dirección IP siguiente router próximo salto.
input	Interfaz entrada.
output	Interfaz salida.
dPkts	Paquetes en el flujo.
dOctets	Número bytes en los paquetes del flujo.
first	Fecha inicio del flujo
last	Fecha fin del flujo.
srcport	Puerto origen.
dstport	Puerto destino.
pad1	Bytes no utilizados.
tcp_flags	Flags TCP

prot	Protocolo.
tos	Tipo servicio IP.
src_as	Número de sistema autónomo BGP origen.
dst_as	Número de sistema autónomo BGP destino.
src_mask	Máscara de red origen.
dst_mask	Máscara de red destino.
pad2	Bytes no utilizados.

4.2.3 NetFlow v9

A diferencia de las demás versiones NetFlow, esta última versión tiene como característica distintiva que está basada en plantillas. Las plantillas van a permitir organizar los flujos y tener definidos los campos que queremos obtener[13].

El formato de registro NetFlow v9 se compone de un encabezado de paquete seguido por una o más plantillas de un conjunto de flujos de datos. Estas plantillas van a proporcionar una descripción de los campos que estarán presentes en los datos del conjunto de flujos[14].

Paquete:

CABECERA PAQUETE
Plantilla conjunto de flujos
Datos conjunto de flujos
Datos conjunto de flujos
…
Plantilla conjunto de flujos
Datos conjunto de flujos
…

- Formato cabecera del paquete:

BYTES	CAMPO	DESCRIPCIÓN
0-1	version	Versión formato NetFlow.
2-3	count	Número de conjunto de flujos exportados.
4-7	sys_uptime	Hora en milisegundos desde que se inició la exportación del paquete.
8-11	unix_secs	Cantidad de segundos desde 0000 UTC 1970.
12-15	package_sequence	Contador de secuencias de todos los paquetes exportados enviados por el dispositivo.
16-19	source_id	Garantiza la unicidad de todos los flujos exportados.

- Formato registro plantilla conjunto de flujos[15]:

BIT 0-15
flowset_id = <255
length
template_id
field_count
field_1_type
field_1_length
field_2_type
field_2_length
…
field_N_type
field_N_length
template_id
field_count
field_1_type
field_1_length
…
field_N_type
field_N_length

Vemos el significado de cada uno de estos campos.

CAMPO	DESCRIPCIÓN
flowset_id	Utilizado para distinguir los registros de plantilla de los registros de datos.
length	Longitud total del conjunto de flujos en bytes. El valor de la longitud se debe utilizar para determinar la posición del siguiente registro de conjunto de flujos, que podría ser plantilla del conjunto de flujos o datos del conjunto de flujos.
template_id	Se asignará un ID único a cada plantilla.
field_count	Número de campos existentes en esta plantilla.
field_type	Representa el tipo de campo. Estos valores serán específicos del proveedor, los cuales se elegirán de los campos conocidos que proporciona Cisco.
field_length	Proporciona la longitud del campo field_type en bytes.

A continuación, vemos los campos (field_type) que utiliza la platilla que contiene nfdump, los cuales son los que obtendremos en nuestras trazas de tráfico[16].

CAMPO	DESCRIPCIÓN
FIRST_SEEN	Fecha inicio flujo.
LAST_SEEN	Fecha fin flujo.
DURATION	Duración flujo
IPV4_SRC_ADDR	Dirección IPv4 origen.
IPV4_DST_ADDR	Dirección IPv4 destino.
L4_SRC_PORT	Puerto origen capa 4.
L4_DST_PORT	Puerto destino capa 4.
PROTOCOL	Protocolo.
TCP_FLAGS	Flags TCP
SRC_TOS	Tipo de configuración del byte de servicio origen.
IN_PKTS	Número paquetes entrada.
IN_BYTES	Número bytes entrada.
OUT_PKTS	Número paquetes salida.
OUT_BYTES	Número paquetes salida.
INPUT_SNMP	Interface entrada.
OUTPUT_SNMP	Interface salida.
SRC_AS	Número de sistema autónomo BGP origen.
DST_AS	Número de sistema autónomo BGP destino.
SRC_MASK	Máscara de red origen.
DST_MASK	Máscara de red destino
DST_TOS	Tipo de configuración del byte de servicio destino.
DIRECTION	Dirección flujo.
IPV4_NEXT_HOP	Dirección IPv4 del router del próximo salto.
BGP_IPV4_NEXT_HOP	Dirección IPv4 del router del próximo salto en el dominio BGP.
SRC_VLAN	VLAN origen.
DST_VLAN	VLAN destino.
IN_SRC_MAC	Dirección MAC origen de entrada.
OUT_DST_MAC	Dirección MAC destino de salida.
IN_DST_MAC	Dirección MAC destino de entrada.
OUT_SRC_MAC	Dirección MAC origen de salida.
MPLS_LABEL_1	Etiqueta MPLS posición 1.
MPLS_LABEL_2	Etiqueta MPLS posición 2.
MPLS_LABEL_3	Etiqueta MPLS posición 3.
MPLS_LABEL_4	Etiqueta MPLS posición 4.
MPLS_LABEL_5	Etiqueta MPLS posición 5.
MPLS_LABEL_6	Etiqueta MPLS posición 6.
MPLS_LABEL_7	Etiqueta MPLS posición 7.
MPLS_LABEL_8	Etiqueta MPLS posición 8.
MPLS_LABEL_9	Etiqueta MPLS posición 9.
MPLS_LABEL_10	Etiqueta MPLS posición 10.
CLIENT_LATENCY	Latencia del cliente.
SERVER_LATENCY	Latencia del servidor.
APP_LATENCY	Latencia de la aplicación.
REPLICATOR_FACTOR	Multidifusión factor de replicación

ENGINE_TYPE/ENGINE_ID	Tipo máquina/ID máquina
EXPORTING_SYSTEM_ID	ID del sistema exportador
RECEIVED_TIMESTAMP	Timestamp recibido.

1- Formato registro plantilla conjunto de flujos

BIT 0-15
flowset_id = >255
length
record_1-field_1_value
record_1-field_2_value
record_1-field_3_value
…
record_2-field_M_value
record_2-field_1_value
record_2-field_2_value
record_2-field_3_value
…
record_2-field_M_value

Vemos el significado de cada uno de estos campos.

CAMPO	DESCRIPCIÓN
flowset_id	Utilizado para distinguir los registros de plantilla de los registros de datos.
length	Longitud total del conjunto de flujos en bytes. El valor de la longitud se debe utilizar para determinar la posición del siguiente registro de conjunto de flujos, que podría ser plantilla del conjunto de flujos o datos del conjunto de flujos.
record_N-field_M	Nos darán los diferentes valores de los campos.
padding	Es el relleno que se añade para alinear el conjunto de flujos a 32 bits.

4.3 DETECCIÓN DE ANOMALÍAS

La detección de anomalías se está convirtiendo en uno de los campos más importantes y de más interés dentro del machine learning.

Su funcionamiento se basa en la suposición de que el comportamiento malicioso que queremos que sea detectado como anomalía será significativamente diferente de los diferentes comportamientos lícitos, lo que supondrá una fácil detección[17].

Uno de los mayores problemas a los que se enfrenta esta técnica es la reducción de las falsas alarmas durante el proceso de detección de patrones de ataques desconocidos, según se comporte el algoritmo respecto a dicho valor mejoraran o empeoraran los resultados.

En el ámbito de la ciberseguridad la detección de anomalías es una de las piezas más importantes en el sistema de defensa, ya que muchos sistemas y aplicaciones se han desarrollado sin tener en cuenta la seguridad, por lo que pueden tener defectos o errores en su diseño que pueden ser explotados por intrusos para atacar. La detección de anomalía no sustituirá a una buena prevención de un sistema, sino que se complementa esos sistemas de seguridad para un mayor refuerzo mejorando la seguridad del sistema.

En general en este ámbito los sistemas de detección de intrusiones ilícitas se clasifican en dos tipos:

- Sistema mediante detección de firmas
- Sistema mediante detección de anomalías

Como diferencias podemos saber que el un sistema de detección de firmas identifica patrones maliciosos que han sido comparados con unos patrones maliciosos dados, mientras que en el caso de la detección de anomalías no se necesita comparar con datos maliciosos, sino que compara con actividades normales para buscar la rareza.

Tanto un sistema como otro van a tener sus ventajas e inconvenientes, en el caso de la detección de firmas tendremos como principal ventaja de detectar ataques conocidos con una fiabilidad muy alta, dando lugar al inconveniente en el caso de que el ataque no sea conocido en el cual el sistema fallara[18].

Mientras para de la detección de anomalías tenemos como ventajas que podremos detectar ataques maliciosos de cualquier tipo sin tener que conocerlos previamente ya que se puede modelar la capacidad de un sistema y con ello detectar desviaciones en ese modelo, también tendrán la capacidad de personalización de los perfiles de actividad normal para cada sistema. Esto supondrá una dificultad para el atacante ya que no sabrá que actividades se pueden llevar a cabo sin ser detectado. Sin embargo, este enfoque tiene sus inconvenientes tales como la complejidad del sistema, las falsas alarmas…

4.3.1 Netflow aplicado a la detección de anomalías

Netflow aplicado a la detección de anomalías nos proporciona una serie de ventajas frente a diversas técnicas utilizadas en ciberseguridad para la detección de ataques como IDSs, Firewalls, antivirus…

Las ventajas que nos proporciona son las siguientes[10]:

- Como muchos dispositivos a parte de la familia Cisco permiten exportar flujos netflow, podremos exportar tráfico desde cualquier router o Switch de nuestra infraestructura, no solo desde donde se ubican IDSs o firwalls.

- Como hemos comentado en puntos atrás, a diferencia de con las IDSs o firewalls, con netflow no se tiene acceso a información confidencial del usuario, ya que los flujos solo contienen datos de la conexión.

- Si se tienen grandes datos de tráfico es recomendable usar netflow ya que al solo contener perfiles de tráfico la trasmisión de paquetes es más rápida.

- Es especialmente útil a la hora de detectar ataques “0 day” o versiones modificadas de estos ataques, ya que estamos basando nuestro estudio en detectar anomalías de cualquier tipo, no en una detección basada en firmas que no valdría para estos casos.

Cabe apreciar que la labor de detección de anomalías mediante estas técnicas debe servir como apoyo a los técnicos de seguridad para un mejor análisis del tráfico que vigilan.

4.3.2 Algoritmos clásicos en la detección de anomalías

Los algoritmos utilizados para detectar anomalías van ser no supervisados y serán los siguientes[8]:

- Basados en la densidad:
  - GaussianMixture
  - BGM (Bayesian Gaussian Mixture)
  - KDE (Kernel Density)
  - EllipticEnvelope. Scikit. Outlier detection.

- Métodos de núcleo:
  - OneClassSVM (One class suport vector machine). Scikit. Novely detection.

- Vecinos más cercanos:
  - LOF (local outlier factor). Scikit. Outlier detection.

- Partición en arboles:
  - IForest (Isolation Forest). Scikit. Outlier detection.

5 MÉTODOS

5.1 GENERACIÓN Y RECOLECTA DE DATOS NETFLOW

Ahora vamos a ver como se han generado los tráficos normales y maliciosos. Luego estos datos una vez estén generados tenemos que recolectarlos para poder utilizarlos correctamente.

5.1.1 Generación tráfico red lícito:

El tráfico de red lícito ha sido generado aprovechando el proyecto Cognet en el cual es participante Telefónica I+D. Para el proyecto Cognet se ha creado una maqueta compuesta por diferentes componentes (servidores, routers, switches…)[19].

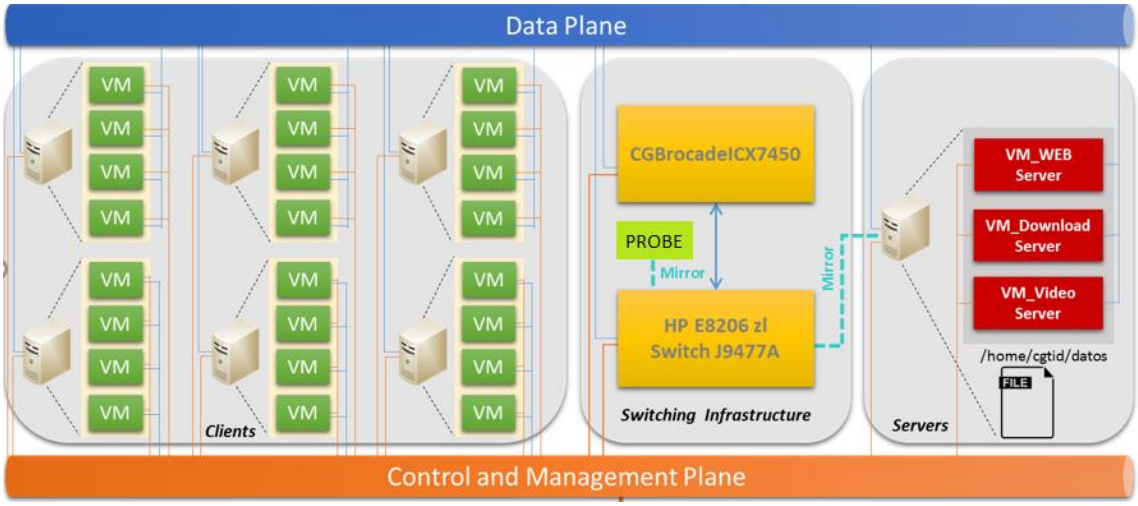

Figura 1. Mouseworld[20]

Estos componentes son los siguientes[20]:

Red de clientes: Simularan los usuarios finales que navegan por la red. Está formada por 64 clientes que estarán configurados para realizar diferentes peticiones de web, descargas y video a los servidores.

Switch BrocadeICX7450: Redirecciona el tráfico por sus puertos y gestiona las interfaces de datos.

Switch HP E8206: Redirecciona el tráfico por sus puertos y gestiona las interfaces de control.

Red de servidores: Simulan los servicios de navegación, descargas y video que utilizaran los diferentes clientes.

Probe: Se trata de una VM por la que pasará todo el tráfico entre clientes y servidores al tener un puerto espejo. En ella es donde se generarán y recolectarán los diferentes datos netflow.

5.1.2 Generación tráfico red malicioso:

El tráfico de red anómalo que mezclaremos con el tráfico normal creado por los clientes y servidores se generará a partir de diferentes archivos .pcap.

Estos archivos contendrán trazas de tráfico malicioso y se inyectarán en la interfaz de la máquina probe por la que pasa todo el tráfico lícito.

Estos tráficos maliciosos utilizados tendrán un volumen de no más de 20 flujos netflow, que se mezclarán con los miles de flujos de tráfico normal generado por los clientes.

La elección del tráfico anómalo se ha intentado dividido en varias familias con la intención de probar diferentes casos que cubriesen el mayor espectro posible de tipos de malware. Los elegidos son los siguientes:

1º Familia de tráfico malicioso:

- Troyano Bunuti mediante RIG EK.
- Dreambot en campaña HookAds mediante RIG EK.
- Troyano Chthonic campaña RoughTed mediante RIG EK.

2º Familia de tráfico malicioso:

- Jaff ransomware en archivo .wsf mediante spam en el correo.
- Jaff ransomware en archivo Word dentro de un archivo .pdf mediante spam en el correo (1º tanda).
- Jaff ransomware en archivo Word dentro de un archivo .pdf mediante spam en el correo (2º tanda).

3º Familia de tráfico malicioso:

- Wannacry ransomware mediante exploit Eternalblue.

4º Familia de tráfico malicioso:

- Ataque fuerza bruta a shh.

El funcionamiento de los diferentes tráficos se explicará con más detalle en la parte de los experimentos prácticos.

Los tráficos malignos se han obtenido de diferentes webs como:

- http://www.malware-traffic-analysis.net/
- http://www.pcapanalysis.com/
- https://isc.sans.edu/

5.1.3 Generación, recolecta y volcado de datos Netflow

El protocolo de red netflow es soportado por todos los switches y routers Cisco. Se han desarrollado diferentes herramientas para que fabricantes (Juniper, Enterasys Switches …) o S.O (Linux, FreeBSD …) puedan soportar netflow.

En la tecnología netflow vamos a distinguir tres componentes básicos en la arquitectura de análisis de tráfico[10]:

- Exportador: Será el dispositivo capaz de generar flujos netflow (router o Switch) y reenviarlos a través de UDP.

- Colector: Se trata de un dispositivo que escucha en puerto UDP determinado siendo capaz de almacenar o reenviar los flujos netflow recibidos del exportador.

- Analizador: Es el encargado de filtrar, analizar y visualizar los flujos netflow.

En nuestro caso se van a utilizar diferentes herramientas para que nuestra máquina Ubuntu probe pueda soportar este protocolo[21]. Para ello vamos a seguir los siguientes pasos:

- Generación: Se pueden utilizar diferentes herramientas para generar datos netflow en máquinas GNU/Linux., en este caso se ha utilizado softflowd.

  En primer lugar, procederemos a instalar softflowd:
  ```
  $ sudo apt-get install softflowd
  ```

  Una vez instalado arrancaremos la herramienta y seleccionamos la interfaz y la máquina de la cual se generarán los datos.
  ```
  $ sudo softflowd -i eth1 -n 127.0.0.1:9995
  ```

  Tendremos las siguientes opciones a la hora de configurar softflowd:

  -i [idx:] interface Especifica la interfaz desde la que se escuchará.
  -r pcap_file Especifica el archivo de captura de paquetes para leer.
  -t timeout=time Especifica el tiempo de espera.
  -m max_flows Especifique el número máximo de flujos a realizar (8192 por defecto).
  -n host: port Enviar paquetes compatibles con Cisco NetFlow al host.
  -p pidfile Grabar pid en el archivo especificado.
  -c pidfile Ubicación de la toma de control.
  -v 1 | 5 | 9 Versión del paquete de exportación de NetFlow.
  -**L hoplimit** Set TTL/hoplimit para datagramas de exportación.
  -**T full | proto | ip** Establece el nivel de seguimiento del flujo..
  -6 Seguimiento de los flujos IPv6, independientemente de si se ha seleccionado.
  -d No se ejecuta demonio (ejecutar en primer plano).
  -D Modo de depuración: primer plano + verbosidad + seguimiento de los flujos v6.
  -s sampling_rate Especifica la frecuencia de muestreo periódico (denominador).
  -h Mostrar ayuda.

