



MÁSTER EN BIG DATA
ONLINE

THREAT HUNTING

Elaborado por: Carlos Federico Peña Peña
Tutor: Juan Manuel Moreno Lamparero

El Salvador, San Salvador – 16 de octubre del año 2024

A mi abuela que desde el cielo me sigue, mi madre que siempre es el motor de mis actividades, mis compañeros de trabajo que siempre compartieron su conocimiento, gracias por estar en esta etapa tan enriquecedora.

Que siempre arda el deseo de aprender y explorar el mundo.

ÍNDICE

RESUMEN.....	1
ABSTRACT	2
INTRODUCCIÓN	3
ANTECEDENTES.....	4
Ciberseguridad	4
Evolución de la ciberseguridad.....	5
Threat hunting.....	7
OBJETIVOS DEL PROYECTO.....	8
Objetivo general	8
Objetivos específicos.....	8
MATERIAL Y MÉTODOS	9
Infraestructura de la solución.....	9
Datasets.....	10
Anomalías en actividades de usuarios	10
Anomalías en actividades de equipos.....	11
Anomalías en actividades de tráfico	11
Estrategia para la extracción de los datasets	12
Machine learning.....	18
Deep learning	19
Bibliotecas y Herramientas.....	20
RESULTADOS	21
Análisis exploratorio de datos (EDA)	21
EDA Actividad de usuarios	21
EDA Actividad de equipos	22
EDA Actividad de tráfico.....	23
Entrenamiento	26
Isolation Forest.....	26
LSTM	32
Análisis de resultados.....	37
Análisis de actividades de usuarios.....	37
Análisis de actividades de equipos.....	41
Análisis de actividades de tráfico	43
Reporte de Power BI	44
CONCLUSIONES.....	53
REFERENCIAS BIBLIOGRÁFICAS.....	54

RESUMEN

En el presente trabajo se presenta el mundo del Threat Hunting o caza de amenazas en el ciberespacio, en un contexto donde los datos crecen constantemente de manera exponencial debido al aumento de dispositivos conectados y aplicaciones en la nube, convirtiendo la protección de la información en una necesidad urgente para todas entidades.

La propuesta se centra en las ventajas del Big Data, combinándolo con técnicas avanzadas de Machine Learning y Deep Learning. El objetivo ha sido detectar anomalías en tres frentes críticos: la actividad de usuarios, el comportamiento de los equipos y el tráfico de red. Y, además, complementar la solución con un proceso modular y adaptativo para un alcance muy alto de tipos de datos, incluyendo un reporte de resultados final.

Aprovechando el poder de Azure Sentinel y su intrínseca relación con herramientas como Microsoft Defender, se implementaron modelos no supervisados como Isolation Forest y redes LSTM (Long Short-Term Memory). Estos modelos que sirven muy bien la detección de anomalías en un conjunto de datos, con esa característica tan importante, son capaces de identificar comportamientos sospechosos que podrían pasar desapercibidos para los sistemas de seguridad convencionales., especialmente, para entornos seguros.

Los resultados que se obtuvieron son muy prometedores, el enfoque no solo identifica comportamientos anómalos, sino que además es bastante adaptativo en sus capacidades de análisis de distintos datos. Se acerca mucho al planteamiento de una herramienta de Threat Hunting completa y evidentemente, con margen de mejora y evolución.

Este trabajo no solo ofrece una solución esencial en la cacería de amenazas en el ciberespacio; también, demuestra cómo el Big Data puede convertirse en el mejor aliado de la ciberseguridad moderna. En un mundo donde los ciberataques evolucionan a una velocidad exponencial, el enfoque que se ha desarrollado proporciona un escudo adaptativo y robusto para las organizaciones del siglo XXI, especialmente cuando los costos de herramientas parecidas son altos, en este aspecto, podemos ser capaces de realizar análisis avanzados en un amplio rango de rubros.

ABSTRACT

In this work, we delve into the world of Threat Hunting in cyberspace, in a context where data is exponentially growing due to the increase in connected devices and cloud applications, making information protection an urgent need for all entities. The proposal focuses on the advantages of Big Data, combined with advanced Machine Learning and Deep Learning techniques. The goal has been to detect anomalies in three critical areas: user activity, equipment behavior, and network traffic. Additionally, the solution is complemented with a modular and adaptive process for a very high range of data types, including a final report of results.

Leveraging the power of Azure Sentinel and its intrinsic relationship with tools like Microsoft Defender, unsupervised models such as Isolation Forest and LSTM networks (Long Short-Term Memory) were implemented. These models, which are very effective in anomaly detection within a dataset, have the important characteristic of being able to identify suspicious behaviors that might go unnoticed by conventional security systems, especially in secure environments.

The results obtained are very promising; the approach not only identifies anomalous behaviors but is also quite adaptive in its analysis capabilities for different data. It closely approaches the concept of a complete Threat Hunting tool, with evident room for improvement and evolution.

This work not only offers an essential solution in threat hunting in cyberspace but also demonstrates how Big Data can become the best ally of modern cybersecurity. In a world where cyber-attacks evolve at an exponential rate, the developed approach provides an adaptive and robust shield for 21st-century organizations, especially when the costs of similar tools are high. In this respect, we can perform advanced analyses across a wide range of sectors.

INTRODUCCIÓN

El Threat Hunting o caza de amenazas, es una práctica proactiva de ciberseguridad que combina habilidades analíticas avanzadas con tecnologías de vanguardia para identificar y neutralizar amenazas antes de que causen daños significativos. En la era digital actual, donde la seguridad de la información es fundamental para organizaciones de todos los tamaños y sectores, el Threat Hunting emerge como una estrategia crucial para fortalecer la postura de seguridad frente a amenazas cada vez más sofisticadas.

La explosión de datos generados por sistemas interconectados, dispositivos IoT y aplicaciones en la nube ha creado un escenario donde las técnicas tradicionales de seguridad ya no son suficientes. Se estima que para 2025, el mundo almacenará 200 zettabytes de datos, gran parte de esta información siendo potencialmente sensible o crítica para las operaciones empresariales (Cybersecurity Ventures, 2022). Este volumen masivo de datos no solo representa un activo valioso para las organizaciones, sino también un desafío significativo en términos de seguridad y protección.

Los atacantes evolucionan constantemente, desarrollando métodos cada vez más sofisticados para evadir las defensas convencionales. Un estudio reciente reveló que el tiempo promedio para identificar y contener una brecha de datos es de 287 días, con un costo promedio de \$4.24 millones por incidente (IBM, 2023). Estas estadísticas alarmantes subrayan la urgente necesidad de estrategias más efectivas y proactivas en la defensa cibernética.