- Inyección: Después de capturar el tráfico que pasa por la interfaz podemos añadir diferentes tráficos para que circulen mezclados por esa misma interfaz, con este proceso conseguiremos lo que comentábamos antes, mezclar el tráfico normal generado por los clientes con el tráfico malicioso.
  Hay que ejecutar el siguiente comando para llevarlo a cabo.
  ```
  $ sudo tcpreplay -i eth1 malware.pcap
  ```

- Recolecta: Se recogerán los datos desde la máquina donde hayamos configurando el envío mediante softflowd. Para ello utilizaremos la herramienta nfcapd, que pertenece a la suit nfdump.

  Como primer paso debemos instalar la suit si no la tenemos instalada.
  ```
  $ sudo apt-get install nfdump
  ```

  Enviamos las trazas netflow al directorio elegido.
  ```
  $ sudo nfcapd -w -D -p 9995 -z -t 120 -l /home/probe/netflow/
  ```

  Las opciones que nos dará nfcapd son las siguientes:

  -w: Su función se basa en que los archivos que son rotados coincidan de forma exacta. Por ejemplo, si los archivos rotan cada 5min, tendremos archivos creados a las 12:05, 12:10, 12:15 y no en 12:03, 12:08, 12:13…
  -D: Se inicializa *nfcapd* en modo demonio.
  -p: El puerto por el que recolectaremos la información. Debe ser el mismo al que estamos enviando desde softflowd.
  -z: Comprime los paquetes de salida
  -t: Intervalo de tiempo en segundos en los que rotaremos los archivos, por defecto 300s.
  -l: Directorio donde se almacenarán las trazas.

- Volcado: Finalmente, una vez tenemos las trazas utilizaremos otra herramienta de la suit nfdump llamada también nfdump para volcar los datos a formato .csv y poder trabajar con ellos.

  ```
  $ nfdump -R /var/test_malware/ -o csv > /home/probe/test_malware.csv
  ```

Por tanto, podríamos resumir la arquitectura del proceso netflow en la siguiente imagen:

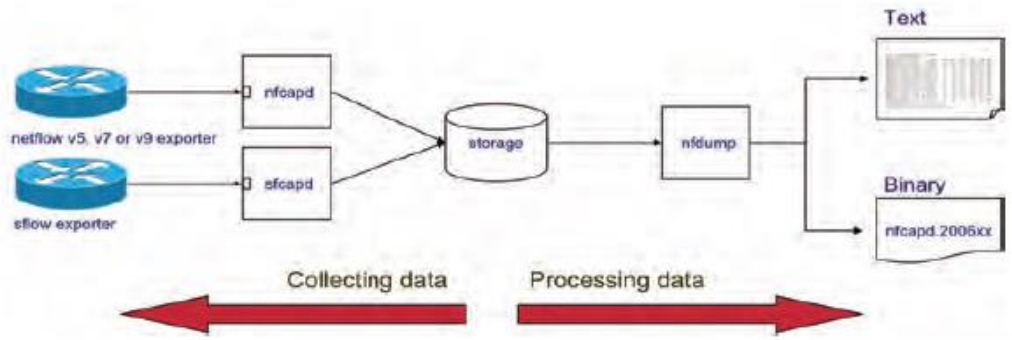

Figura 2. Proceso tratamiento Netflow[10]

5.2 ALGORITMOS SELECCIONADOS

Los algoritmos seleccionados para el experimento son los siguientes:

- Isolation Forest
- One class support vector machine.
- Local outlier factor.

Se han seleccionado estos algoritmos ya que son los que mejores métricas de resultados dan en diferentes estudios sobre la detección de anomalías para proyectos similares a este, además es fácil implementarlos con la librería que proporciona sklearn[22].

5.2.1 One class SVM

El algoritmo One class SVM utilizado para detectar anomalías, su característica principal es que computa las entradas del kernel dinámicamente para evitar limitaciones de memoria.

Puede ser visto como un SVM regular de dos clases, donde todos los datos de entrenamiento se encuentran en la primera clase mientras que los datos de test se toman como un miembro de la segunda clase[23].

Este algoritmo es del tipo novely detection, lo que quiere decir que los datos con los que realicemos el entrenamiento deben ser limpios, sin muestras de anomalías.

Su funcionamiento se basa en realizar diferentes agrupaciones de valores, los valores que queden fuera de esos grupos creados serán considerados como anomalías. Aquí vemos un ejemplo sencillo de su funcionamiento en dos dimensiones[24]:

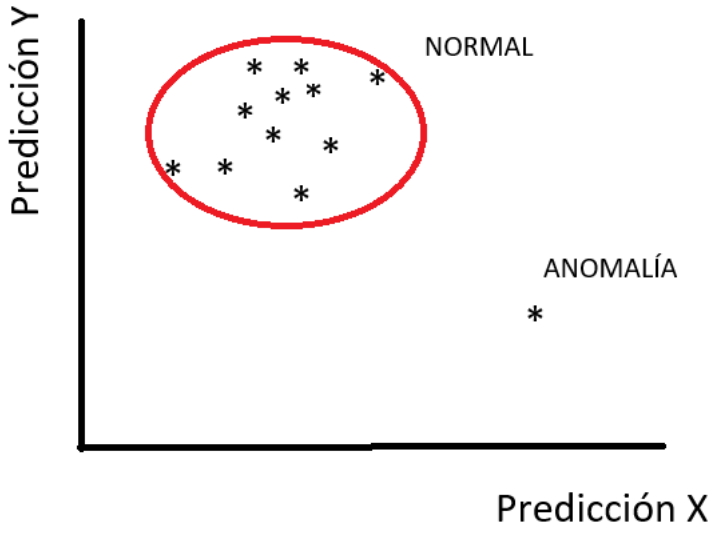

Figura 3. OCSVM

5.2.1.1 Argumentos utilizados

Los argumentos que tendremos a la hora de ejecutar el algoritmo serán los siguientes:

- **kernel:** Especificaremos el tipo de kernel que usará el algoritmo, en nuestro caso tendremos el kernel "rbf" (radial basis function).

- **nu:** Será la llamada contaminación, este valor va a ser muy importante en nuestro estudio ya que va a delimitar donde estará el rango de anomalías a detectar. El algoritmo no sabe distinguir el corte entre anomalía y normal, sino que nos dará un valor "score" con el que obtenemos un listado de la mayor anomalía a la menor. Como veremos más adelante en el estudio variaremos entre los valores de 0,1 y 0,01 de contaminación, que darán el 10% y 1% de anomalías del dataset total.

- **gamma:** Coeficiente dado al kernel "rbf". Le daremos un valor "auto" que será 1/número_caracteristicas.

- **tol:** Tolerancia que usaremos para el criterio de parada. Nuestro valor será 0,001.

- **shrinking:** Definirá si usaremos la heurística de reducción, en este caso lo marcaremos como TRUE.

- **cache_size:** Especificamos el tamaño en cache del kernel. Estará dado en Mb y nuestro valor elegido será 200Mb.

- **verbose:** Habilita la salida detallada, no nos será necesario y lo marcaremos como "FALSE".

- **max_iter:** Límite de iteraciones. Por defecto elegiremos -1 que será sin iteraciones.

- **random_state:** Será la semilla del generador de números aleatorios que se utiliza cuando se barajan los datos. En nuestro caso lo tendremos a "NONE" ya que no tendremos semilla.

5.2.2 Isolation Forest

El algoritmo Isolation Forest también es muy utilizado para detectar anomalías, en este caso este algoritmo construye diferentes árboles de decisión. Estos árboles se crearán utilizando submuestras de los datos de entrenamiento. Después los datos de test pasarán por esos árboles creado para aislar las anomalías.

Por tanto, como vemos funciona aislando las diferentes anomalías en árboles, cuanto antes se aísle un dato, mayor anomalía será considerada[25].

Ahora este algoritmo es del tipo outlier detection, lo que quiere decir que los datos con los que realicemos el entrenamiento deben contener anomalías para poder construir los árboles correctamente.

Vemos un ejemplo comparativo en dos dimensiones:

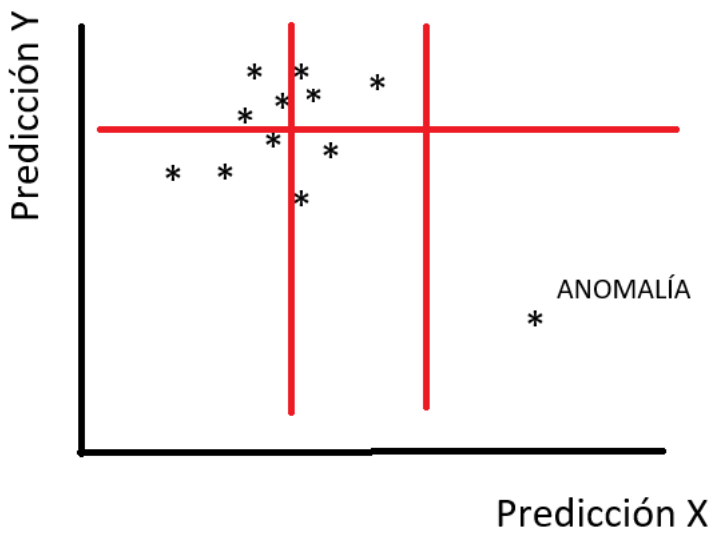

Figura 4. Isolation Forest

5.2.2.1 Argumentos utilizados

Para el algoritmo iForest los argumentos que tendremos a la hora de ejecutar el algoritmo serán los siguientes:

- **n_estimators:** Número de estimadores en el conjunto. En nuestro caso elegiremos un número de estimadores igual a 100.

- **max_samples:** Número de muestras a extraer de los datos de entrenamiento. Tendremos un total de 256 muestras para el estudio.

- **contaminación:** Es el valor de la contaminación comentado anteriormente para el algoritmo OCSVM, se va a variar entre 0,1 y 0,01 como hemos comentado antes.

- **max_features**: Será el número de características a dibujar de los datos de test para cada estimador.

- **bootstrap:** Si tenemos valor “TRUE” los árboles individuales se ajustan a subconjuntos aleatorios de los datos de entrenamiento muestreados con reemplazo. Si elegimos valor “FALSE” que será nuestro caso, tendremos el muestreo sin reemplazo.

- **n_jobs:** Número de trabajos en paralelo, normalmente se corresponderá al número de cores, en nuestro estudio lo dejaremos a 1.

- **verbose:** Controla la verbosidad del proceso de construcción del árbol.

- **random_state:** Al igual que para el anterior algoritmo, será la semilla del generador de números aleatorios que se utiliza cuando se barajan los datos. En nuestro caso lo tendremos a “NONE” ya que no tendremos semilla.

5.2.3 Local outlier factor

Al igual que los algoritmos OCSVM e iForest, este algoritmo es uno de los más utilizados a la hora de detectar anomalías.

Utiliza el concepto de densidad local, donde la distancia entre los k vecinos más próximos se utiliza para calcular la densidad. Esto nos dará regiones donde habrá una mayor densidad, y otras donde haya menos que se detectaran como anomalías[26].

Al igual que el algoritmo isolation forest es del tipo outlier detection, por lo que los datos de entrenamiento deben contener anomalía.

Vemos el ejemplo comparativo en dos dimensiones:

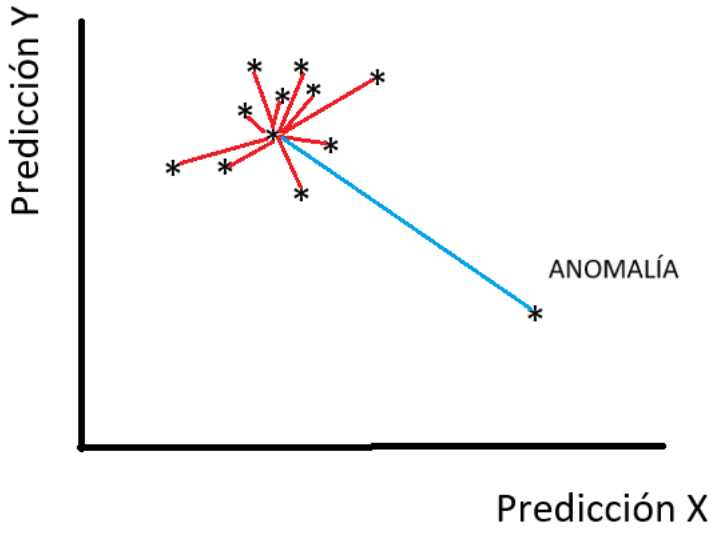

Figura 5. LOF

5.2.3.1 Argumentos utilizados

Para el algoritmo LOF vamos a tener los siguientes argumentos a la hora de ejecutar el algoritmo:

- **n_neighbors:** Número de vecinos que se usarán para calcular las distancias. Si es mayor que el número de todas las muestras proporcionadas, se utilizaran todas las muestras.

- **algorithm:** Algoritmo utilizado para calcular los vecinos más cercanos (BallTree, KDTree, brute-force). En este caso elegiremos el valor “auto” que decidirá el algoritmo más apropiado en base a los valores del entrenamiento.

- **metric:** Métrica utilizada para calcular las distancias entre muestras. Hay diferentes tipos y por defecto elegiremos “minkowski”.

- **contaminatión:** Misma explicación que para los otros algoritmos, será común a todos ellos y nos servirá para poder compararlos. Como hemos omentado variamos entre el 10% y el 1%.

- **n_jobs:** Como hemos comentando antes también será número de trabajos en paralelo, normalmente se corresponderá al número de cores, en nuestro estudio lo dejaremos a 1.

5.3 DISEÑO DE BIBLIOTECA. CLASES TRANSFORMADORES Y PIPELINES

Para poder realizar un correcto y ordenado tratamiento de los datos optimizando el código se han desarrollados diferentes clases transformadoras y pipelines.

Además, una de las claves para obtener buenos resultados detectando anomalías es la extracción de las diferentes características de nuestros datos. Dependiendo el tratamiento que realicemos a estas características vamos a obtener resultados muy dispares, por tanto, con las clases transformadoras y los pipelines podremos probar diferentes combinaciones rápidamente[27].

5.3.1 Clases transformadoras

Serán las encargadas de realizar diferentes funciones específicas de cada tipo. Serán del siguiente tipo[28]:

```
from sklearn.base import BaseEstimator, TransformerMixin

class SampleExtractor(BaseEstimator, TransformerMixin):

    def __init__(self, vars):
        self.vars = vars  #Pasamos variables.

    def transform(self, X, y=None):
        return do_something_to(X, self.vars)  #Donde se extraen las características.

    def fit(self, X, y=None):
        return self  #Generalmente no se hace nada.
```

Las clases transformadoras que se han definido son las siguientes:

- class Ascending_Order : Devolverá el dataset ordenado ascendentemente a partir de la columna que le pasamos como parámetro.

- class Delete_Columns : Devolverá el dataset sin las columnas a borrar que pasamos por parámetro.

- class Split_Columns : Devolverá el dataset con una columna añadida la cual es resultado de dividir dos columnas del dataset.

- class Normalize_Columns : Devuelve el dataset con las columnas normalizas que le indiquemos, en caso de no indicar ninguna columna devolverá el dataset normalizado completo.

- class OneHotEncode_Columns : Devolverá el dataset con las columnas categóricas que indiquemos codificadas.

- class Substract_Columns : Devolverá el dataset con una columna añadida la cual es resultado de restar dos columnas del dataset.

- class Add_Columns : Devolverá el dataset con una columna añadida la cual es resultado de sumar dos columnas del dataset.

- class Multiply_Columns : Devolverá el dataset con una columna añadida la cual es resultado de multiplicar dos columnas del dataset.

- class Filter_Values : Devuelve el dataset filtrado por lo valores máximos y mínimos de una columna seleccionada, habrá que indicar también si queremos esos intervalos abiertos o cerrados.

- class Change_Values : Devolverá el dataset con los valores elegidos de una columna seleccionada cambiados por otros valores dados.

- class Adjust_Times : Devolverá el dataset con las columnas de tiempo seleccionadas ajustadas a una frecuencia temporal seleccionada.

- class Decomposition : Devuelve el dataset con diferentes columnas añadidas fruto de reducir las dimensiones del dataset en el valor que especifiquemos. Estas técnicas de reducción de dimensiones son PCA, FastICA, TruncatedSVD y NMF.

5.3.2 Pipelines

La función de los pipelines será encadenar las diferentes clases transformadoras que elijamos logrando agilizar el proceso. Las clases transformadoras se realizarán en orden, y la entrada de cada una será la salida de la anterior debiendo tener bastante orden a la hora de elegir que operaciones realizar.

Tendrá la siguiente forma:

```
Pipeline = Pipeline ([
('NameTransformerClass1', TransformerClass1()),
('NameTransformerClass2', TransformerClass2()),
('NameTransformerClassN', TransformerClassN()),
])

Transformerdataset = pipeline.tranform(dataset)
```

6 APLICACIÓN PRÁCTICA Y RESULTADOS

6.1 PREPROCESAMIENTO DE LOS DATOS

Como hemos visto antes en el apartado Netflow, nfdump nos proporciona una plantilla específica para exportar los diferentes campos.

Por tanto, tendremos los siguientes campos en formato .csv que serán con los que podremos trabajar realizando el procesamiento de ellos y aplicando los algoritmos.

A continuación, vemos los campos (field_type) que utiliza la platilla que contiene nfdump, los cuales son los que obtendremos en nuestras trazas de tráfico[16].

CAMPO	DESCRIPCIÓN
FIRST_SEEN	Fecha inicio flujo.
LAST_SEEN	Fecha fin flujo.
DURATION	Duración flujo
IPV4_SRC_ADDR	Dirección IPv4 origen.
IPV4_DST_ADDR	Dirección IPv4 destino.
L4_SRC_PORT	Puerto origen capa 4.
L4_DST_PORT	Puerto destino capa 4.
PROTOCOL	Protocolo.
TCP_FLAGS	Flags TCP
SRC_TOS	Tipo de configuración del byte de servicio origen.
IN_PKTS	Número paquetes entrada.
IN_BYTES	Número bytes entrada.
OUT_PKTS	Número paquetes salida.
OUT_BYTES	Número paquetes salida.
INPUT_SNMP	Interface entrada.
OUTPUT_SNMP	Interface salida.
SRC_AS	Número de sistema autónomo BGP origen.
DST_AS	Número de sistema autónomo BGP destino.
SRC_MASK	Máscara de red origen.
DST_MASK	Máscara de red destino
DST_TOS	Tipo de configuración del byte de servicio destino.
DIRECTION	Dirección flujo.
IPV4_NEXT_HOP	Dirección IPv4 del router del próximo salto.
BGP_IPV4_NEXT_HOP	Dirección IPv4 del router del próximo salto en el dominio BGP.
SRC_VLAN	VLAN origen.
DST_VLAN	VLAN destino.
IN_SRC_MAC	Dirección MAC origen de entrada.
OUT_DST_MAC	Dirección MAC destino de salida.
IN_DST_MAC	Dirección MAC destino de entrada.
OUT_SRC_MAC	Dirección MAC origen de salida.
MPLS_LABEL_1	Etiqueta MPLS posición 1.

MPLS_LABEL_2	Etiqueta MPLS posición 2.
MPLS_LABEL_3	Etiqueta MPLS posición 3.
MPLS_LABEL_4	Etiqueta MPLS posición 4.
MPLS_LABEL_5	Etiqueta MPLS posición 5.
MPLS_LABEL_6	Etiqueta MPLS posición 6.
MPLS_LABEL_7	Etiqueta MPLS posición 7.
MPLS_LABEL_8	Etiqueta MPLS posición 8.
MPLS_LABEL_9	Etiqueta MPLS posición 9.
MPLS_LABEL_10	Etiqueta MPLS posición 10.
CLIENT_LATENCY	Latencia del cliente.
SERVER_LATENCY	Latencia del servidor.
APP_LATENCY	Latencia de la aplicación.
REPLICATOR_FACTOR	Multidifusión factor de replicación
ENGINE_TYPE/ENGINE_ID	Tipo máquina/ID máquina
EXPORTING_SYSTEM_ID	ID del sistema exportador
RECEIVED_TIMESTAMP	Timestamp recibido.

Preprocesamiento mediante clases transformadoras y pipelines:

Para un primer preprocesamiento vamos a utilizar las clases transformadoras creadas que describimos antes, las cuales se unirán todas en un pipeline. Los procesamientos realizados son los siguientes:

- Eliminación de características que nos proporciona Netflow que no aportan ningún valor ya que tienen valores a 0 al no ser soportadas por nuestra máquina. Esto se puede deber a una configuración especifica de la máquina donde capturemos el tráfico, o que realmente al no estar capturando desde un dispositivo Cisco perdamos algunas características del tráfico.
  Nos quedan las siguientes 11 características:

CAMPO	DESCRIPCIÓN
FIRST_SEEN	Fecha inicio flujo.
LAST_SEEN	Fecha fin flujo.
DURATION	Duración flujo
IPV4_SRC_ADDR	Dirección IPv4 origen.
IPV4_DST_ADDR	Dirección IPv4 destino.
L4_SRC_PORT	Puerto origen capa 4.
L4_DST_PORT	Puerto destino capa 4.
PROTOCOL	Protocolo.
TCP_FLAGS	Flags TCP
IN_PKTS	Número paquetes entrada.
IN_BYTES	Número bytes entrada.

- Ajustar las características de tiempo a la unidad segundos. Con esto conseguimos una mayor precisión en el tiempo a la hora de organizar los flujos.

- Cambiamos el valor de los flujos de duración con valor 0 aun valor bajo próximo a 0, con ello conseguimos que el algoritmo asimile mejor esos flujos.

- Añadimos nuevas características de interés a partir de las que ya tenemos:

- IN_PKTS_SEG: Número de paquete de entrada por segundo. Viene dado por la división del número de paquetes de entrada entre la duración del flujo.

- IN_BYTES_SEG: Número de bytes de entrada por segundo. Viene dado por la división del número de bytes de entrada entre la duración del flujo.

- IN_BYTES_PKTS: Número de bytes de entrada por cada paquete. Se calcula como la división del número de bytes por segundo entre el número de paquetes por flujo.

Por tanto, tenemos las siguientes características después de añadir las columnas dichas:

CAMPO	DESCRIPCIÓN
FIRST_SEEN	Fecha inicio flujo.
LAST_SEEN	Fecha fin flujo.
DURATION	Duración flujo
IPV4_SRC_ADDR	Dirección IPv4 origen.
IPV4_DST_ADDR	Dirección IPv4 destino.
L4_SRC_PORT	Puerto origen capa 4.
L4_DST_PORT	Puerto destino capa 4.
PROTOCOL	Protocolo.
TCP_FLAGS	Flags TCP
IN_PKTS	Número paquetes entrada.
IN_BYTES	Número bytes entrada.
IN_PKTS_SEG	Número paquetes entrada por segundo.
IN_BYTES_SEG	Número bytes entrada por segundo.
IN_BYTES_PKTS	Número bytes por paquete.

- Codificamos las características categóricas PROTOCOL y TCP_FLAGS a valores numéricos. Si no realizamos este procesamiento el algoritmo no podrá asimilar esas características produciéndose un error.

- Eliminamos las siguientes características:
  LAST_SEEN: Con la fecha de inicio del flujo(FIRST_SEEN) y la duración del flujo (DURATION) podemos definir la fecha final del flujo, por tanto, podemos excluir esta característica.

  IPV4_DST_ADDR: Nos quedaremos sólo con la IP origen (IPV4_SRC_ADDR) ya que la utilizaremos como índice para poder tener una referencia del flujo como veremos en los pasos posteriores. La IP destino será descartada.

  L4_SRC_POR: El puerto origen no nos será de utilidad a la hora de detectar los flujos anómalos por lo que lo descartaremos.

- Por último, normalizamos el dataset para un mayor análisis de los datos. Con lo que tenemos para este primer preprocesamiento con clases transformadoras y pipelines.

CAMPO	DESCRIPCIÓN
FIRST_SEEN	Fecha inicio flujo.
DURATION	Duración flujo
IPV4_SRC_ADDR	Dirección IPv4 origen.
L4_DST_PORT	Puerto destino capa 4.
PROTOCOL	Protocolo codificado a valores.
TCP_FLAGS	Flags TCP codificados a valores.
IN_PKTS	Número paquetes entrada.
IN_BYTES	Número bytes entrada.
IN_PKTS_SEG	Número paquetes entrada por segundo.
IN_BYTES_SEG	Número bytes entrada por segundo.
IN_BYTES_PKTS	Número bytes por paquete.

Preprocesamiento mediante agrupaciones:

En el siguiente paso vamos a realizar agrupaciones para optimizar el dataset. Se han realizado las siguientes operaciones.

- Agregar todos los flujos con mismas características FIRST_SEEN (misma hora, minuto y segundo), IPV4_SRC_ADDR, L4_DST_PORT, PROTOCOL y TCP_FLAGS. En estas agrupaciones se probaron diferentes opciones como ajustar la fecha de inicio de los flujos a una frecuencia mayor del segundo para las agrupaciones.

- Esos flujos agregados sus características DURATION, IN_PKTS, IN_BYTES, IN_PKTS_SEG, IN_BYTES_SEG y IN_BYTES_PKTS se sumarán. Esto lo podemos hacer ya que al ser características numéricas es coherente sumar la duración y el número de paquetes y bytes de los flujos agregados. Se han probado otras operaciones con estas características como realizar la media de los flujos o la desviación típica, pero se obtienen peores resultados en las pruebas.

- Agregamos una nueva característica llamada N_CON que será el número de flujos agregados en cada nuevo flujo.

- Hacemos índice a las características FIRST_SEEN y IPV4_SRC_ADDR las cuales no se pasarán al algoritmo, sino que servirán de referencia para diferenciar diferentes flujos.

Con lo qué, tendremos las siguientes características finales procesadas que le pasaremos a los algoritmos de detección de anomalías.

CAMPO	DESCRIPCIÓN
CARACTERÍSTICAS INDICE	
FIRST_SEEN	Fecha inicio flujo.
IPV4_SRC_ADDR	Dirección IPv4 origen.
CARACTERÍSTICAS DATOS	

DURATION	Duración flujo
L4_DST_PORT	Puerto destino capa 4.
PROTOCOL	Protocolo codificado a valores.
TCP_FLAGS	Flags TCP codificados a valores.
IN_PKTS	Número paquetes entrada.
IN_BYTES	Número bytes entrada.
IN_PKTS_SEG	Número paquetes entrada por segundo.
IN_BYTES_SEG	Número bytes entrada por segundo.
IN_BYTES_PKTS	Número bytes por paquete.
N_CON	Número de flujos agregados.

6.2 APLICACIÓN ALGORITMOS Y RESULTADOS

Ahora vamos a la parte más práctica del estudio, se van a aplicar los diferentes algoritmos elegidos a nuestros datos procesados, como hemos visto antes el tráfico utilizado será tráfico de red normal (servicios de navegación, descargas y video) que generan diferentes usuarios mezclando con diferentes familias de tráficos maliciosos. Estas familias de tráfico malicioso utilizadas se explicarán con más detalle ahora para cada prueba.

Como hemos comentado antes para las pruebas el tráfico malicioso se ha dividido en varias familias con la intención de probar diferentes casos que cubriesen el mayor espectro posible de tipos de malware.

6.2.1 Métricas para el análisis de los algoritmos.

Para ver la fiabilidad de nuestro algoritmo vamos a calcular diferentes métricas dados a partir de los siguientes valores:

- Verdaderos positivos: Número de anomalías detectadas como positivas
- Verdadero negativo: Número de no anomalías no detectadas como anomalías.
- Falso positivo: Número de anomalías detectadas falsas.
- Falso negativo: Número de anomalías no detectadas como anomalías.

Estas métricas se han elegido ya que son las clásicas para poder estudiar los resultados de tales experimentos estadísticos, en la mayoría de estudias relacionados con el análisis de algoritmos son empleadas para poder realizar estudios comparativos.

Por tanto a partir de los valores propuestos sacaremos las siguientes métricas que nos van a delimitar la calidad del algoritmo[29]:

TPR (Tasa verdaderos positivos) = $\frac{\mathbf{VP}}{\mathbf{VP + FN}}$

La tasa de verdaderos positivos nos va a representar el porcentaje de anomalías verdaderas que vamos a detectar respecto de todas las anomalías que tenemos. Su valor deseado máximo es 1 y significará que detectamos todas las anomalías en el rango de contaminación exacto.

TNR (Tasa verdaderos negativos) = $\frac{VN}{VN + FP}$

La tasa de verdaderos negativos representa el porcentaje de tráfico normal no detectado respecto a los errores en la detección, por tanto, cuando su valor a desear máximo sea 1 todo el tráfico normal será detectado como normal.

FPR (Tasa falsos positivos) = $\frac{FP}{FP + VN}$

La tasa de falsos positivos no es más que el porcentaje de tráfico normal detectado como anómalo respecto al tráfico normal no detectado, en este caso nos interesa que su valor sea lo más cercano a 0 así no tendremos tráfico normal detectado como anomalía.

FNR (Tasa falsos negativos) = $\frac{FN}{FN + VP}$

La tasa de falsos negativos nos va a dar el porcentaje de anomalías verdaderas no detectadas como anomalías respecto a las anomalías verdaderas. Nos interesa que sea un valor lo más cercano a 0 posible ya que si es así detectaremos todas las anomalías en el rango de contaminación exacto.

Con estos valores podremos calcular el área bajo la curva ROC, que nos dará la calidad de detección de nuestro algoritmo. Si se encuentra en valores en torno al 0,5 no tendremos calidad de detección ya que tiende a la aleatoriedad el algoritmo, por el contrario, si obtenemos 1 estaremos ante la detección perfecta. Se calculará como[30]:

AUC (Área bajo la curva) = $\frac{TPR+TNR}{2}$

6.2.2 1º Familia de tráfico malicioso:

Este tipo de malware aprovecha vulnerabilidades de la familia RIG EK. Esta familia se trata de un exploit kit (descubrir y explotar vulnerabilidades en equipos afectados) que aprovecha vulnerabilidades para atacar a tecnologías Java, Flash, Internet Explorer y Silverligth[31].

Su funcionamiento es sencillo y se basa en enviar a la víctima un enlace que contiene un JavaScript, al ejecutarlo se comprobará los plugins vulnerables. Hay que saber que el exploit kit no dispone en el de un payload malicioso con el que afectar el equipo, sino que se deberá añadir después. Por tanto, la peligrosidad máxima de este tipo de malware es que son la puerta de entrada de diferentes amenazas a estos equipos como ransonware, spambots, spyware o troyanos[32].

En España este tipo de malware es una de las mayores amenazas junto a otros malware como HackerDefender, Conficker, Nivdort o Cryptowall[33].

6.2.2.1 1º Prueba, envío troyano Bunuti mediante RIG EK

Características tráfico usado

En esta primera prueba se va a enviar a partir de RIG EK el troyano llamado Bunuti a través de una vulnerabilidad en flash, el cual convierte en un proxy con muchos puertos abiertos al equipo atacado. Estos ordenadores afectados serán la puerta para acceder a las VPN

revendiendo los ciberdelincuentes estos servicios, notando los equipos afectados un rendimiento bajo en su conexión[34].

Propiedades del malware[35]:

- Exploit Flash:

SHA256 hash: 892b3990a09bb3391c5a1a591d9908a0e77db7385addc2c38cfcb32db265a970
Tamaño archivo: 16,299 bytes
Descripción archivo: Rig EK flash exploit on 2017-06-21 (1st and 2nd runs)

- Troyano Bunuti:

SHA256 hash: e7049140f7f07a48406a7de2e3a08fc8759af9f94de533210e99f69c34bc27aa
Tamaño archivo: 177,498 bytes
Dirección archivo: C:\Users\[username]\AppData\Local\Temp*[random characters]*.exe
Descripción archivo: Rig EK payload of Bunitu on 2017-06-21

SHA256 hash: 951d15d83264c1ffd387fd871d3f0776e0016b00bc353e17c6911562da6e2169
Tamaño archivo: 13,312 bytes
Dirección archivo: C:\Users\[username]\AppData\Local\drudppf.dll
Descripción archivo: Post-infection artifact

IPs anómalas a detectar:

78.47.1.222	Puerto 80	GET /yo/? (Puerta al RIG EK)
188.225.79.216	Puerto 80	RIG EK
5.79.85.218	Puerto 443	Tráfico Bunuti después de la infección
62.75.222.235	Puerto 443	Tráfico Bunuti después de la infección
66.199.231.77	Puerto 443	Tráfico Bunuti después de la infección
209.85.144.100	Puerto 443	Tráfico Bunuti después de la infección

Aplicación de algoritmos y resultados

El dataset utilizado tiene las siguientes características:

Flujos totales	Flujos después proc	Anomalías después proc	% Anomalías después proc
23261	11289	8	0,070865%

Vamos a realizar la comparación entre los tres algoritmos seleccionados para una contaminación del 10% y del 1%

Contaminación 10%

Este valor de contaminación es el que utilizan normalmente los algoritmos por defecto siendo bastante alto a priori si se tiene un número bajo de anomalías.

- OneClassSVM

Al ejecutar el algoritmo hemos obtenido los siguientes valores de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos. Cabe destacar que tendremos unos valores altos de FP ya que sólo tenemos 8 flujos anómalos de un total de 11288, lo que conllevara que en contaminaciones altas se detecten flujos normales como anómalos al no existir tantos flujos anómalos que cubran esa contaminación. Esto ocurrirá igual en los demás experimentos. También obtenemos métricas que nos darán la calidad de detección de los datos anómalos. Podemos apreciar unas métricas altas en la detección.

Anomalías para el 10%	VP	VN	FP	FN
1128	8	10161	1120	0

Métricas	Resultados
TPR (Tasa verdaderos positivos)	1
TNR (Tasa verdaderos negativos)	0,900718021
FPR (Tasa falsos positivos)	0,099281979
FNR (Tasa falsos negativos)	0
AUC	0,950359011

Podemos apreciar unas métricas altas en la detección dando en principio un buen funcionamiento el algoritmo.

- IForest

Obtenemos los siguientes valores de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos junto con las métricas.

Anomalías para el 10%	VP	VN	FP	FN
1128	3	10156	1125	5

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0,375
TNR (Tasa verdaderos negativos)	0,900274798
FPR (Tasa falsos positivos)	0,099725202
FNR (Tasa falsos negativos)	0,625
AUC	0,637637399

Van a empeorar bastante los resultados con respecto a los otros dos algoritmos, ya que nos vamos a dejar sin detectar 5 anomalías de las 8 en total.