Lee y Lee definen el Threat Hunting como "el proceso de buscar proactivamente a través de redes y conjuntos de datos para detectar y aislar amenazas avanzadas que eluden las soluciones de seguridad existentes" (Lee R. M. y Lee R., 2018). Esta definición enfatiza la naturaleza proactiva del Threat Hunting, distinguiéndolo de los enfoques reactivos tradicionales. Como señala Bianco, "El Threat Hunting es tanto un arte como una ciencia, requiriendo creatividad e intuición junto con rigurosos métodos analíticos" (Bianco D., 2021).

Con el desarrollo de las problemáticas y necesidades actuales para el Threat Hunting, el Big Data compagina los intereses del Threat Hunting y se vuelve parte de las soluciones a las exploraciones de amenazas, teniendo grandes ventajas, como la elasticidad del tratamiento de datos masivos, técnicas de análisis de datos modernas y de vanguardia, arquitecturas de datos para unificar entornos y poder converger soluciones adaptativas para el Threat Hunting.

Por eso, es fundamental demostrar una solución que capture la esencia del Threat Hunting y el Big Data, que, además, pueda ser expresada como un lenguaje universal para empapar la importancia de la ciberseguridad en los tiempos contemporáneos. Este será el camino que seguiremos en este trabajo de fin de máster, en tres específicos comportamientos: "Anomalías en actividades de usuarios", "Anomalías en actividades de equipos" y "Anomalías en actividad de tráfico".

ANTECEDENTES

Ciberseguridad

La ciberseguridad moderna se enfrenta a desafíos sin precedentes debido a la complejidad y sofisticación de las amenazas actuales. El panorama de amenazas está en constante evolución, con actores maliciosos que desarrollan continuamente nuevas técnicas para evadir las defensas tradicionales.

En este contexto, el Threat Hunting se ha integrado estrechamente con las operaciones de los Centros de Operaciones de Seguridad (SOC) y los Sistemas de Gestión de Información y Eventos de Seguridad (SIEM).

La incorporación de técnicas de Machine Learning y Deep Learning ha llevado el Threat Hunting a un nuevo nivel de sofisticación, permitiendo la detección de amenazas más sutiles y complejas. Algunas aplicaciones específicas del Machine Learning y Deep Learning en Threat Hunting incluyen:

1. **Detección de Anomalías:** Identificación de patrones de comportamiento inusuales en sistemas y redes. Esto es especialmente útil en entornos donde las actividades son rutinarias, por lo tanto, podemos encontrar patrones en entornos “seguros”.
2. **Análisis de Comportamiento del Usuario y de la Entidad (UEBA):** Modelado del comportamiento normal de usuarios y entidades para detectar desviaciones.
3. **Clasificación de Malware:** Categorización automática de nuevas muestras de malware basada en características y comportamientos.
4. **Predicción de Vulnerabilidades:** Anticipación de posibles puntos débiles en sistemas y aplicaciones.
5. **Automatización de la Respuesta a Incidentes:** Implementación de respuestas automatizadas basadas en la detección de amenazas. Claro, implementar dichas respuestas, es rotundamente necesario hacer pruebas exhaustivas para reducir los falsos positivos.
6. **Análisis de secuencias temporales en el tráfico de red:** Monitoreo de anomalías en el tráfico de red, enfocado al análisis del tiempo de las actividades.

Sin embargo, al igual que las aplicaciones de las tecnologías de análisis de datos y big Data en entornos más comerciales, la implementación del Machine Learning y Deep Learning en el Threat Hunting no está exenta de desafíos, como la calidad de los datos, la interpretabilidad de los modelos y la necesidad de adaptabilidad frente a nuevas amenazas.

Evolución de la ciberseguridad

La ciberseguridad ha experimentado una transformación significativa en las últimas décadas, evolucionando desde simples medidas de protección hasta estrategias complejas y multifacéticas. Esta evolución puede dividirse en varias etapas clave:

1. Era de los Antivirus y Firewalls (1980s - 1990s)

Los primeros días de la ciberseguridad se centraron principalmente en la protección contra virus y el control de acceso básico.

- **Antivirus:** Enfocados en la detección y eliminación de software malicioso basado en firmas.
- **Firewalls:** Implementación de reglas de acceso para filtrar el tráfico de red entrante y saliente.

2. Surgimiento de los IDS/IPS (Late 1990s - 2000s)

Con el aumento de la complejidad de las redes y las amenazas, surgieron los Sistemas de Detección de Intrusiones (IDS) y los Sistemas de Prevención de Intrusiones (IPS) (Axelsson S., 2000).

- **IDS:** Monitoreo pasivo del tráfico de red para detectar actividades sospechosas.
- **IPS:** Evolución de los IDS con capacidad para bloquear activamente amenazas detectadas.

3. Era del Cumplimiento y Gestión de Riesgos (Mid 2000s - 2010s)

La introducción de regulaciones como SOX, HIPAA, y más tarde GDPR, llevó a un enfoque más formal en la gestión de riesgos y el cumplimiento normativo (ISACA, 2019).

- **Gestión de Riesgos:** Implementación de marcos como COBIT y NIST para evaluar y mitigar riesgos de seguridad.
- **Cumplimiento Normativo:** Adaptación de políticas y procedimientos para cumplir con regulaciones específicas de la industria.

4. Ascenso de la Seguridad Basada en Inteligencia (2010s - Presente)

El concepto de Threat Intelligence ganó terreno, marcando un cambio hacia un enfoque más proactivo en la ciberseguridad.

- **Threat Intelligence:** Recopilación y análisis de información sobre amenazas para mejorar la toma de decisiones en seguridad.
- **Seguridad Adaptativa:** Implementación de soluciones de seguridad que se ajustan dinámicamente a las amenazas emergentes.

Evolución de la Ciberseguridad

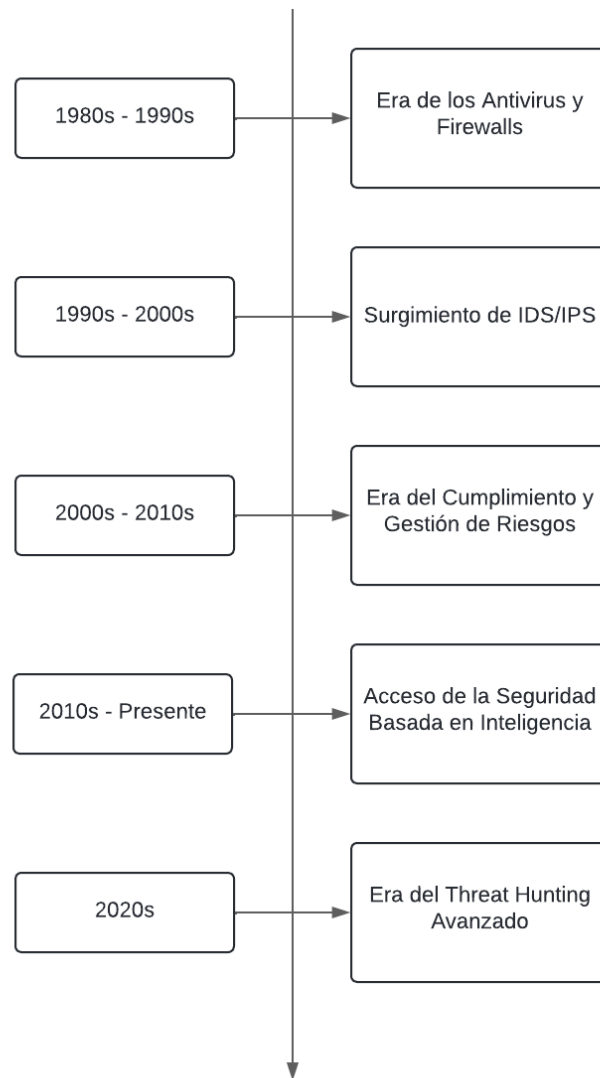


Figura 1. Línea del tiempo de la evolución de la ciberseguridad.

Threat hunting

El Threat Hunting emergió como una disciplina formal en respuesta a la creciente sofisticación de las amenazas cibernéticas y las limitaciones de los enfoques reactivos tradicionales. Aunque la práctica de buscar proactivamente amenazas ha existido en varias formas durante décadas, el término "Threat Hunting" comenzó a ganar tracción en la comunidad de seguridad a principios de la década de 2010.

Marcos y Metodologías de Threat Hunting:

- **The Hunting Loop** (Bianco D., 2021):
 - a. Crear una hipótesis
 - b. Investigar usando herramientas y técnicas
 - c. Identificar nuevos patrones y TTP
 - d. Informar los resultados
- **MITRE ATT&CK Framework:**
 - e. Matriz de tácticas y técnicas utilizadas por los adversarios
 - f. Base de conocimientos para planificar y ejecutar cazas de amenazas
- **The Threat Hunting Project** (Rodriguez J., 2022):
 - g. Enfoque colaborativo para compartir técnicas y procedimientos de caza
 - h. Desarrollo de playbooks y herramientas de código abierto

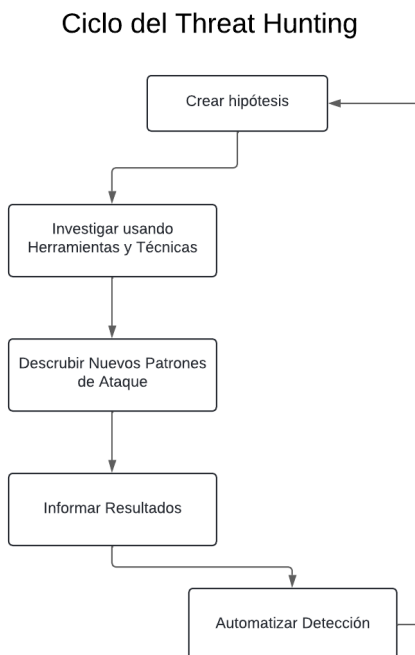


Figura 2. Ciclo del Threat Hunting.

OBJETIVOS DEL PROYECTO

Objetivo general

1. Implementar un Sistema de Threat Hunting, basado en las herramientas de big Data, técnicas de modelos no supervisados en Machine Learning y Deep Learning, explicado en un reporte visual e interactivo que permita el análisis detallado de las anomalías detectadas.

Objetivos específicos

1. Entrenar un modelo de Isolation Forest y LSTM enfocado en la detección de anomalías, explorando su comportamiento para llegar a un resultado acertado para la aplicación.
2. Crear un proceso modular y flexible capaz de adaptarse a diferentes aplicaciones y datos., implementando las capacidades del lenguaje de programación Python.

MATERIAL Y MÉTODOS

Infraestructura de la solución

Para la solución que se presentará para el Threat Hunting, de la mano de herramientas de big Data, obtienes los conocimientos de los siguientes módulos estudiados en el máster:

1. Curso de iniciación a Python
 - Lenguaje utilizado para todo el desarrollo del trabajo final de máster.
2. Fundamento de tratamiento de datos para Data Science
 - Aplicación de la manipulación de los datos, independientemente el formato en el que se presenten. Necesario para poder preparar los datasets para su posterior entrenamiento del modelo elegido.
3. Aprendizaje Automático Aplicado
 - Aplicación de las técnicas que nos ayudaran a determinar los patrones que posibles amenazas puedan estar siguiendo.
4. Inteligencia de Negocio y Visualización
 - Técnicas para presentar un reporte de seguridad visual y certero.
5. Valor y contexto de la analítica Big Data
 - Aplicación de tecnologías en la nube, en este caso, Azure y su aplicación en el campo de la ciberseguridad.

La arquitectura perseguida de la solución de Threat Hunting nos la muestra el diagrama de la **Figura 3** que nos servirá para las tres aplicaciones principales que desarrollaremos “Anomalías en actividades de usuarios”, “Anomalías en actividades de equipos” y “Anomalías en el tráfico de red”.