- LOF

Vemos los valores de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos además de las métricas para el último algoritmo a comparar.

Anomalías para el 10%	VP	VN	FP	FN
1128	7	10160	1121	1

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0,875
TNR (Tasa verdaderos negativos)	0,900629377
FPR (Tasa falsos positivos)	0,099370623
FNR (Tasa falsos negativos)	0,125
AUC	0,887814688

Vemos una mejora substancial respecto al algoritmo iForest, pero inferior al OCSVM.

Contaminación 1%

Ahora con este valor de contaminación vamos a detectar muy pocas anomalías respecto al dataset total, con un valor tan pequeño podremos ver si los algoritmos aíslan realmente las anomalías arriba en la tabla o no siendo más fiables.

- OneClassSVM

Igual que para el caso del 10% de contaminación al ejecutar el algoritmo hemos obtenido los siguientes valores de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos junto con las métricas.

Anomalías para el 1%	VP	VN	FP	FN
112	8	11177	104	0

Métricas	Resultados
TPR (Tasa verdaderos positivos)	1
TNR (Tasa verdaderos negativos)	0,990780959
FPR (Tasa falsos positivos)	0,009219041
FNR (Tasa falsos negativos)	0
AUC	0,99539048

Vemos en este caso que el algoritmo mejora sus tasas si estrechamos el rango de anomalías, lo que viene a significar que el algoritmo localiza muy bien las anomalías en los primeros puestos.

- iForest

Vemos los valores obtenidos y las métricas.

Anomalías para el 1%	VP	VN	FP	FN
112	1	11170	111	7

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0,125
TNR (Tasa verdaderos negativos)	0,990160447
FPR (Tasa falsos positivos)	0,009839553
FNR (Tasa falsos negativos)	0,875
AUC	0,557580223

Obtenemos las métricas las cuales empeoran si reducimos las anomalías, dándonos unos valores para nada óptimos ya que están rozando la aleatoriedad.

- LOF

Obtenemos los siguientes valores y métricas para el último caso.

Anomalías para el 1%	VP	VN	FP	FN
112	0	11169	112	8

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0
TNR (Tasa verdaderos negativos)	0,990071802
FPR (Tasa falsos positivos)	0,009928198
FNR (Tasa falsos negativos)	1
AUC	0,495035901

Por último, para el algoritmo LOF obtenemos unos resultados no validos ya que tenemos unos resultados iguales si el sistema fuera aleatorio

Comparación entre algoritmos

Después de obtener los resultados para este primer malware vamos a ver una comparación conjunta.

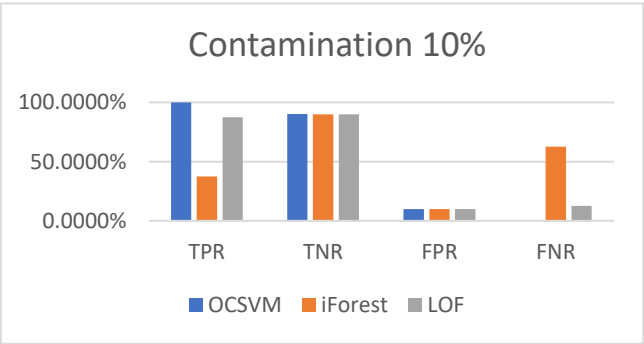

Figura 6. RIGEK1 10%

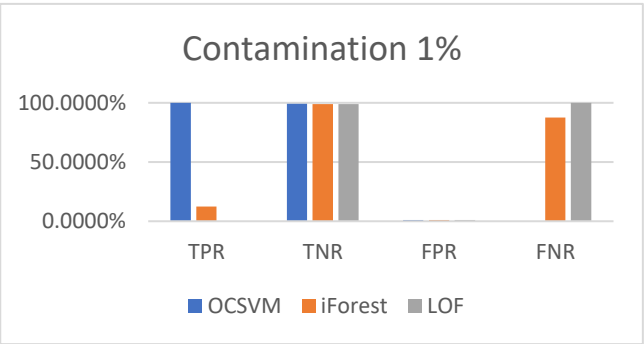

Figura 7. RIGEK1 1%

En estas gráficas se ve muy bien que para obtener valores buenos en la detección debemos tener unos porcentajes altos de TPR y TNR, mientras que de FPR y FNR deben ser bajos.

El algoritmo que mejores resultados da para esta primera prueba es el OCSVM ya que mantiene constantes sus valores, aunque se estreche el cerco de anomalías.

Los algoritmos iForest y LOF cuando se detectan tasas altas de anomalías tienen resultados medianamente óptimos, pero al reducir ese cerco de anomalías mediante la contaminación se vuelven inservibles lo que les hace malos detectores.

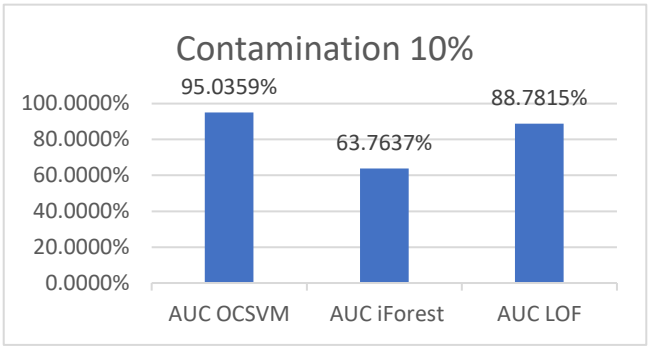

Figura 8. RIGEK1 10%

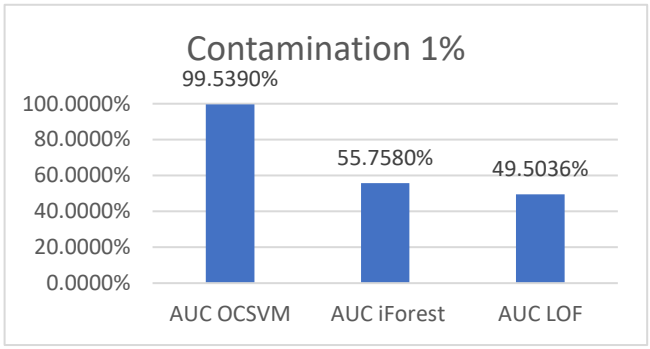

Figura 9. RIGEK1 1%

Esto lo podemos apreciar analizando los valores del AUC, donde vemos que si el algoritmo se mantiene constante en tasas altas será bueno.

6.2.2.2 2º Prueba, envío dreambot en campaña HookAds mediante RIG EK

Características tráfico usado

Ahora en este segundo caso el equipo será infectado con un dreambot a través del RIG EK en vez de con un troyano. Es enviado a través de la campaña HookAds (publicidad maliciosa) y modifica entradas del registro guardando archivos maliciosos en carpetas ocultas del sistema activando procesos ocultos. Su objetivo es recabar información personal o financiera del equipo infectado sin ser detectado el mayor tiempo posible[36].

Propiedades del malware[37]:

- Exploit Flash:

SHA256 hash: f27d2c74b94d9f08fee0e166472b6275613e04e955ea631d06e63ac11e9badd3
Tamaño archivo: 16,296 bytes
Descripción archivo: Rig EK flash exploit on 2017-06-19

- Dreambot

SHA256 hash: 320da7b8f30c94a14159d3de36a1e21594424ba37c6885bedc1a26ab52ab38a5
Tamaño archivo: 350,208 bytes

Dirección archivo: C:\Users\[username]\AppData\Local\Temp\2nxu57tc.exe
Descripción archivo: HookAds campaign Rig EK payload on 2017-06-19 – Dreambot

IPs anómalas a detectar:

80.77.82.41	Puerto 80	GET /banners/uaps – Redireccionar HookAds
92.53.119.254	Puerto 80	RIG EK
144.168.45.110	Puerto 80	Tráfico Dreambot después de la infección

Aplicación de algoritmos y resultados

El dataset utilizado en este caso será diferente y tiene las siguientes características:

Flujos totales	Flujos después proc	Anomalías después proc	% Anomalías después proc
90002	37939	5	0,013179%

Al igual que antes vamos a realizar la comparación entre los tres algoritmos seleccionados para una contaminación del 10% y del 1%

Contaminación 10%

Este valor de contaminación es el que utilizan normalmente los algoritmos por defecto siendo bastante alto a priori si se tiene un número bajo de anomalías.

- OneClassSVM

Por el mismo procedimiento que antes ejecutamos el algoritmo y hemos obtenido los siguientes valores de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos.

Anomalías para el 10%	VP	VN	FP	FN
3793	5	34146	3788	0

Métricas	Resultados
TPR (Tasa verdaderos positivos)	1
TNR (Tasa verdaderos negativos)	0,900142353
FPR (Tasa falsos positivos)	0,099857647
FNR (Tasa falsos negativos)	0
AUC	0,950071176

Obtenemos un valor de AUC muy semejante al mismo caso en la primera prueba siendo esto bastante importante.

- iForest

Hemos obtenido los siguientes valores y métricas.

Anomalías para el 10%	VP	VN	FP	FN
3793	5	34146	3788	0

Métricas	Resultados
TPR (Tasa verdaderos positivos)	1
TNR (Tasa verdaderos negativos)	0,900142353
FPR (Tasa falsos positivos)	0,099857647
FNR (Tasa falsos negativos)	0
AUC	0,950071176

Ahora al contrario que para la primera prueba el algoritmo iForest da unos resultados bastante buenos, debido al rango de anomalías elegido, cuando bajemos este valor al 1% podremos ver que no mantiene estas tasas.

- LOF

Los valores y métricas en el caso de este algoritmo son las siguientes.

Anomalías para el 10%	VP	VN	FP	FN
3793	5	34146	3788	0

Métricas	Resultados
TPR (Tasa verdaderos positivos)	1
TNR (Tasa verdaderos negativos)	0,900142353
FPR (Tasa falsos positivos)	0,099857647
FNR (Tasa falsos negativos)	0
AUC	0,950071176

Va a ocurrir en este caso lo mismo que para iForest, teniendo tasas altas cuanto tenemos una contaminación alta.

Contaminación 1%

Realizamos el mismo proceso reduciendo el valor de la contaminación al 1%, ahora con un valor tan pequeño podremos ver realmente la capacidad del algoritmo que podemos prever en el 10%.

- OneClassSVM

Tenemos los valores de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos junto con las métricas.

Anomalías para el 1%	VP	VN	FP	FN
379	5	37560	374	0

Métricas	Resultados
TPR (Tasa verdaderos positivos)	1
TNR (Tasa verdaderos negativos)	0,990140771
FPR (Tasa falsos positivos)	0,009859229
FNR (Tasa falsos negativos)	0
AUC	0,995070385

Al igual que para la primera prueba el algoritmo OCSVM funciona realmente bien mejorando los resultados.

- iForest

Tenemos los siguientes valores y métricas.

Anomalías para el 1%	VP	VN	FP	FN
379	3	37558	376	2

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0,6
TNR (Tasa verdaderos negativos)	0,990088048
FPR (Tasa falsos positivos)	0,009911952
FNR (Tasa falsos negativos)	0,4
AUC	0,795044024

Aunque se mantiene en unos valores aceptables de detección el algoritmo no se mantiene en niveles altos contaminaciones bajas como el caso del OCSVM.

- LOF

Por último, tenemos para este caso los valores y las métricas.

Anomalías para el 1%	VP	VN	FP	FN
379	0	37555	379	5

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0
TNR (Tasa verdaderos negativos)	0,990008963
FPR (Tasa falsos positivos)	0,009991037
FNR (Tasa falsos negativos)	1
AUC	0,495004481

Las tasas de detección se asemejarán a la aleatoriedad al bajar la contaminación.

Comparación entre algoritmos

Analizamos ahora los resultados en conjunto.

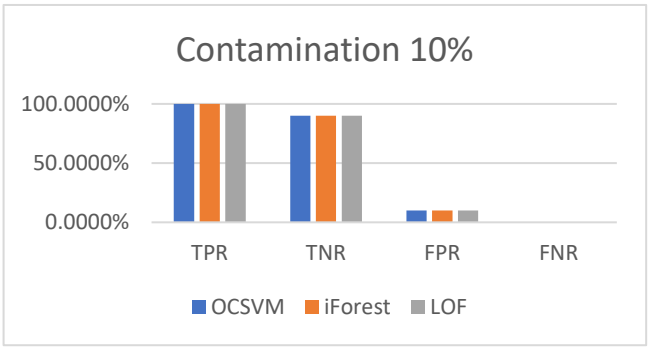

Figura 10. RIGEK2 10%

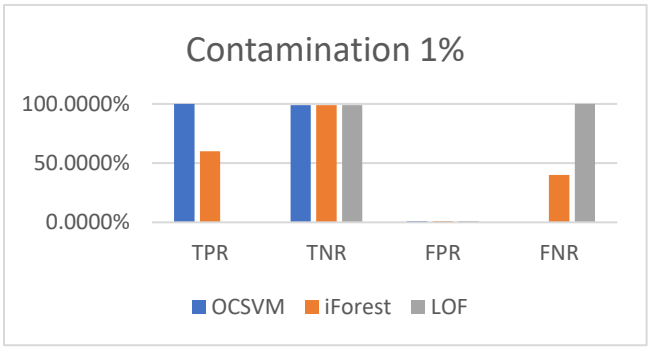

Figura 11. RIGEK2 1%

El algoritmo que mejores resultados da para esta segunda prueba al igual que para la primera es el OCSVM ya que tiene unos valores altos de AUC para contaminaciones bajas.

El algoritmo iForest en este caso tendrá resultados aceptables, mientras que él algoritmo LOF tiende a la aleatoriedad al reducir la contaminación.

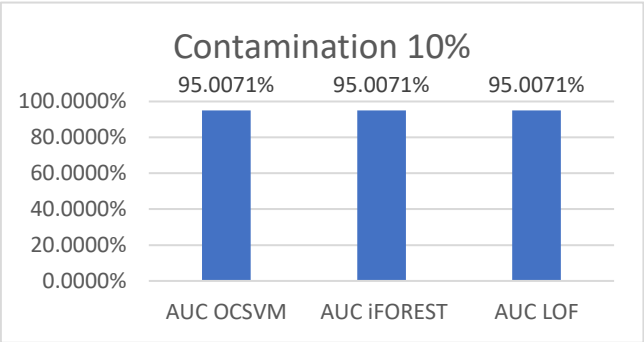

Figura 12. RIGEK2 10%

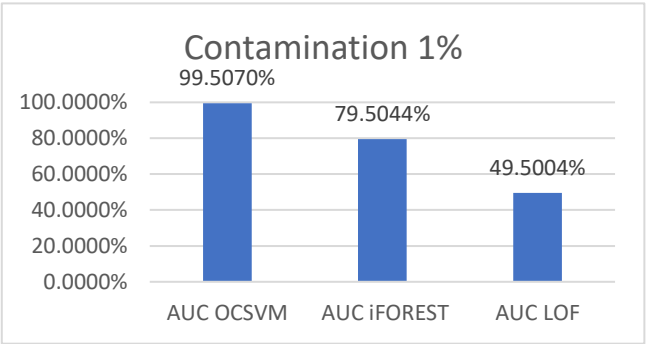

Figura 13. RIGEK2 1%

Lo explicado lo podemos apreciar con las gráficas del AUC, en las que claramente vemos que el algoritmo con mejor calidad es el OCSVM ya que tiene tasas altas para valores bajos de contaminación.

A partir de los datos podemos ir apreciando que el algoritmo OCSVM funciona bastante viene para este tipo de familias Exploit Kit, lo que corroboramos con la tercera prueba de esta familia.

6.2.2.3 3º Prueba, envío troyano Chthonic campaña RoughTed mediante RIG EK

En el último caso para RIG EK se estudiará otro troyano, llamado Chthonic. Es enviado a través de la campaña RoughTed (enlaces a webs maliciosas con troyanos, ransomware…). Funciona similar al anterior modificando registros y creando ejecutables. Su objetivo es recabar

información personal y financiera del equipo infectado para enviarse dinero, creando también una interfaz similar a los bancos consiguiendo credenciales además de obtener el código enviado para la doble autentificación[38].

Propiedades del malware[39]:

- Exploit Flash:

SHA256 hash: 9fc5fb99f72be24ec7d1e2004f1c1f2083885059e0e072314cb712934415bc24
Tamaño archivo: 16,468 bytes
Descripción archivo: Rig EK Flash exploit seen on 2017-06-06

- Chthonic:

SHA256 hash: 0434a5b69bea3a10443c0740bca4f36772cf67130c6b7da5b1b16494b3e12377
Dirección archivo: C:\Users\[username]\AppData\Local\Temp*[random characters]*.tmp
Dirección archivo: C:\Users\[username]\AppData\Roaming\Microsoft Visual Studio\MicrosoftVisualStudioM.exe
Dirección archivo: C:\Users\[username]\AppData\Roaming\Windows Media Player\WindowsMediaPlayerU.exe
Tamaño archivo: 212,992 bytes
Descripción archivo: Chthonic banking Trojan

- Cambios persistentes en el registro de windows:

Nombre de llave: HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run

Nombre de valor: MicrosoftVisualStudioM
Tipo de valor: REG_SZ
Datos de valor: C:\Users\[username]\AppData\Roaming\Microsoft Visual Studio\MicrosoftVisualStudioM.exe

Nombre de valor: WindowsMediaPlayerU
Tipo de valor: REG_SZ
Datos de valor: C:\Users\[username]\AppData\Roaming\Windows Media Player\WindowsMediaPlayerU.exe

IPs anómalas a detectar:

144.76.174.172	Puerto 80	GET /xfile (Puerta a RIG EK mediante RoughTed)
194.87.95.16	Puerto 80	RIG EK
47.91.78.49	Puerto 80	Tráfico Chthonic después de la infección
23.94.5.133	Puerto 53	Consulta DNS a web maliciosa
45.32.28.232	Puerto 53	Consulta DNS a web maliciosa
45.56.117.118	Puerto 53	Consulta DNS a web maliciosa
87.98.175.85	Puerto 53	Consulta DNS a web maliciosa
93.170.96.235	Puerto 53	Consulta DNS a web maliciosa
108.61.164.218	Puerto 53	Consulta DNS a web maliciosa
188.165.200.156	Puerto 53	Consulta DNS a web maliciosa
45.63.99.180	Puerto 53	Intento conexión TCP desde servidor
141.138.157.53	Puerto 53	Intento conexión TCP desde servidor

Aplicación de algoritmos y resultados

El dataset utilizado en esta última familia tendrá las siguientes características:

Flujos totales	Flujos después proc	Anomalías después proc	% Anomalías después proc
60345	37128	34	0,091575%

Realizamos la comparación entre los tres algoritmos seleccionados para una contaminación del 10% y del 1%.