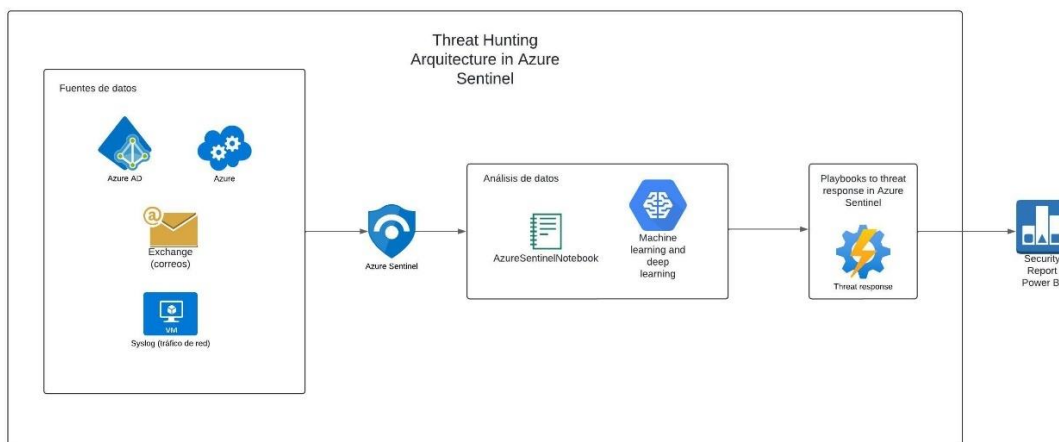


Figura 3. Arquitectura planteada para la solución de Threat Hunting, siendo el eje principal la solución SIEM y SOAR de Azure en la nube Azure Sentinel. Teniendo como fuente de datos principal un Log Analytics de Azure, donde podremos obtener datos de distintas fuentes conectadas al recurso de la nube, siguiendo al análisis de los datos elegidos para nuestros modelos de Machine Learning y Deep Learning, y, por último, la respuesta de dichas anomalías con playbooks de Azure (Logic Apps) de ser necesario según la aplicación de Threat Hunting elegida, todo esto, visualizado en un reporte de Power BI. Fuente: Desarrollo propio.

Datasets

Para la elección de los datasets se realizó un exhaustivo análisis de los datos disponibles, siendo este paso crucial en los casos planteado en este trabajo, que, empezamos entendiendo con el eje principal de la arquitectura, Azure Sentinel, que esta intrínsecamente conectado al Log Analytics, un recurso de Azure que recoge fuente de datos y permite su extracción por medio del lenguaje de consultas KQL (Kusto Query Language). Es importante destacar que este espacio de datos dependerá de los conectores que estará mandando datos al espacio de Log Analytics. Para cada una de las tres aplicaciones de este trabajo se siguió el siguiente proceso de estudio de datos:

1. Búsqueda de campos claves en todas las tablas disponibles en Log Analytics.
2. Elección de las tablas adecuadas para la búsqueda de anomalías.
3. Exploración de los campos disponibles que nos provee la tabla elegida.
4. Exploración del volumen de datos con posibles campos a elegir.
5. Elección de los campos, tomando en cuenta, valor de los campos para la aplicación elegida y volumen de los datos.

Como hemos mencionado, las aplicaciones serán “Anomalías en actividades de usuarios”, “Anomalías en actividades de equipos” y “Anomalías en actividades de tráfico”, por tanto, repasaremos el proceso de elección de datasets para cada una de las aplicaciones.

Anomalías en actividades de usuarios

Siguiendo la estrategia mencionada anteriormente, se llegó al resultado del **Query 1** (Ejecutado en la consola de Log Analytics de Azure Sentinel):

SecurityEvent

*| project TimeGenerated =
 format_datetime(datetime_utc_to_local(TimeGenerated,"America/El_Salvador"), "yyyy-MM-dd
 HH:mm:ss"), TargetUserName, Account, EventID = tostring(EventID), Activity, Process*

Query 1. Query para extraer las actividades de usuarios.

La tabla *SecurityEvent* contiene actividad de los usuarios desde los registros de seguridad de Windows, obteniendo activades como: inicios de sesión, modificación de recursos, procesos ejecutados, etc. Cada actividad es mapeada dentro del campo *EventID*, detallando la actividad con el campo *Activity*, además también brinda información valiosa del proceso en ejecución con el campo *Process* y por último, complementamos con el nombre de la cuenta que ejecuta la acción con *Account*, hacia donde se dirige con *TargetUserName* y el tiempo con *TimeGenerated* (Con formato de hora local).

Anomalías en actividades de equipos

Siguiendo la estrategia mencionada anteriormente, se llegó al resultado del **Query 2** (Ejecutado en la consola de Log Analytics de Azure Sentinel):

DeviceEvents

```
| project TimeGenerated =  
format_datetime(datetime_utc_to_local(TimeGenerated,"America/El_Salvador"), "yyyy-MM-dd  
HH:mm:ss"), DeviceName, InitiatingProcessFileName, ActionType, InitiatingProcessAccountName,  
InitiatingProcessCommandLine
```

Query 2. Query para extraer las actividades de equipos.

La tabla *DeviceEvent* contiene actividad de los equipos desde Microsoft Defender para Endpoint, obteniendo actividades como: modificación de la configuración de los dispositivos y telemetría de los equipos en general. Cada actividad es mapeada dentro del campo *ActionType*, nombre del equipo con *DeviceName*, nombre de la cuenta que inicio el proceso con *InitiatingProcessAccountName*, el archivo de proceso ejecutado con *InitiatingProcessFileName*, el comando del proceso ejecutado con *InitiatingProcessCommandLine* y el tiempo con *TimeGenerated* (Con formato de hora local).

Anomalías en actividades de tráfico

Siguiendo la estrategia mencionada anteriormente, se llegó al resultado del **Query 3** (Ejecutado en la consola de Log Analytics de Azure Sentinel):

CommonSecurityLog

```
| where DeviceProduct startswith "Firewall"  
| where Activity == "trafico aceptado"  
| parse AdditionalExtensions with * "ad.duration=" Duration ";" *  
| project TimeGenerated = format_datetime(datetime_utc_to_local(TimeGenerated,  
"America/El_Salvador"), "yyyy-MM-dd HH:mm:ss"), DeviceProduct, ApplicationProtocol,  
DeviceInboundInterface, ReceivedBytes = tostring(ReceivedBytes), SentBytes = tostring(SentBytes),  
Protocol = tostring(Protocol), Duration = tostring(Duration), SourcePort = tostring(SourcePort),  
DestinationPort = tostring(DestinationPort)
```

Query 3. Query para extraer las actividades de tráfico.

La tabla *CommonSecurityLog* contiene información conectada a un servidor Syslog, el entorno de trabajo tiene este servidor como puente a los firewalls, de esta forma se establece la conexión al LogAnalytics por medio del conector Common Event Format (CEF) en Azure Sentinel, en esta tabla podemos ver toda la actividad de tráfico, se han puesto dos filtros, el primero es para extraer solo información de los firewalls y el segundo es para extraer solamente el tráfico que ha sido aceptado, recordemos que el principal objetivo es detectar anomalías en el tráfico “seguro” que dejamos pasar a nuestra organización por medio de los firewalls. Extraemos los campos: *ApplicationProtocol*, *DeviceInboundInterface*, *ReceivedBytes*, *SentBytes*, *Protocol*, *Duration*, *SourcePort*, *DestinationPort*, estos campos son bien nativos del comportamiento del tráfico, por lo tanto, se llegó a la conclusión de incluirlo para darle más valor al posterior tratamiento de la información. Por último, el campo *DeviceProduct* que nos indica el firewall que manda el registro y el tiempo con *TimeGenerated* (Con formato de hora local).