Contaminación 10%

Analizamos los diferentes algoritmos.

- OneClassSVM

Seguimos el mismo criterio que en las anteriores pruebas y ejecutamos el algoritmo obteniendo los siguientes valores de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos.

Anomalías para el 1%	VP	VN	FP	FN
3712	34	33416	3678	0

Métricas	Resultados
TPR (Tasa verdaderos positivos)	1
TNR (Tasa verdaderos negativos)	0,900846498
FPR (Tasa falsos positivos)	0,099153502
FNR (Tasa falsos negativos)	0
AUC	0,950423249

Se cumple un valor de AUC similar a los casos anteriores, ya que seguimos detectando todas las anomalías.

- iForest

Tenemos siguientes valores y métricas.

Anomalías para el 1%	VP	VN	FP	FN
3712	21	33403	3691	13

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0,617647059
TNR (Tasa verdaderos negativos)	0,900496037
FPR (Tasa falsos positivos)	0,099503963
FNR (Tasa falsos negativos)	0,382352941
AUC	0,759071548

Obtenemos unos valores aceptables para una contaminación alta, habrá que ver ahora cómo se comporta en contaminaciones bajas.

- LOF

Los valores y las métricas serán los expuestos en las siguientes tablas.

Anomalías para el 1%	VP	VN	FP	FN
3712	3	33385	3709	31

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0,088235294
TNR (Tasa verdaderos negativos)	0,900010783
FPR (Tasa falsos positivos)	0,099989217
FNR (Tasa falsos negativos)	0,911764706
AUC	0,494123039

Para el algoritmo LOF obtendremos unos valores similares a la aleatoriedad con valores altos de anomalías, lo que nos llevara a un mismo caso si reducimos la contaminación.

Contaminación 1%

Reducimos el valor de la contaminación al 1% y realizamos el mismo proceso.

- OneClassSVM

La tabla de valores y métricas para el OCSVM son las suiguientes.

Anomalías para el 1%	VP	VN	FP	FN
371	34	36757	337	0

Métricas	Resultados
TPR (Tasa verdaderos positivos)	1
TNR (Tasa verdaderos negativos)	0,990914973
FPR (Tasa falsos positivos)	0,009085027
FNR (Tasa falsos negativos)	0
AUC	0,995457486

Mejoran los valores cuando bajamos la contaminación igual que en las pruebas anteriores.

- IForest

Vemos los valores obtenidos junto con las métricas.

Anomalías para el 1%	VP	VN	FP	FN
371	4	36727	367	30

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0,117647059
TNR (Tasa verdaderos negativos)	0,990106217
FPR (Tasa falsos positivos)	0,009893783
FNR (Tasa falsos negativos)	0,882352941
AUC	0,553876638

Cundo reducimos la contaminación ocurre lo mismo que para las demás pruebas, el algoritmo tiende a la aleatoriedad no dando buenos resultados.

- LOF

Los datos obtenidos de verdaderos y falsos que derivan en las métricas son los siguientes.

Anomalías para el 1%	VP	VN	FP	FN
371	0	36723	371	34

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0
TNR (Tasa verdaderos negativos)	0,989998382
FPR (Tasa falsos positivos)	0,010001618
FNR (Tasa falsos negativos)	1
AUC	0,494999191

También ocurre lo mismo para el algoritmo LOF siendo aleatorio, esto es debido a que en el intervalo bajo de anomalías no detectamos ninguna correcta.

Comparación entre algoritmos

Analizamos ahora los resultados en conjunto para la última prueba de la familia Exploit Kit.

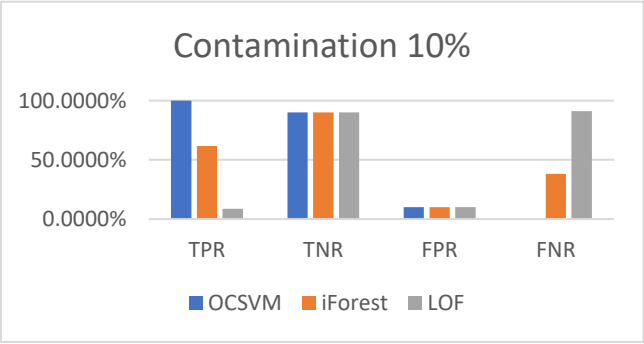

Figura 14. RIGEK3 10%

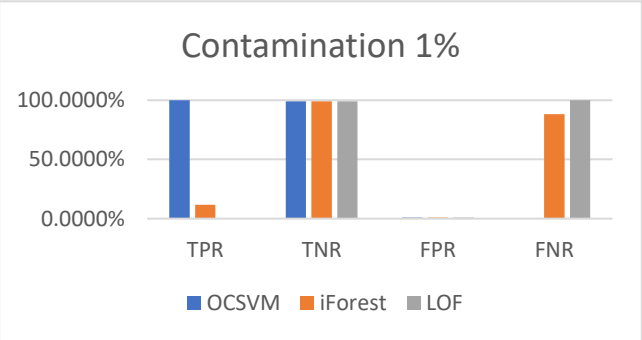

Figura 15. RIGEK3 1%

Por último, podemos corroborar viendo los resultados que con diferentes ataques para la familia Exploit Kit el algoritmo que mejores resultados da es el OCSVM. En todos los casos

hemos obtenidos valores altos de TPR y TNR, y bajos de FPR y FNR dando lugar a esos resultados óptimos.

Los valores de AUC obtenidos son altos, ya que como vemos en resumen con este algoritmo siempre hemos detectado todas las anomalías tengamos una contaminación alta o baja.

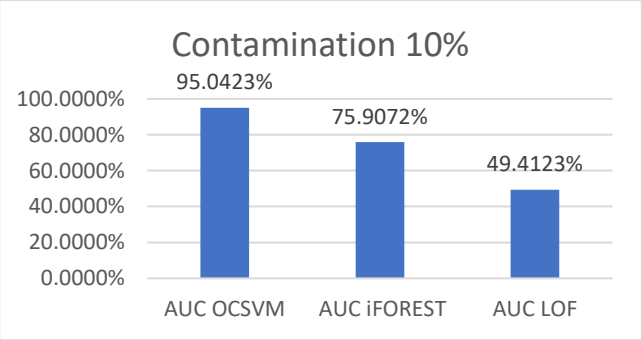

Figura 16. RIGEK3 10%

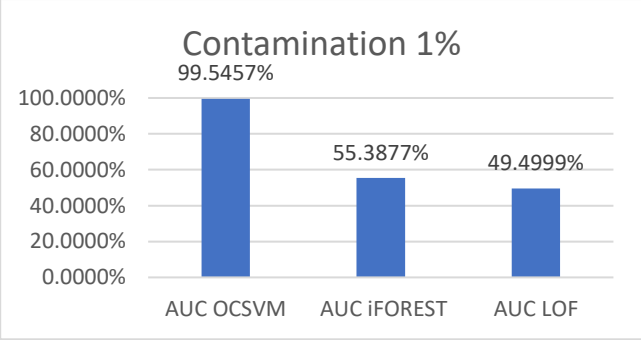

Figura 17. RIGEK3 1%

Con estas graficas podemos apreciar la veracidad de los expuesto antes viendo la calidad que posee el algoritmo al detectar.

6.2.3 2º Familia de tráfico malicioso:

Ahora se van a seleccionar 4 tipos diferentes de malware de la familia ransomware. Este tipo de malware cifran los archivos en el host infectado solicitando un rescate normalmente en bitcoins para descifrarlo. Funciona infectando los equipos mediante enlaces maliciosos o archivos adjunto en correo, webs… El problema mayor viene en los esquemas de cifrado de estos archivos los cuales son muy seguros siendo difícil descifrar sin recurrir a pagar[40].

En este caso vamos a estudiar 4 ejemplos diferentes del tipo Jaff que es un cryptovirus ransomware. Como característica principal Jaff encripta los archivos con la terminación .jaff, .sVn o .wlu en el terminal infectado. El contagio se realizara a través de archivos .wsf, .pdf, .word[41]…

6.2.3.1 1º Prueba, envío Jaff ransomware en archivo .wsf mediante spam en el correo

Características tráfico usado

Para este primer caso el equipo será infectado mediante un archivo .wfs (windows script file), que es un script de Windows. Llegará mediante un correo basura comprimido en un .zip y al ejecutarlo encriptará los archivos del equipo. Los archivos serán encriptados con la terminación .sVn.

Propiedades del malware[42]:

- Ransomware:

SHA256 hash: 001268d7fad7806705b3710ccc8cfffb2c2cfb830a273d7a0f87a5fa6422b9f5
Tamaño archivo: 174,592 bytes
Descripción archivo: Jaff ransomware

IPs anómalas a detectar:

119.28.98.205	Puerto 80	GET /a5/
198.23.48.27	Puerto 80	Tráfico Jaff ransomware después de la infección

Aplicación de algoritmos y resultados

Para este primer caso de la familia ransomware vamos a utilizar el siguiente dataset:

Flujos totales	Flujos después proc	Anomalías después proc	% Anomalías después proc
70012	56169	3	0,005341%

Para esta segunda familia vamos a realizar también la comparación entre los tres algoritmos seleccionados para una contaminación del 10% y del 1%.

Contaminación 10%

Analizamos los diferentes algoritmos seleccionados.

- OneClassSVM

Igual que para la anterior familia de malware ejecutamos el algoritmo obteniendo los siguientes valores de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos.

Anomalías para el 10%	VP	VN	FP	FN
5616	3	50553	5613	0

Métricas	Resultados
TPR (Tasa verdaderos positivos)	1
TNR (Tasa verdaderos negativos)	0,900064096
FPR (Tasa falsos positivos)	0,099935904
FNR (Tasa falsos negativos)	0
AUC	0,950032048

Podemos apreciar que, aunque hayamos cambiado de familia de malware el algoritmo OCSVM sigue obteniendo tasas altas.

- IForest

Vemos los valores obtenidos junto con las métricas.

Anomalías para el 10%	VP	VN	FP	FN
5616	3	50553	5613	0

Métricas	Resultados
TPR (Tasa verdaderos positivos)	1
TNR (Tasa verdaderos negativos)	0,900064096
FPR (Tasa falsos positivos)	0,099935904
FNR (Tasa falsos negativos)	0
AUC	0,950032048

En este caso para la familia ransomware el algoritmo Isolation Forest obtiene tasas altas, veamos si las puedes mejorar al bajar la contaminación o empeoran.

- LOF

Obtenemos los valores y las métricas después de ejecutar el algoritmo.

Anomalías para el 10%	VP	VN	FP	FN
5616	3	50553	5613	0

Métricas	Resultados
TPR (Tasa verdaderos positivos)	1
TNR (Tasa verdaderos negativos)	0,900064096
FPR (Tasa falsos positivos)	0,099935904
FNR (Tasa falsos negativos)	0
AUC	0,950032048

Veremos como evolucionan estos valores al bajar la contaminación.

Contaminación 1%

Reducimos el valor de la contaminación al 1% y realizamos los mismos pasos.

- OneClassSVM

Ejecutamos el algoritmo obteniendo los siguientes valores de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos.

Anomalías para el 1%	VP	VN	FP	FN
561	3	55608	558	0

Métricas	Resultados
TPR (Tasa verdaderos positivos)	1
TNR (Tasa verdaderos negativos)	0,990065164
FPR (Tasa falsos positivos)	0,009934836
FNR (Tasa falsos negativos)	0
AUC	0,995032582

Podemos corroborar lo visto para contaminaciones bajas, el algoritmo OCSVM se comporta también muy bien para la familia ransomware.

- IForest

Tenemos los siguientes valores y métricas.

Anomalías para el 1%	VP	VN	FP	FN
561	0	55605	561	3

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0
TNR (Tasa verdaderos negativos)	0,990011751
FPR (Tasa falsos positivos)	0,009988249
FNR (Tasa falsos negativos)	1
AUC	0,495005875

Por desgracia vemos que cuando reducimos la contaminación el algoritmo deja de detectar las anomalías siendo equiparable a la aleatoriedad.

- LOF

Para el último algoritmo tenemos los siguientes valores.

Anomalías para el 1%	VP	VN	FP	FN
561	2	55607	559	1

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0,666666667
TNR (Tasa verdaderos negativos)	0,99004736
FPR (Tasa falsos positivos)	0,00995264
FNR (Tasa falsos negativos)	0,333333333
AUC	0,828357013

Al contrario que ocurre para el algoritmo iForest vemos en este el AUC empeora un poco, aunque sigue siendo un valor aceptable.

Comparación entre algoritmos

Volvemos a analizar los resultados en global con diferentes gráficas para este primer caso de la familia ransomware.

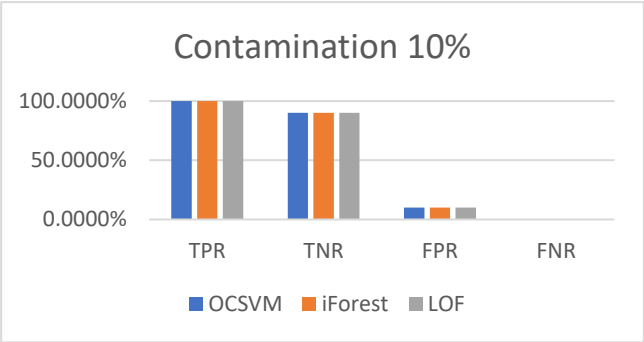

Figura 18. Jaff110%

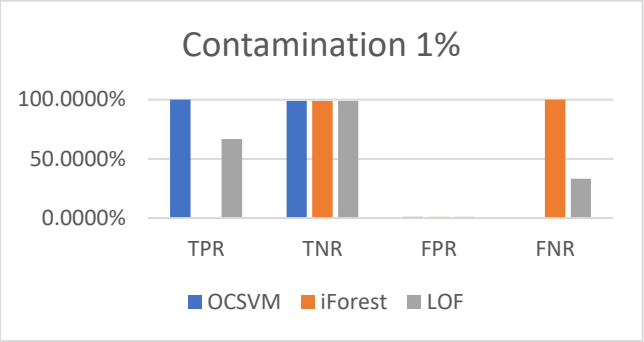

Figura 19. Jaff1 1%

Como comentábamos antes para el otro caso de estudio para obtener valores buenos en la detección debemos tener unos porcentajes altos de TPR y TNR, mientras que de FPR y FNR deben ser bajos.

Vemos en resumidas cuentas que cuando tenemos contaminaciones altas los algoritmos van a funcionar bien al tener más rango de anomalías para detectar, pero cuando reducimos la contaminación apreciamos que el meor algoritmo será el OCSVM.

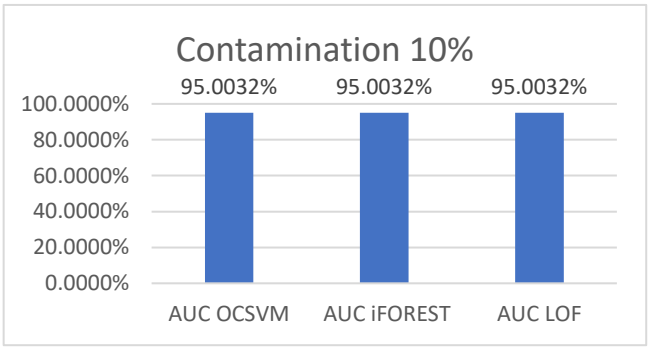

Figura 20. Jaff1 10%

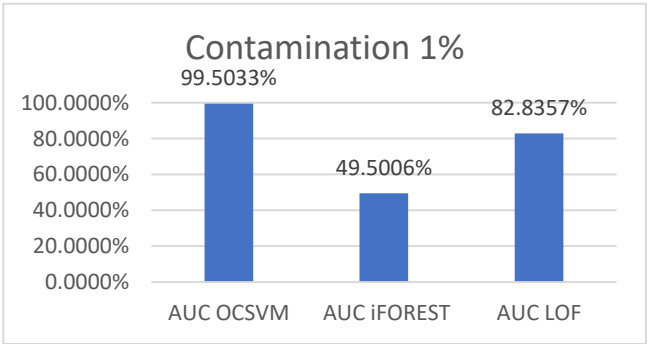

Figura 21. Jaff1 1%

El AUC crecerá al bajar la contaminación para el algoritmo OCSVM, lo mismo que ocurría para la familia de Exploit Kit. Esto es buena señal ya que podemos ir vislumbrando que el procesamiento junto con el algoritmo elegido funciona muy bien con diferentes familias de malware, esto lo intentaremos corroborar con los siguientes ejemplos de ransomware.

6.2.3.2 2 º prueba, envío Jaff ransomware en archivo Word dentro de un archivo .pdf mediante spam en el correo (1º tanda)

Características tráfico usado

Ahora Jaff ransomware se propaga a través de un archivo Word que habilita las macros. Este archivo a su vez vendrá dentro de un .pdf el cual estará adjunto el correo spam. En este caso la encriptación tendrá terminación .wlu.

Propiedades del malware[43]:

- Ransomware:

SHA256 hash: 824901dd0b1660f00c3406cb888118c8a10f66e3258b5020f7ea289434618b13
Tamaño archivo: 251,904 bytes
Dirección archivo: C:\Users\[username]\AppData\Local\Temp\bruhadson8.exe

IPs anómalas a detectar:

5.101.66.85	Puerto 80	GET /a5/
211.174.62.52	Puerto 80	Tráfico Jaff ransomware después de la infección

Aplicación de algoritmos y resultados

Para este caso vamos a utilizar el siguiente dataset:

Flujos totales	Flujos después proc	Anomalías después proc	% Anomalías después proc
55008	47630	2	0,004199%

Comparamos entre los tres algoritmos seleccionados para una contaminación del 10% y del 1%.

Contaminación 10%

Analizamos los diferentes algoritmos seleccionados.

- OneClassSVM

Obtenemos los valores de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos junto con sus métricas.

Anomalías para el 10%	VP	VN	FP	FN
4763	2	42867	4761	0

Métricas	Resultados
TPR (Tasa verdaderos positivos)	1
TNR (Tasa verdaderos negativos)	0,900037793
FPR (Tasa falsos positivos)	0,099962207
FNR (Tasa falsos negativos)	0
AUC	0,950018896

Tenemos un AUC alto similar a las demás pruebas realizadas.

- IForest

Vemos los valores y métricas obtenidas.

Anomalías para el 10%	VP	VN	FP	FN
4763	2	42867	4761	0

Métricas	Resultados
TPR (Tasa verdaderos positivos)	1
TNR (Tasa verdaderos negativos)	0,900037793
FPR (Tasa falsos positivos)	0,099962207
FNR (Tasa falsos negativos)	0
AUC	0,950018896

Al igual que antes el algoritmo Isolation Forest obtiene tasas altas, pero seguramente bajaran cuando disminuyamos la contaminación.

- LOF

Tenemos los siguientes valores y métricas.

Anomalías para el 10%	VP	VN	FP	FN
4763	2	42867	4761	0

Métricas	Resultados
TPR (Tasa verdaderos positivos)	1
TNR (Tasa verdaderos negativos)	0,900037793
FPR (Tasa falsos positivos)	0,099962207
FNR (Tasa falsos negativos)	0
AUC	0,950018896

A priori obtenemos buenos resultados a expensas de ver el comportamiento a menos contaminación.

Contaminación 1%

Reducimos el valor de la contaminación al 1% y realizamos los mismos pasos.

- OneClassSVM

Los valores y métricas son estos.

Anomalías para el 1%	VP	VN	FP	FN
476	2	47154	474	0

Métricas	Resultados
TPR (Tasa verdaderos positivos)	1

TNR (Tasa verdaderos negativos)	0,990047871
FPR (Tasa falsos positivos)	0,009952129
FNR (Tasa falsos negativos)	0
AUC	0,995023936

Otra vez tenemos buenas noticias para contaminaciones bajas, ya que el algoritmo OCSVM se comporta muy bien para la familia ransomware.

- IForest

Tenemos los siguientes valores y métricas.

Anomalías para el 1%	VP	VN	FP	FN
476	1	47153	475	1

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0,5
TNR (Tasa verdaderos negativos)	0,990026875
FPR (Tasa falsos positivos)	0,009973125
FNR (Tasa falsos negativos)	0,5
AUC	0,745013437

Al detectar una anomalía verdadera en ese rango pequeño de anomalías obtenemos unas tasas más o menos normales.

- LOF

Tenemos los valores y las métricas.

Anomalías para el 1%	VP	VN	FP	FN
476	1	47153	475	1

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0,5
TNR (Tasa verdaderos negativos)	0,990026875
FPR (Tasa falsos positivos)	0,009973125
FNR (Tasa falsos negativos)	0,5
AUC	0,745013437

Vemos que el algoritmo LOF al bajar la contaminación decrementa su calidad detectando peor las anomalías.

Comparación entre algoritmos

Volvemos a analizar los resultados en global con diferentes gráficas para este primer caso de la familia ransomware.

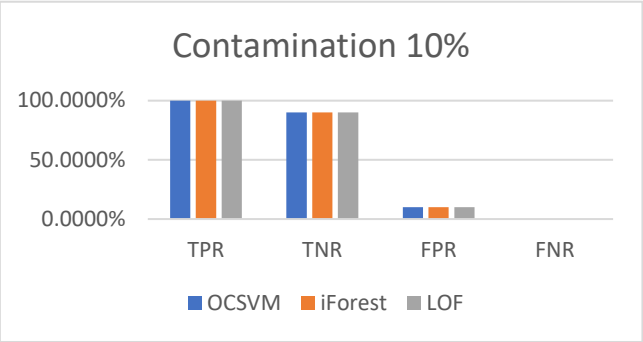

Figura 22. Jaff2 10%

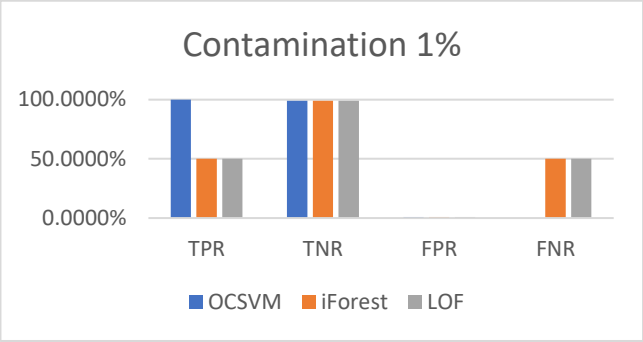

Figura 23. Jaff2 1%

Se sigue un poco línea de lo comentado antes, ya que el algoritmo OCSVM sigue funcionando realmente bien además de equipararse a él en resultados el algoritmo LOF para este tipo de ransomware. El algoritmo iForest será el que peor resultados dará para esta prueba

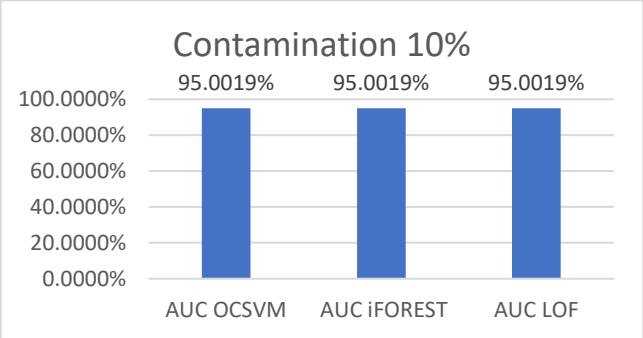

Figura 24. Jaff2 10%

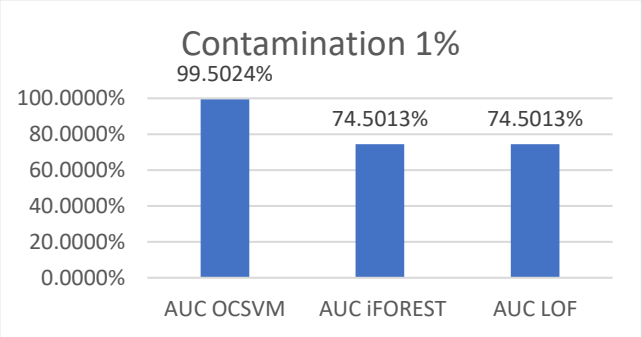

Figura 25. Jaff2 10%

Vemos que los valores de AUC son bastantes buenos para los tres algoritmos en ese caso de ransomware, aunque empeoran el algoritmo iForest y el LOF al reducir la contaminación. Vamos a ver que ocurre en la última prueba para esta familia ransomware si se siguen obteniendo estos resultados.

6.2.3.3 3º prueba, envío Jaff ransomware en archivo Word dentro de un archivo .pdf mediante spam en el correo (2º tanda)

Características tráfico usado

Tenemos un ejemplo similar al anterior enviado desde otro tipo de correo. En este caso la encriptación tendrá también la terminación .wlu.

Propiedades del malware[43]:

- Ransomware:

SHA256 hash: 2cc1d8edc318e0e09aad6afbc48999980f8e39e54734bca4c1a95c7b5db39569
Tamaño archivo: 217,088 bytes
Dirección archivo: C:\Users\[username]\AppData\Local\Temp\bruhadson8.exe

IPs anómalas a detectar:

34.225.214.20	Puerto 80	GET /a5/
195.130.247.50	Puerto 80	Tráfico Jaff ransomware después de la infección

Aplicación de algoritmos y resultados

Para este caso de la familia ransomware vamos a utilizar el siguiente dataset:

Flujos totales	Flujos después proc	Anomalías después proc	% Anomalías después proc
114990	84858	2	0,002357%

Comparamos entre los tres algoritmos seleccionados para una contaminación del 10% y del 1%.

Contaminación 10%

Analizamos para este último caso los diferentes algoritmos seleccionados.

- OneClassSVM

Tenemos los siguientes valores de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos junto con sus métricas.

Anomalías para el 10%	VP	VN	FP	FN
8485	2	76373	8483	0

Métricas	Resultados
TPR (Tasa verdaderos positivos)	1
TNR (Tasa verdaderos negativos)	0,90003064
FPR (Tasa falsos positivos)	0,09996936
FNR (Tasa falsos negativos)	0
AUC	0,95001532

Se obtiene un AUC alto similar a las demás pruebas realizadas.

- IForest

Estos son los valores y métricas obtenidas.

Anomalías para el 10%	VP	VN	FP	FN
8485	2	76373	8483	0

Métricas	Resultados
TPR (Tasa verdaderos positivos)	1
TNR (Tasa verdaderos negativos)	0,90003064
FPR (Tasa falsos positivos)	0,09996936
FNR (Tasa falsos negativos)	0
AUC	0,95001532

Obtenemos tasas altas a priori para contaminaciones altas, habrá que ver la evolución en contaminaciones bajas.

- LOF

Tenemos los siguientes valores y métricas.

Anomalías para el 10%	VP	VN	FP	FN
8485	2	76373	8483	0

Métricas	Resultados
TPR (Tasa verdaderos positivos)	1
TNR (Tasa verdaderos negativos)	0,90003064
FPR (Tasa falsos positivos)	0,09996936
FNR (Tasa falsos negativos)	0
AUC	0,95001532

Al igual que para el caso anterior obtenemos buenos resultados a priori a expensas de ver el comportamiento con menos contaminación.

Contaminación 1%

Reducimos el valor de la contaminación al 1% y realizamos los mismos pasos.

- OneClassSVM

Los valores y métricas son estos.

Anomalías para el 1%	VP	VN	FP	FN
848	2	84010	846	0

Métricas	Resultados
TPR (Tasa verdaderos positivos)	1
TNR (Tasa verdaderos negativos)	0,990030169
FPR (Tasa falsos positivos)	0,009969831
FNR (Tasa falsos negativos)	0
AUC	0,995015084

Obtenemos buenos resultados como esperamos, el algoritmo OCSVM se comporta muy bien para todos los malwares analizados.

- IForest

Obtenemos los siguientes valores y métricas.

Anomalías para el 1%	VP	VN	FP	FN
848	1	84009	847	1

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0,5
TNR (Tasa verdaderos negativos)	0,990018384
FPR (Tasa falsos positivos)	0,009981616
FNR (Tasa falsos negativos)	0,5
AUC	0,745009192

Para el algoritmo IForest vemos que se sigue la línea general de las demás pruebas empeorando al bajar la contaminación.

- LOF

Vemos para finalizar los valores y las métricas en el caso del algoritmo LOF.

Anomalías para el 1%	VP	VN	FP	FN
848	0	84008	848	2

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0
TNR (Tasa verdaderos negativos)	0,990006599
FPR (Tasa falsos positivos)	0,009993401
FNR (Tasa falsos negativos)	1
AUC	0,4950033

El algoritmo LOF para este caso no ofrecerá garantías ya que vemos la detección tiende a la aleatoriedad.

Comparación entre algoritmos

Volvemos a analizar los resultados en global con diferentes gráficas para este caso de la familia ransomware.

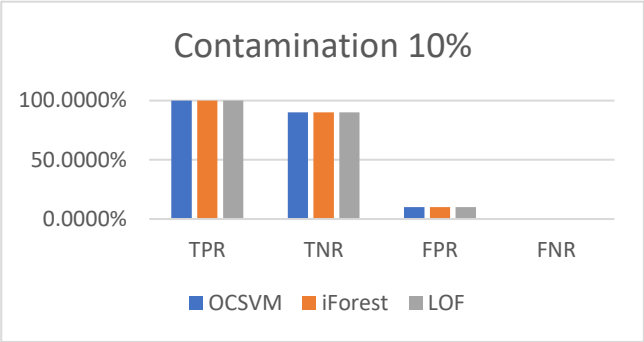

Figura 26. Jaff3 10%

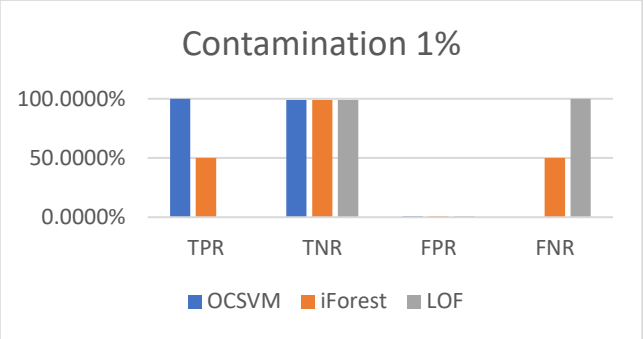

Figura 27. Jaff3 1%

Podemos seguir viendo para la familia ransomware que el mejor algoritmo para todas las pruebas es el OCSVM, dándonos unos resultados sorprendentemente altos a la hora de detectar anomalías.

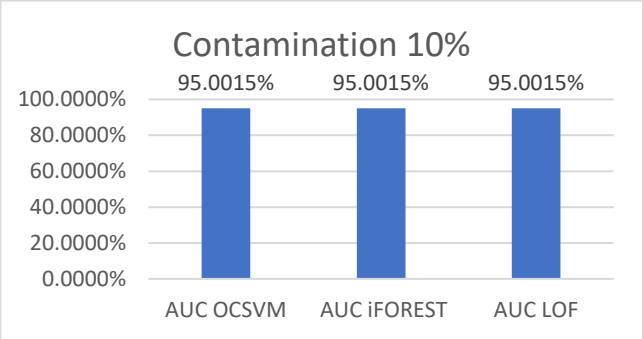

Figura 28. Jaff3 10% 1

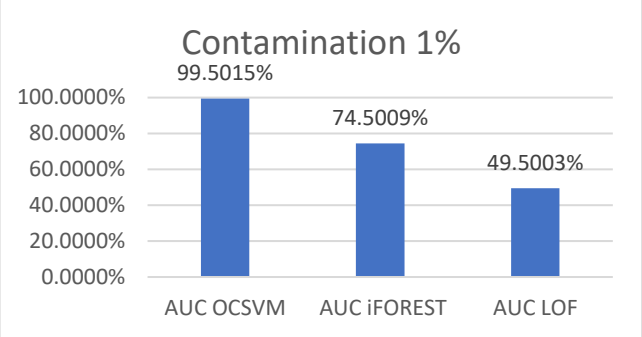

Figura 29. Jaff3 1%

Con los valores del AUC vemos realmente la calidad del algoritmo OCSVM y sus altas tasas de detección.

6.2.4 3º Familia de tráfico malicioso:

6.2.4.1 Envío Wannacry ransomware mediante exploit Eternalblue

Características tráfico usado

En esta tercera familia vamos a ver un caso de bastante actualidad que puso en jaque a medio mundo, el ransomware wannacry. Este tipo de ransomware tiene como característica principal que le difiere de los demás que se propaga por la red aprovechando una vulnerabilidad en el protocolo SMB, lo que hace que se propague por la red hacia diferentes terminales[44]. El

exploit que aprovecha este fallo de seguridad en el protocolo SMB será el Eternalblue. En este caso la encriptación tendrá la terminación .wncry[45].

Propiedades del malware[44]:

- Ransomware:

SHA256 hash: ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa

Dirección archivo: C:\Users\[username]\AppData\Local\Temp\ @WanaDecryptor@.exe

IPs anómalas a detectar:

10.128.0.243	Puerto 445	Tráfico Wannacry ransomware.

Aplicación de algoritmos y resultados

Para esta nueva familia de ransomware vamos a utilizar el siguiente dataset:

Flujos totales	Flujos después proc	Anomalías después proc	% Anomalías después proc
55005	37762	78	0,206557%

Comparamos entre los tres algoritmos seleccionados para una contaminación del 10% y del 1%.

Contaminación 10%

Analizamos los diferentes algoritmos seleccionados.

- OneClassSVM

Analizamos como para todos los casos el algoritmo OCSVM y obtenemos los valores de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos junto con sus métricas.

Anomalías para el 10%	VP	VN	FP	FN
3776	78	33986	3698	0

Métricas	Resultados
TPR (Tasa verdaderos positivos)	1
TNR (Tasa verdaderos negativos)	0,901868167
FPR (Tasa falsos positivos)	0,098131833
FNR (Tasa falsos negativos)	0
AUC	0,950934083

Se mantienen las métricas altas para wannacry, veremos si al ajustar la contaminación a un valor más bajos siguen estas tasas buenas.

- IForest

Vemos los valores y métricas obtenidas.