Estrategia para la extracción de los datasets

Teniendo listo el análisis de los datasets y finalizando los campos que extraeremos para cada aplicación de Threat Hunting, ahora, nos falta el último paso, la estrategia para la extracción de estos datos a nuestro espacio de trabajo. Nos plantearemos las siguientes fases a seguir (correspondientes a nuestra arquitectura de trabajo) en la **Figura 4**. Es importante destacar que las extracciones de los datos dependerán del volumen de los mismos, es decir, no es lo mismo que en una hora de tiempo se tengan 100,000 registros que 1,000,000 de registros, por lo tanto, debe ser considerado en nuestra estrategia de obtención de datos.



Figura 4. Estrategia de extracción de datasets al entorno de trabajo. Fuente: Desarrollo propio.

Teniendo esta estrategia definida, en el **Script 1** la obtención del token para el api de Log Analytics.

Funcion para poder autenticarnos con los servicios de la fuente de datos del TFM, en este caso recursos de Azure

def token_recurso(recurso):

Client ID

client_id = "clientid"

Client Secret

client_secret = "clientsecret"

Tenant ID de los recursos a solicitar

tenant_id = "tenantid"

Direccion de autenticacion

authentication_request_url = f'https://login.microsoftonline.com/{tenant_id}/oauth2/token'

Tipo de flujo OAuth

grant_type = "client_credentials"

Recurso al que pediremos acceso

resource = recurso

Parametros para pedir el token de acceso

token_data = {

'grant_type': grant_type,

'client_id': client_id,

'client_secret': client_secret,

'resource': resource

}

Peticion para recibir el token de autorizacion

token_response = requests.post(authentication_request_url, data=token_data)

Extraccion del token

token_response_json = token_response.json()

```
access_token = token_response_json.get('access_token')
```

```
# Retornamos la informacion de la respuesta y el token obtenido
```

```
return access_token, token_response_json
```

Script 1. Script de Python para pedir autorización al recurso de Log Analytics. Con el parámetro “recurso” para determinar a que recurso queremos acceder.

En el **Script 2** hacemos la petición al recurso de Log Analytics, tomando en cuenta si se ha determinado un rango de tiempo para los datos.

```
# Funcion extraer los datos en un rango de tiempo establecido
```

```
def datos_de_queries(query, hours_to_end, hours_step, hours_before):
```

```
    # Conteo de las horas, si es cero empezamos desde la hora actual, si no, empezamos desfasados en el tiempo
```

```
    hours_count = hours_before
```

```
    hours_to_end = hours_to_end + hours_before
```

```
    # Creamos dataframe para enviar los datos obtenidos
```

```
    datos_finales_df = pd.DataFrame()
```

```
    # Pedimos autorizacion para acceder al recurso
```

```
    access_token, token_response_json = token_recurso("https://api.loganalytics.io")
```

```
    # Parametros para la url
```

```
    loganalytics_version = "v1"
```

```
    lognalaytics_workspace = "loganalyticsworkspace"
```

```
    # Creamos la url final para el espacio de Log Analytics
```

```
    loganalytics_request_url =
```

```
f"https://api.loganalytics.io/{loganalytics_version}/workspaces/{lognalaytics_workspace}/query"
```

```
    # Si existe el token de acceso, procedemos
```

```
    if access_token:
```

```
# Hasta obtener todos los datos, cerramos bucle
while (True):

    # Obtenemos la fecha ultima de los datos en formato
    end_date = (datetime.datetime.now() - datetime.timedelta(hours=hours_count))

    # Obtenemos la fecha primera de los datos
    start_date = (end_date - datetime.timedelta(hours=hours_step))

    # Damos formato adecuado para la api a las fechas
    start_date = start_date.strftime("%Y-%m-%dT%H:%M:%S.%fZ")
    end_date = end_date.strftime("%Y-%m-%dT%H:%M:%S.%fZ")

    # Preparamos el formato de los parametros de la api
    params_loganalytics = {"query": query,
                           "timespan": f"{start_date}/{end_date}"}

    # Preparamos el formato de los headers para la api
    headers = {"Authorization": f"Bearer {access_token}"}

    # Peticion para recibir los datos
    sentinel_response = requests.get(loganalytics_request_url, headers=headers,
    params=params_loganalytics).json()

    # Si hay respuesta, procedemos
    if sentinel_response:

        # Creamos formato de las columnas
        columns = {}

        # Extraemos el campo de tablas de la respuesta
        tabla_de_respuesta = sentinel_response["tables"]

        # Extraemos las columnas de la respuesta
        tabla_de_respuesta_columnas = tabla_de_respuesta[0]["columns"]
```

```

        # Extraemos las columnas
        for i in range(len(tabla_de_respuesta_columnas)):
            columnas[tabla_de_respuesta_columnas[i]["name"]] =
tabla_de_respuesta_columnas[i]["type"]

        # Extraemos los datos actuales de la petición en un formato de dataframe
        datos_actuales_df = pd.DataFrame(tabla_de_respuesta[0]["rows"],
columns=columnas.keys()).astype(columnas)

        # Agregamos los datos actuales a los datos finales
        datos_finales_df = pd.concat([datos_finales_df, datos_actuales_df], ignore_index=True)

    else: print("Error en la respuesta")

    # Salimos del bucle hasta terminar el rango de tiempo de los datos
    hours_count = hours_count + hours_step
    if hours_count >= hours_to_end:
        break
    else:
        print('Error al obtener el token:', token_response_json.get('error_description'))

    return datos_finales_df

```

Script 2. Script de Python para datos a Log Analytics en un rango tiempo especificado. Función que tiene de parámetros “query” para el Query que se ejecutara con la api de Log Analytics, “hours_to_end” para el límite máximo de tiempo, “hours_step” para el paso de horas entre las peticiones de los datos y “hours_before” para el desfase del tiempo, de ser necesario o requerido.

Siguiendo nuestro planteamiento inicial para la extracción de los datos, tenemos una función para extraer el token de autorización al api de Log Analytics y tenemos una función para extraer los datos en un rango de tiempo de un query especificado, en el **Script 3** podemos observar el ejemplo de funcionamiento de nuestra estrategia.

Ejemplo de Query que ocuparemos ejemplificar la estrategia de extracción de los datos

```

query_actividad_de_usuario = """
SecurityEvent
| project TimeGenerated =
format_datetime(datetime_utc_to_local(TimeGenerated,"America/El_Salvador"), "yyyy-MM-dd
HH:mm:ss"), TargetUserName, Account, EventID = tostring(EventID), Activity, Process
"""

```

Extraccion de los datos con el anterior Query especificado

```
data_actividad_de_usuarios = datos_de_queries(query_actividad_de_usuario, 720, 20, 0)
```

Script 3. Ejemplo de extracción de los datos con las funciones creadas, en un intervalo de tiempo de 720 horas, con un paso de 20 horas y sin ningún desfase de tiempo.

En resumen, la **Figura 5** nos muestra el proceso de la extraccion de los datos con las funciones anteriormente ejemplificadas. **Nota: Es importante importar los paquetes necesarios de Python para poder ejecutar los códigos: Request, Pandas y Datetime.**

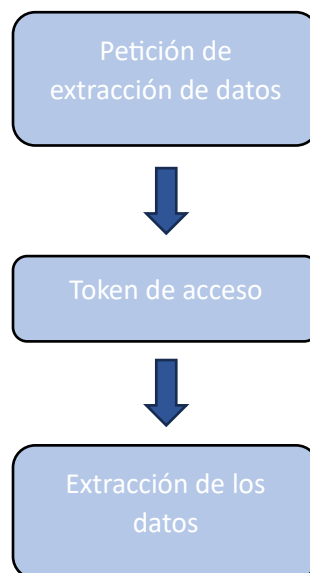


Figura 5. Extracción de datos desde Log Analytics. Fuente: Desarrollo propio.

Machine learning

Dentro de las técnicas de Machine Learning tenemos distintos escenarios posibles de aplicación, englobados en dos grandes grupos: Modelos supervisados y no supervisados. Para nuestro enfoque es especialmente importante este punto, debido al contexto de la aplicación, los datos o registros no vendrán con etiquetas de referencia a las anomalías que estamos persiguiendo, por tanto, un enfoque de entrenamiento no supervisado es particularmente idóneo para nuestros objetivos.

Isolation Forest es un algoritmo muy utilizado para la detección de anomalías, que tiene la particularidad de ser un algoritmo de aprendizaje no supervisado. En general, funciona creando un conjunto de arboles de decisión, que aíslan los puntos atípicos en los datos. El algoritmo selecciona al azar una característica y divide los datos en función de un valor aleatorio dentro del rango de la característica seleccionada, concluyendo en que las observaciones que requieren menos divisiones para ser aisladas son clasificadas como anomalías (Dhiraj K. & James S., 2024).

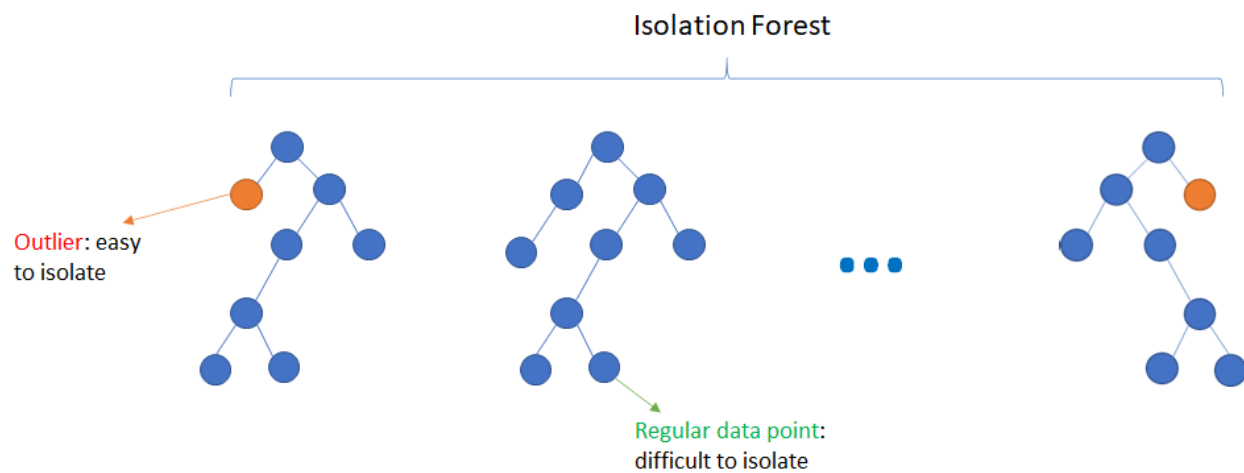


Figura 6. Esquema del Isolation Forest (Pallavi Pandey, 2022).

Deep learning

Las Redes Neuronales Recurrentes (RRN), específicamente la versión Long Short-Term Memory (LSTM), son modelos de Deep Learning que trabajan con datos secuenciales y temporales, por tanto, esto es ideal para la detección de anomalías en una serie de datos. Al igual que en nuestro modelo de Machine Learning, Isolation Forest, también con estas redes neuronales podemos trabajar con una estrategia no supervisada de datos.

Las RRN nos permiten identificar patrones en un largo plazo dentro de nuestros datos, este punto es crucial para los análisis de Threat Hunting en la ciberseguridad, específicamente para Amenazas Persistentes Avanzadas (APT), estos modelos son capaces de retener información durante un periodo de tiempo, este enfoque nos permite poder detectar comportamientos anómalos en los datos.

El modelo LSTM puede aprender correctamente las características que se eliminaron del conjunto de datos durante el entrenamiento. Esta capacidad permitió al modelo diferenciar con precisión el tráfico regular y los ataques a la red (Ruba Elhafiz, 2023).