Anomalías para el 10%	VP	VN	FP	FN
3776	78	33986	3698	0

Métricas	Resultados
TPR (Tasa verdaderos positivos)	1
TNR (Tasa verdaderos negativos)	0,901868167
FPR (Tasa falsos positivos)	0,098131833
FNR (Tasa falsos negativos)	0
AUC	0,950934083

Siguiendo la línea de algunos ejemplos anteriores el algoritmo iForest se comporta bastante bien para contaminaciones altas.

- LOF

Vemos los valores y las métricas asociadas para este caso.

Anomalías para el 10%	VP	VN	FP	FN
3776	9	33917	3767	69

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0,115384615
TNR (Tasa verdaderos negativos)	0,900037151
FPR (Tasa falsos positivos)	0,099962849
FNR (Tasa falsos negativos)	0,884615385
AUC	0,507710883

El algoritmo LOF por el contrario se comporta bastante mal ya que tiende a la aleatoriedad.

Contaminación 1%

Reducimos el valor de la contaminación al 1% y vemos que valores obtenemos.

- OneClassSVM

Los valores y métricas son estos.

Anomalías para el 1%	VP	VN	FP	FN
377	78	37385	299	0

Métricas	Resultados
TPR (Tasa verdaderos positivos)	1
TNR (Tasa verdaderos negativos)	0,992065598

FPR (Tasa falsos positivos)	0,007934402
FNR (Tasa falsos negativos)	0
AUC	0,996032799

Obtenemos el mejor valor de AUC del conjunto de todas las pruebas para el caso del wannacry, esto nos dice que el algoritmo OCSVM nos da muy buenas resultados también para esta familia relacionada con la familia ransomware estudiada anteriormente.

- IForest

Vemos los valores y las métricas que hemos obtenido.

Anomalías para el 1%	VP	VN	FP	FN
377	0	37307	377	78

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0
TNR (Tasa verdaderos negativos)	0,989995754
FPR (Tasa falsos positivos)	0,010004246
FNR (Tasa falsos negativos)	1
AUC	0,494997877

Aunque el algoritmo iForest daba buenos resultados para contaminaciones altas, en cuanto hemos bajado la contaminación el algoritmo ha perdido calidad y tiende a la aleatoriedad.

- LOF

Tenemos los valores y métricas para el último algoritmo.

Anomalías para el 1%	VP	VN	FP	FN
377	0	37307	377	78

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0
TNR (Tasa verdaderos negativos)	0,989995754
FPR (Tasa falsos positivos)	0,010004246
FNR (Tasa falsos negativos)	1
AUC	0,494997877

Igual que ocurre para el algoritmo iForest tiende a la aleatoriedad.

Comparación entre algoritmos

Analizamos los resultados en global con diferentes gráficas para el ransomware wannacry.

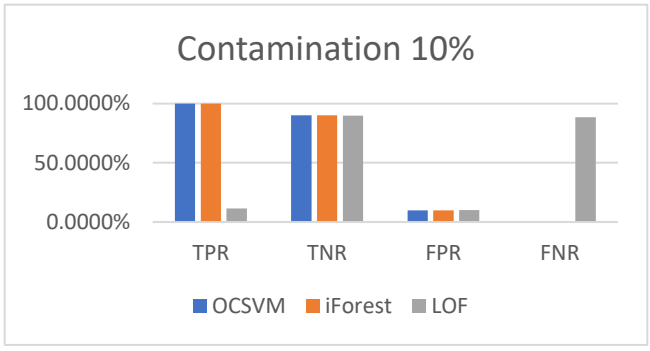

Figura 30. Wannacry 10%

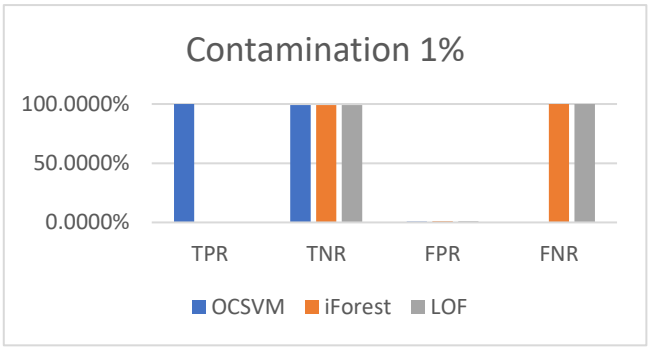

Figura 31. Wannacry 1%

Se mantienen los valores altos en TPR y TNR, y bajos en FPR y FNR para el algoritmo OCSVM lo que nos lleva a tener buenos resultados de AUC.

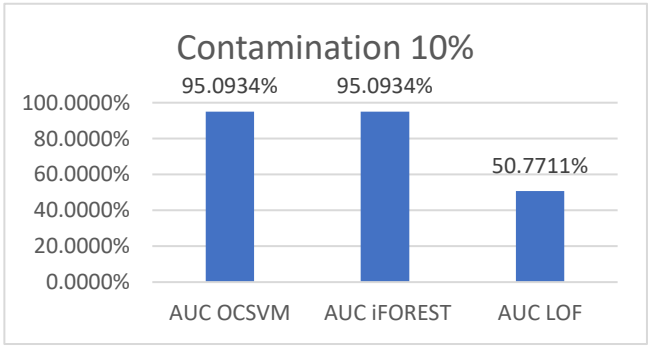

Figura 32. Wannacry 10%

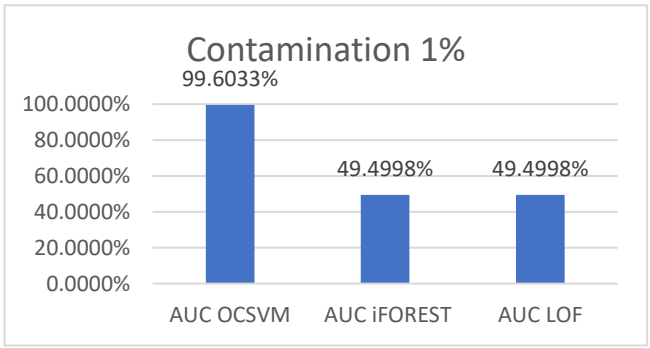

Figura 33. Wannacry 1%

Con los valores del AUC podemos comprobar que se mantiene el buen funcionamiento del algoritmo OCSVM, aumentando su valor cuando decrementamos la contaminación. Los algoritmos iForest y LOF funcionan de manera bastante irregular dando unos valores bastante malos.

Que se mantengan los buenos resultados para el algoritmo wannacry son muy buenas noticias, ya que al ser malware de última actualidad viene a corroborar la finalidad de los algoritmos de machine learning para detectar anomalías, aunque tengamos un malware nuevo, al no ser como otros sistemas típicos de detección de firmas detectará el malware como anómalo.

6.2.5 4º Familia de tráfico malicioso:

Por último, se ha elegido como malware a estudiar el tráfico de fuerza bruta a ssh mediante honeypot. Cuando un atacante intenta acceder a un terminal mediante el protocolo ssh y no dispone del usuario y contraseña realizará ataques de fuerza bruta para conseguirlas. Una vez consigue acceder al sistema podrá introducir el malware que vea oportuno[46].

Este tráfico de fuerza bruta será generado mediante una honeypot.

6.2.5.1 Tráfico fuerza bruta a ssh

Características tráfico usado

En este caso el tráfico que estudiaremos únicamente serán las diferentes pruebas de usuario y contraseña shh, sin incluir malware después de acceder al sistema.

IPs anómalas a detectar:

172.16.1.230	Puerto 22	Pruebas credenciales.

Aplicación de algoritmos y resultados

Para esta nueva familia vamos a utilizar el siguiente dataset:

Flujos totales	Flujos después proc	Anomalías después proc	% Anomalías después proc
105648	54287	100	0,184206%

Seguimos el mismo procedimiento utilizado en las familias anteriores y comparamos entre los tres algoritmos seleccionados para una contaminación del 10% y del 1%.

Contaminación 10%

Analizamos los diferentes algoritmos seleccionados.

- OneClassSVM

Analizamos los valores y las métricas que obtenemos.

Anomalías para el 10%	VP	VN	FP	FN
5428	6	48765	5422	94

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0,06
TNR (Tasa verdaderos negativos)	0,8999391
FPR (Tasa falsos positivos)	0,1000609
FNR (Tasa falsos negativos)	0,94
AUC	0,47996955

Por primera vez para la familia de tráfico fuerza bruta a ssh obtenemos malos resultados con el algoritmo OCSVM. Veremos cómo evoluciona el AUC para otras contaminaciones, pero a priori no tiene buena pinta.

- IForest

Los valores y las métricas son los expuestos.

Anomalías para el 10%	VP	VN	FP	FN
5428	15	48775	5412	85

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0,15
TNR (Tasa verdaderos negativos)	0,900123646
FPR (Tasa falsos positivos)	0,099876354
FNR (Tasa falsos negativos)	0,85
AUC	0,525061823

Al cambiar de algoritmo y estudiar el iForest vemos que se mantienen los valores malos.

- LOF

Para el algoritmo LOF tenemos los siguientes calores y métricas.

Anomalías para el 10%	VP	VN	FP	FN
5428	34	48893	5294	66

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0,34
TNR (Tasa verdaderos negativos)	0,90230129
FPR (Tasa falsos positivos)	0,09769871
FNR (Tasa falsos negativos)	0,66
AUC	0,621150645

Para contaminaciones altas el único algoritmo que no ofrece resultados que tienden a la aleatoriedad es el LOF, este tráfico por tanto estamos viendo que se comporta de una manera muy diferente a las demás familias, veremos cómo evoluciona al reducir la contaminación.

Contaminación 1%

Analizamos para esta contaminación los diferentes algoritmos seleccionados.

- OneClassSVM

Tenemos los siguientes valores de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos junto con sus métricas.

Anomalías para el 1%	VP	VN	FP	FN
542	1	53646	541	99

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0,01
TNR (Tasa verdaderos negativos)	0,990016056
FPR (Tasa falsos positivos)	0,009983944
FNR (Tasa falsos negativos)	0,99
AUC	0,500008028

Al reducir la contaminación seguimos con unos muy malos iguales a si fuera aleatorio el sistema. Podemos decir que está familia no sigue la línea de las demás familias analizadas para el algoritmo OCSVM. Esto se puede deber a la naturaleza de este tipo de tráfico que hace que tengamos que realizar otro procesamiento de los datos o probar más algoritmos.

- iForest

Los siguientes valores y métricas serán las obtenidas.

Anomalías para el 1%	VP	VN	FP	FN
542	5	53650	537	95

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0,05
TNR (Tasa verdaderos negativos)	0,990089874
FPR (Tasa falsos positivos)	0,009910126
FNR (Tasa falsos negativos)	0,95
AUC	0,520044937

Obtenemos unos resultados bastante malos para este algoritmo también.

- LOF

Los valores y las métricas son los siguientes.

Anomalías para el 1%	VP	VN	FP	FN
542	0	53645	542	100

Métricas	Resultados
TPR (Tasa verdaderos positivos)	0
TNR (Tasa verdaderos negativos)	0,989997601
FPR (Tasa falsos positivos)	0,010002399
FNR (Tasa falsos negativos)	1
AUC	0,4949988

Aunque obteníamos métricas mejores en esta familia para el algoritmo LOF en contaminaciones altas, vemos que al disminuirla sigue la línea de los demás algoritmos y tiende a la aleatoriedad.

Comparación entre algoritmos

Analizamos los resultados en global con diferentes gráficas para el tráfico ssh.

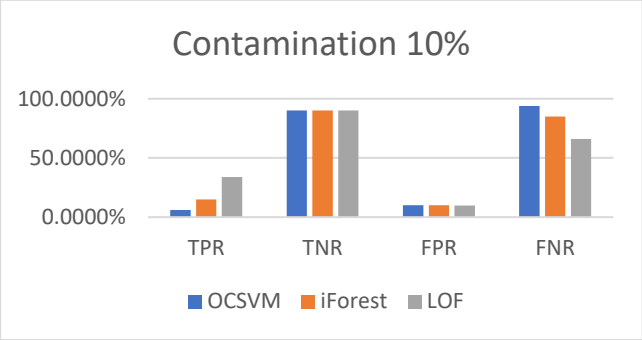

Figura 34. SSH 10%

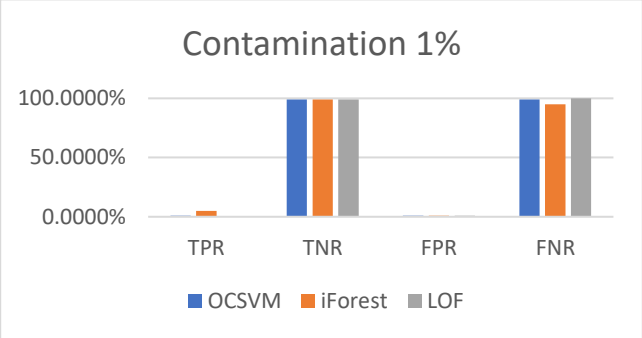

Figura 35. SSH 1%

Vemos que los resultados son bastantes malos para el tráfico de fuerza bruta a ssh. Sobre todo, es curioso los resultados para el algoritmo OCSCVM, ya que obtenía valores muy altos de AUC para las familias anteriormente estudiadas, pero en este caso puede ser debido a la topología del tráfico al ser pruebas de conexiones lo toma como tráfico no anómalo el algoritmo.

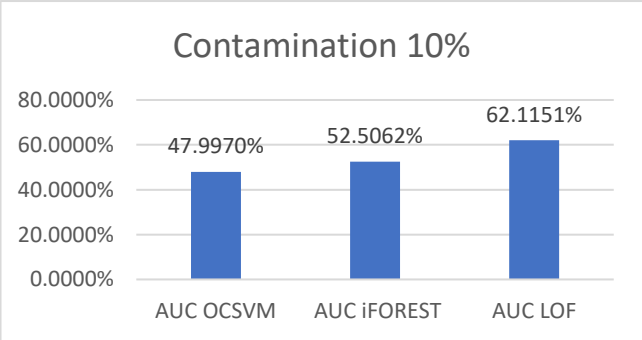

Figura 36. SSH 10%

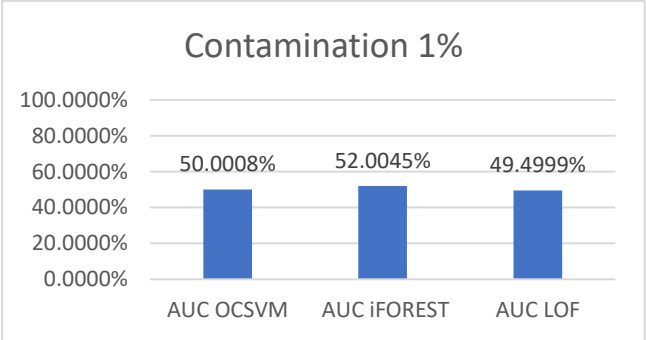

Figura 37. SSH 1%

Los valores del AUC obtenidos podemos apreciar que son bastantes malos.

7 CONCLUSIONES

Después de realizar un estudio con diferentes tipos de tráficos maliciosos hemos llegado a la conclusión que el algoritmo que mejores resultados da con el procesamiento de los datos realizados es el OneClassSVM. Sus tasas son muy altas para todos los malwares de diferentes tipos analizados lo que nos lleva a buenas noticias en la elección del preprocesamiento de los datos para este algoritmo específico.

Los algoritmos iForest y LOF van a funcionar bien en determinados malwares, pero en otros su funcionamiento tiende a la aleatoriedad siendo inutilizables. Esto produce que no tengan tanta fiabilidad a la hora de intentar detectar anomalías en nuevos tráficos maliciosos, no como con el algoritmo OneClassSVM.

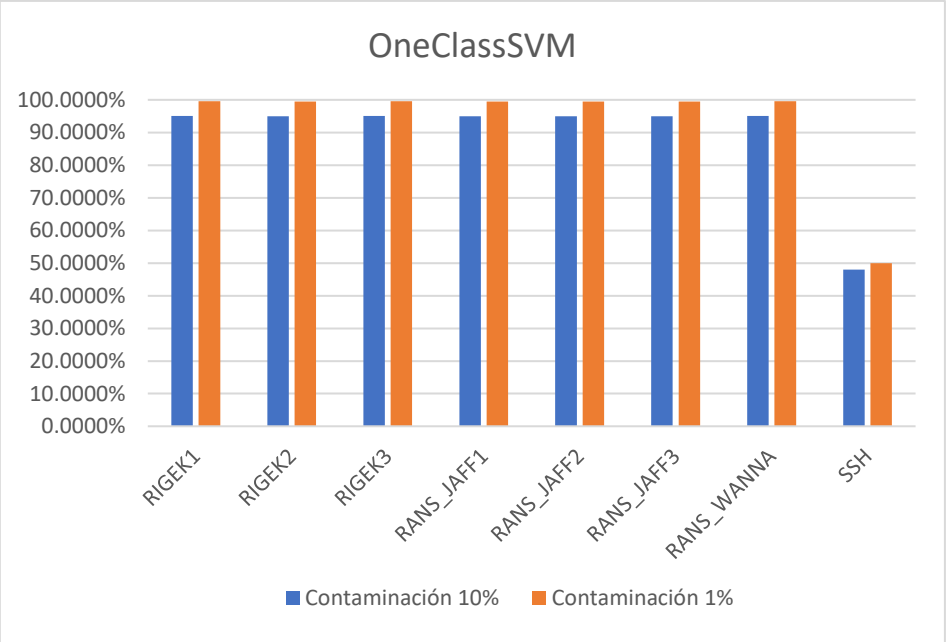

Figura 30. OCSVM

En esta gráfica podemos ver el resumen completo para el algoritmo OneClassSVM.