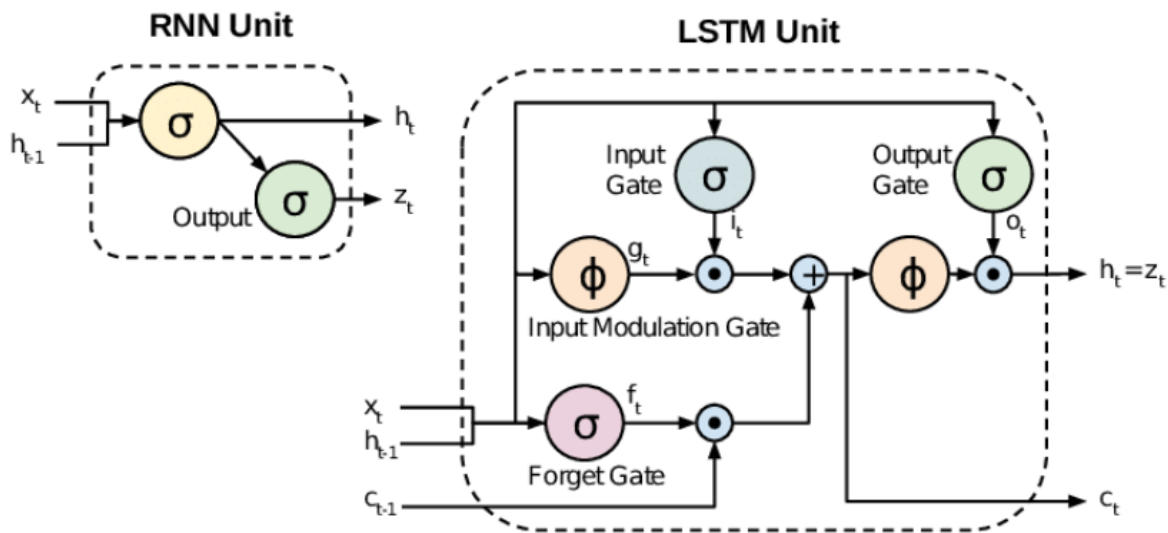


Figura 7. Comparación de RNN y LSTM (Ashutosh Tripathi, 2021).

Bibliotecas y Herramientas

1. Python

Lenguaje de programación de alto nivel, es un lenguaje muy versátil que destaca mucho en el ámbito del análisis de datos e inteligencia artificial. Por ende, es el elegido para poder montar nuestro trabajo.

2. Paquetes de Python

- a. **Requests:** Es una biblioteca para hacer solicitudes HTTP, además facilita la interacción con servicios web y APIs. Es la principal herramienta para poder extraer los datos desde Log Analytics.
- b. **Os:** Permite interactuar con el sistema operativo, lo usamos para poder extraer los modelos entrenados, guardar csv y la verificación de dichos archivos.
- c. **Pandas:** Biblioteca de análisis y manipulación de datos. Lo ocupamos principalmente por la estructura de los dataframes.
- d. **Datetime:** Permite manipular fechas y horas.
- e. **Seaborn:** Nos permite hacer visuales de alto nivel, está basada en matplotlib.
- f. **Matplotlib:** Biblioteca de trazado 2D, que nos permite crear gráficos.
- g. **Numpy:** Nos proporciona un gran soporte para la computación científica, especialmente para el procesamiento de datos y el análisis numérico.
- h. **Plotly:** Nos permite crear gráficos interactivos.
- i. **Joblib:** Biblioteca para la guardar y cargar modelos de aprendizaje automático.
- j. **Sklearn:** Biblioteca de aprendizaje automático que nos permite utilizar herramientas para el análisis de datos y la minería de datos.
- k. **Tensorflow:** Biblioteca para la inteligencia artificial, proporciona una plataforma flexible para construir y entrenar modelos de Machine Learning y Deep Learning, entre otras técnicas.

3. Azure

Es la nube de Microsoft. Aquí es donde el entorno de trabajo converge, debido a que la infraestructura esta al 99% en la nube.

4. Azure Sentinel

Es la solución de Gestion de Informacion y Eventos de Seguridad de Microsoft (SIEM) basado en la nube. Es el eje central de la aplicación trabajada en este proyecto.

5. Microsoft Defender

Es una suite de seguridad que nos brinda protección contra virus, amenazas en línea y monitoreo de identidad, esto lo utilizamos para extraer eventos de los equipos.

6. Syslog

Es un protocolo estándar para el registro de mensajes en sistemas informáticos, se utiliza para conectar la ingesta de logs de tráfico desde los firewalls.

RESULTADOS

Análisis exploratorio de datos (EDA)

Como primicia, los datos que extraeremos de Log Analytics están en formato string todos los campos, por tanto, el respectivo cambio de formato se decidió hacerlo para cada entrenamiento de cada aplicación, es decir, tendremos diferentes planteamientos en la preparación de los datos, para las actividades de usuario, equipos y tráfico. Además, también plantearemos la cantidad de datos para cada query en un día.

EDA Actividad de usuarios

Primero empezamos analizando el volumen de los datos, utilizando la consola de Log Analytics en Azure Sentinel, en la **Figura 8** podemos ver el resultado.

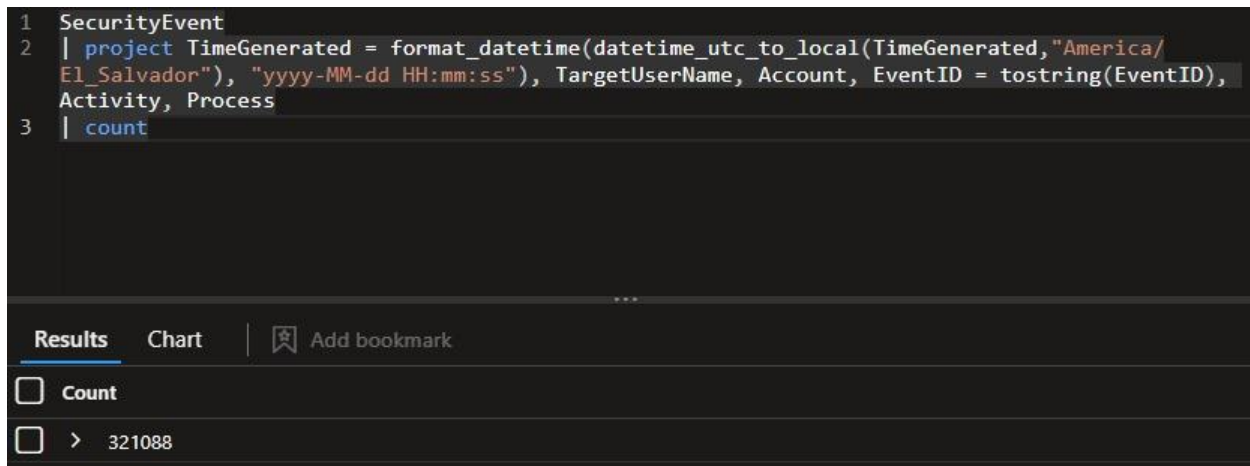


Figura 8. Volumen de datos en un día para la aplicación de actividad de usuarios.