Se puede apreciar su buen funcionamiento ya que detecta anomalías con unos valores muy altos, y si disminuimos la contaminación suben esas tasas. Esto no se cumple para la familia de tráfico de fuerza bruta ssh ya que al tratarse de un tráfico más parecido a lo normal (realiza pruebas de credenciales) puede que el algoritmo lo asuma así. Pare resolver esto deberíamos probar otro procesamiento de los datos con diferentes características utilizadas o probar nuevos ejemplos con otro algoritmo empleados en la detección de anomalías.

Estos buenos resultados son reseñables ya que cuando estamos disminuyendo la contaminación estamos reduciendo el rango de anomalías para detectar, lo que significa que nuestras anomalías a detectar deben estar arriba en la tabla de mayores detecciones.

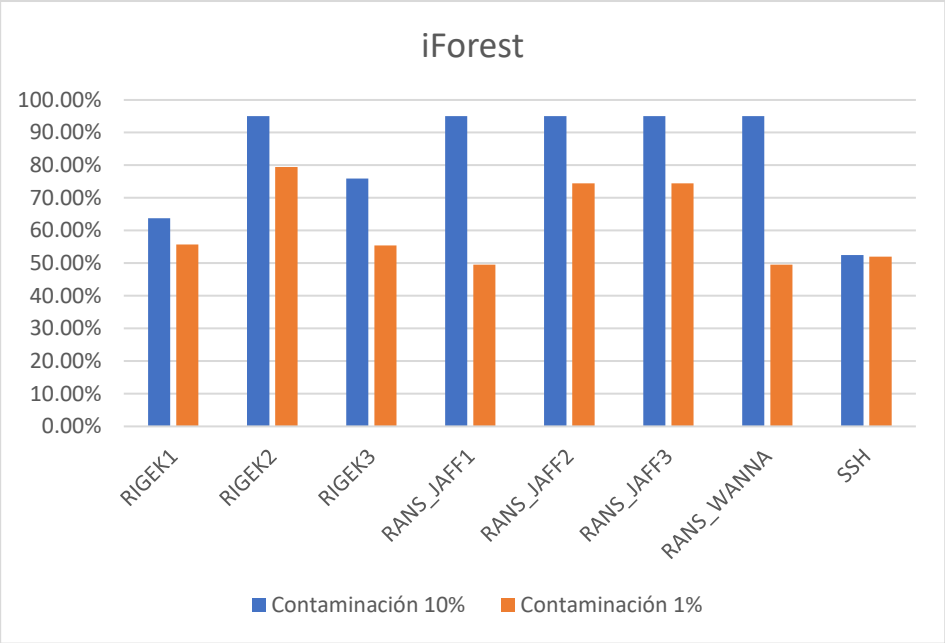

Figura 31. Isolation Forest

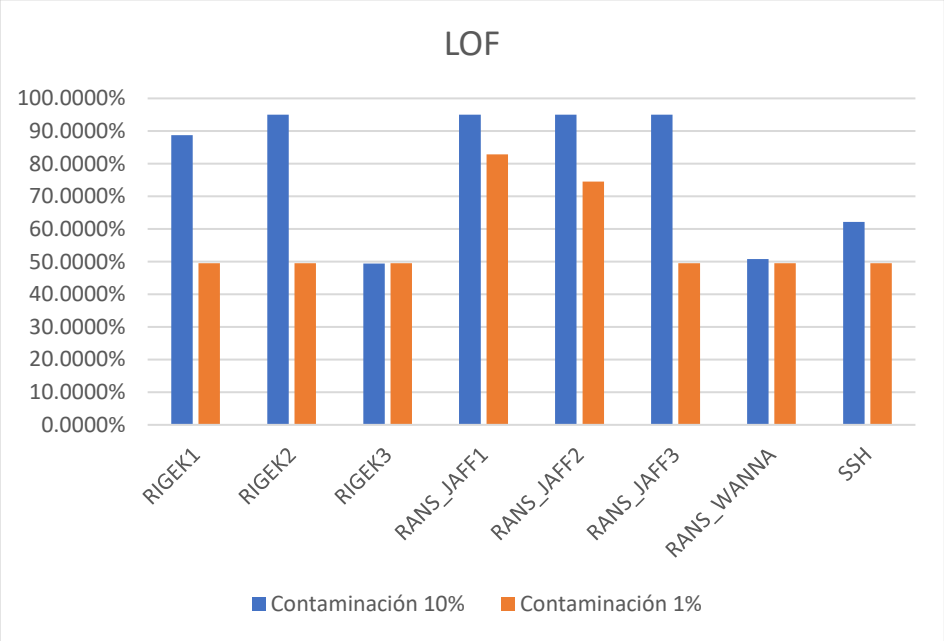

Figura 32. LOF

Para los otros dos algoritmos estudiamos podemos ver que funcionan bastante peor a la hora de detectar las anomalías, en un primer caso cuando tenemos una contaminación alta se puede suponer un buen funcionamiento para los dos en la mayoría de los malwares analizados, pero al bajar ese rango de anomalías detectables pierde calidad llegando a ser similar a la aleatoriedad en diversos malwares.

Al ser tan irregulares al bajar el rango de anomalías no vamos a poder utilizarlos con una fiabilidad buena en nuestros sistemas.

8 CÓDIGO PYTHON DESARROLLADO

El código Python que se ha desarrollado en los experimentos propuestos anteriormente está estructurado de la siguiente forma:

- **Script entrenamiento_algoritmo.py**: Este script se basará en cargar el algoritmo que vayamos a implementar para poder exportar el modelo de entrenamiento en formato pkl. Este modelo será cargado en el script de test.

  Los argumentos que se deberán pasar al ejecutar son:
  -i: Dataset sobre el que se quiera trabajar.
  -o: Nombre que daremos al modelo de entrenamiento exportado.

- **Script test_algoritmo.py**: En este script ejecutaremos el algoritmo que hemos elegido en el script de entrenamiento. Obtendremos un archivo .csv con las anomalías para la contaminación especificada y las diferentes métricas obtenidas.

  Los argumentos que se deberán pasar al ejecutar son:
  -i: Dataset sobre el que se quiera trabajar.
  -m: Modelo .pkl que hemos exportado en el entrenamiento
  -a: Archivo con las ips anómalas para poder calcular las métricas.
  -d: Archivo con las fechas que comprenden las ips anómalas para poder calcular las métricas.

- **Script clases_transformadoras.py**: Contendrá las clases transformadas explicadas anteriormente que se podrán desarrollar en los pipelines.

- **Archivo ips.txt**: Contendrá las ips anómalas de nuestro dataset para poder etiquetar los flujos anómalos y poder calcular las métricas automáticamente.

- **Archivo fechas.txt**: Contendrá las fechas que contienen las ips anómalas de nuestro dataset para poder etiquetar los flujos anómalos y poder calcular las métricas automáticamente.

Todo el código python desarrollado para este trabajo está expuesto en el siguiente enlace github:

https://github.com/ivangom14/Machine-Learning

9 LÍNEAS FUTURAS

Al ser un campo de investigación muy grande vamos a estar en constante desarrollo lo que nos hace tener muchas opciones abiertas para continuar estudiando diferentes casos. En cuanto a las líneas futuras a la vista de los buenos resultados obtenidos para el algoritmo OCSVM se abren varios frentes para seguir progresando en el futuro:

- Probar este procesamiento de datos y algoritmos implementados con tráfico de usuarios reales, esto nos dará unas métricas más reales al no usar entornos simulados como es el caso de nuestro trabajo.

- Generar diferentes tráficos de red maliciosos para ver si se siguen manteniendo estas tasas para los diferentes algoritmos.

- Implementar diferentes procesamientos de los datos y probar otros algoritmos, esto nos lleva a combinar diferentes tipos de algoritmos en cadena siendo supervisados y no supervisados.

- Utilizar diferentes frameworks como Apache Spark o Hadoop para gestionar grandes volúmenes de datos.

- Probar diferentes protocolos de red a parte de netflow que nos ofrezcan diferentes características para utilizar.

10 BIBLIOGRAFÍA

[1] J. V. Galán, "Aplicación de Técnicas de Aprendizaje Automático en el Sector Ferroviario Memoria," 2016.

[2] "Cuatro verdades sobre Machine Learning | TrendTIC." [Online]. Available: http://www.trendtic.cl/2017/01/cuatro-verdades-sobre-machine-learning/. [Accessed: 03-Sep-2017].

[3] D. Raquel Noblejas Sampedro TUTOR, D. A. Víctor Villagrá González, D. Enrique Vázquez Gallo Vocal, D. A. Víctor Villagrá González Secretario, D. José María del Álamo Ramiro Suplente, and D. Manuel Álvarez-Campana Fernández-Corredor, "TRABAJO FIN DE GRADO TÍTULO: Estudio de algoritmos de detección de anomalías y propuesta de recomendaciones para su aplicación a entornos de ciberseguridad," 2016.

[4] "Los mitos en torno al Machine Learning | Ebanking News." [Online]. Available: http://www.ebankingnews.com/noticias/loa-mitos-en-torno-al-machine-learning-0036743. [Accessed: 03-Sep-2017].

[5] "Anomaly/Novelty detection with scikit-learn." [Online]. Available: https://es.slideshare.net/agramfort/anomalynovelty-detection-with-scikitlearn. [Accessed: 03-Sep-2017].

[6] S. Ben-David and S. Shalev-Shwartz, *Understanding Machine Learning: From Theory to Algorithms*. 2014.

[7] S. Ortega Lázaro and S. (Tutor) Jiménez Fernández, "Predicción de extremos de viento en parques eólicos mediante técnicas de machine learning. Universidad de Alcalá Escuela Politécnica Superior," 2016.

[8] M. L. Toolkit, "ML-SPL Quick Reference Guide ML-SPL Quick Reference Guide Machine Learning Toolkit Predict Numeric Fields (Regression) Pr edict Categorical Fields (Classification) Detect Categorical Outliers Detect Data Detect Numeric Outliers OneClassSVM Preprocessi."

[9] "Netflow - EcuRed." [Online]. Available: https://www.ecured.cu/Netflow. [Accessed: 04-Sep-2017].

[10] D. Kerr, "NetFlow y su aplicación en seguridad," pp. 33–42.

[11] "NetFlow Version 1." [Online]. Available: https://www.plixer.com/support/netflow-v1/. [Accessed: 03-Sep-2017].

[12] "NetFlow Version 5." [Online]. Available: https://www.plixer.com/support/netflow-v5/. [Accessed: 03-Sep-2017].

[13] V. Carela-Espanol, P. Barlet-Ros, and J. Solé-Pareta, "Traffic classification with sampled netflow," *Peopleacupcedu*, vol. 33, no. 2, p. 34, 2009.

[14] Cisco Systems, "Cisco IOS NetFlow Version 9 Flow-Record Format," *White Pap.*, no. May, pp. 1–14, 2011.

[15] "NetFlow Version 9." [Online]. Available: https://www.plixer.com/support/netflow-v9/. [Accessed: 03-Sep-2017].

[16] S. Before and A. Do, "InfoSec Reading Room."

[17] C. Wagner, R. State, T. Engel, and M. Learning, “Machine Learning Approach for IP-Flow Record Anomaly Detection To cite this version : Machine Learning Approach for IP-Flow Record Anomaly Detection,” pp. 28–39, 2011.

[18] S. Omar, A. Ngadi, and H. H. Jebur, “Machine Learning Techniques for Anomaly Detection: An Overview,” *Int. J. Comput. Appl.*, vol. 79, no. 2, pp. 975–8887, 2013.

[19] “Impact of Machine Learning on 5G Use Cases and Scenarios,” no. 1.

[20] “CogNet ‹ 5G-PPP.” [Online]. Available: https://5g-ppp.eu/cognet/. [Accessed: 04-Sep-2017].

[21] “Generar recolectar y tratar Netflow « #4sysadmins.” [Online]. Available: https://nebul4ck.wordpress.com/2015/06/25/generar-recolectar-y-tratar-netflow/. [Accessed: 04-Sep-2017].

[22] “scikit-learn: machine learning in Python — scikit-learn 0.19.0 documentation.” [Online]. Available: http://scikit-learn.org/stable/. [Accessed: 04-Sep-2017].

[23] K. Heller, K. Svore, A. D. Keromytis, and S. Stolfo, “One class support vector machines for detecting anomalous windows registry accesses,” *Work. Data Min. Comput. Secur. (DMSEC), Melbourne, FL, Novemb. 19, 2003*, 2003.

[24] L. M. Manevitz, M. Yousef, N. Cristianini, J. Shawe-Taylor, and B. Williamson, “One-Class SVMs for Document Classification,” *J. Mach. Learn. Res.*, vol. 2, pp. 139–154, 2001.

[25] F. T. Liu and K. M. Ting, “Isolation Forest.”

[26] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, “LOF: Identifying Density-Based Local Outliers,” *Proc. 2000 Acm Sigmod Int. Conf. Manag. Data*, pp. 1–12, 2000.

[27] “Using scikit-learn Pipelines and FeatureUnions | zacstewart.com.” [Online]. Available: http://zacstewart.com/2014/08/05/pipelines-of-featureunions-of-pipelines.html. [Accessed: 04-Sep-2017].

[28] “Using Pipelines and FeatureUnions in scikit-learn - Michelle Fullwood.” [Online]. Available: http://michelleful.github.io/code-blog/2015/06/20/pipelines/. [Accessed: 04-Sep-2017].

[29] S. García, M. Grill, J. Stiborek, and A. Zunino, “An empirical comparison of botnet detection methods,” *Comput. Secur.*, vol. 45, pp. 100–123, Sep. 2014.

[30] “Challenge: Evaluation - Causality Workbench.” [Online]. Available: http://www.causality.inf.ethz.ch/challenge.php?page=evaluation#cont. [Accessed: 04-Sep-2017].

[31] “El renacer de los exploit kits: Rig EK ya es el segundo malware más común.” [Online]. Available: http://www.silicon.es/renacer-exploit-kits-2335616?inf_by=59649a53681db8301d8b4aae. [Accessed: 04-Sep-2017].

[32] “Exploit kits, amados y odiados a partes iguales - S3lab.” [Online]. Available: http://s3lab.deusto.es/exploit-kits-amados-odiados/. [Accessed: 04-Sep-2017].

[33] E. Press, “El aumento de los ‘exploit kit’ se revela con Rig EK, segunda amenaza a nivel mundial en marzo.”

[34] “Bunitu, un troyano que infecta equipos para que actúen como proxys y hacer uso de VPNs.” [Online]. Available: https://www.redeszone.net/2015/08/06/bunitu-un-troyano-que-infecta-equipos-para-que-actuen-como-proxys-y-hacer-uso-de-vpns/.

[Accessed: 04-Sep-2017].

[35] "Malware-Traffic-Analysis.net - 2017-06-21 - Rig EK sends Bunitu Trojan." [Online]. Available: http://www.malware-traffic-analysis.net/2017/06/21/index.html. [Accessed: 04-Sep-2017].

[36] "Dreambot malware troyanos bancarios - detectarlo y eliminarlo - Cómo, Foro de Tecnología y Seguridad PC | SensorsTechForum.com." [Online]. Available: https://sensorstechforum.com/es/dreambot-banking-trojan-malware-detect-remove/. [Accessed: 04-Sep-2017].

[37] "Malware-Traffic-Analysis.net - 2017-06-19 - Rig EK from the HookAds campaign sends Dreambot." [Online]. Available: http://www.malware-traffic-analysis.net/2017/06/19/index.html. [Accessed: 04-Sep-2017].

[38] "El troyano Chthonic ataca más de 150 bancos de todo el mundo – Blog oficial de Kaspersky Lab." [Online]. Available: https://latam.kaspersky.com/blog/el-troyano-chthonic-ataca-mas-de-150-bancos-de-todo-el-mundo/4717/. [Accessed: 04-Sep-2017].

[39] "Malware-Traffic-Analysis.net - 2017-06-06 - RoughTed campaign Rig EK." [Online]. Available: http://www.malware-traffic-analysis.net/2017/06/06/index.html. [Accessed: 04-Sep-2017].

[40] "Así es el virus que vuelve a amenazar a todo el mundo | Tecnología Home | EL MUNDO." [Online]. Available: http://www.elmundo.es/grafico/tecnologia/2017/06/27/59527daf468aebc00d8b45d7.html. [Accessed: 04-Sep-2017].

[41] "Jaff, otro peligroso ransomware | unocero." [Online]. Available: https://www.unocero.com/noticias/jaff-peligroso-ransomware/. [Accessed: 04-Sep-2017].

[42] "Malware-Traffic-Analysis.net - 2017-06-13 - malspam pushing Jaff ransomware from .wsf files." [Online]. Available: http://www.malware-traffic-analysis.net/2017/06/13/index.html. [Accessed: 04-Sep-2017].

[43] "Malware-Traffic-Analysis.net - 2017-06-01 - malspam pushing Jaff ransomware from Word docs in PDF attachments." [Online]. Available: http://www.malware-traffic-analysis.net/2017/06/01/index2.html. [Accessed: 04-Sep-2017].

[44] "Wanna Decryptor: así funciona el ransomware que se ha usado en el ciberataque a Telefónica." [Online]. Available: https://www.xataka.com/seguridad/wanna-decryptor-asi-funciona-el-supuesto-ransomware-que-se-ha-usado-en-el-ciberataque-a-telefonica. [Accessed: 09-Sep-2017].

[45] "No Woman No Cry – Ransomware WannaCry – Follow The White Rabbit." [Online]. Available: https://www.fwhibbit.es/analizando-wannacry-ransomware-ms17-010. [Accessed: 13-Sep-2017].

[46] "Cowrie Honeypot: Ataques de fuerza bruta | Revista .Seguridad." [Online]. Available: https://revista.seguridad.unam.mx/numero-28/cowrie-honeypot. [Accessed: 09-Sep-2017].