Y terminamos con el peso de los datos, esto es muy importante, debido a que estamos ocupando API, y si se tiene restricción de Bytes por petición es muy importante tomarlo en cuenta, lo vemos en la **Figura 9**.

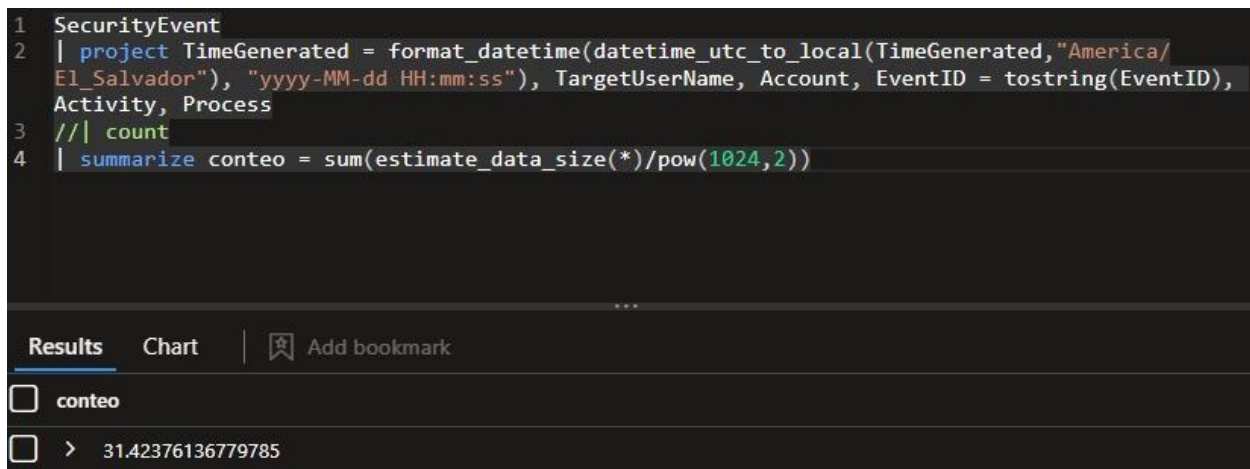


Figura 9. Volumen de datos en un día para la aplicación de actividad de usuarios.

Ahora, con estas primicias ya podemos iniciar con la evaluación de nuestro dataset, con un enfoque de variabilidad de cada campo, primero veremos el tipo de datos en la **Figura 10** y luego veremos la variabilidad de cada campo en la **Figura 11**.

```
data_actividad_de_usuarios = datos_de_queries(query_actividad_de_usuario, 60, 20, 0)
✓ 18.1s Python

print(data_actividad_de_usuarios.dtypes)
✓ 0.0s Python

TimeGenerated      string[python]
TargetUserName     string[python]
Account            string[python]
EventID            string[python]
Activity           string[python]
Process            string[python]
dtype: object
```

Figura 10. Tipos de datos para cada campo, podemos observar que, en los parámetros de la función, hemos puesto valores relacionados al análisis de cantidad y volumen que mencionamos anteriormente, por tanto, en este caso, podemos pedir datos de más de un día sin ningún problema de volumen, esto nos servirá para el posterior entrenamiento.

```
for i in data_actividad_de_usuarios.columns:
    frecuencias = data_actividad_de_usuarios[i].value_counts()
    desviacion_estandar = frecuencias.std()
    media = frecuencias.mean()
    coeficiente_de_variacion = (desviacion_estandar/media)
    print(f"Coficiente de variacion para {i} es: ", coeficiente_de_variacion)
✓ 0.3s Python

Coficiente de variacion para TimeGenerated es: 1.0468012262504642
Coficiente de variacion para TargetUserName es: 6.008661419485184
Coficiente de variacion para Account es: 6.836295920773074
Coficiente de variacion para EventID es: 4.3227730823718264
Coficiente de variacion para Activity es: 4.3227730823718264
Coficiente de variacion para Process es: 10.58390781407431
```

Figura 11. Variabilidad de cada campo, es mayor a 1 en cada campo, podemos interpretar que tenemos una variación alta.

EDA Actividad de equipos

Primero empezamos analizando el volumen de los datos, utilizando la consola de Log Analytics en Azure Sentinel, en la **Figura 12** podemos ver el resultado.

```
1 DeviceEvents
2 | project TimeGenerated = format_datetime(datetime_utc_to_local(TimeGenerated,"America/
  El_Salvador"), "yyyy-MM-dd HH:mm:ss"), DeviceName, InitiatingProcessFileName, ActionType,
  InitiatingProcessAccountName, InitiatingProcessCommandLine
3 | count
4 ||| summarize conteo = sum(estimate_data_size(*)/pow(1024,2))

...

Results Chart | Add bookmark

☐ Count
☐ > 1329128
```

Figura 12. Volumen de datos en un día para la aplicación de actividad de equipos.

Y terminamos con el peso de los datos, esto es muy importante, debido a que estamos ocupando API, y si se tiene restricción de Bytes por petición es muy importante tomarlo en cuenta, lo vemos en la **Figura 13**.

```

1 DeviceEvents
2 | project TimeGenerated = format_datetime(datetime_utc_to_local(TimeGenerated,"America/
  El_Salvador"), "yyyy-MM-dd HH:mm:ss"), DeviceName, InitiatingProcessFileName, ActionType,
  InitiatingProcessAccountName, InitiatingProcessCommandLine
3 //| count
4 | summarize conteo = sum(estimate_data_size(*)/pow(1024,2))

```

Results Chart | Add bookmark

☐ conteo

☐ > 393.76351261138916

Figura 13. Volumen de datos en un día para la aplicación de actividad de equipos.

Es muy interesante como en un día las cantidades aumentaron considerablemente, esto quiere decir que probablemente para el entrenamiento de esta aplicación necesitaremos abordar una estrategia diferente a la de usuarios, con estas primicias ya podemos iniciar con la evaluación de nuestro dataset, con un enfoque de variabilidad de cada campo, primero veremos el tipo de datos en la **Figura 10** y luego veremos la variabilidad de cada campo en la **Figura 11**.

```

data_actividad_de_equipos = datos_de_queries(query_actividad_de_equipos, 12, 2, 0)
✓ 1m 13.9s Python

print(data_actividad_de_equipos.dtypes)
✓ 0.0s Python

```

TimeGenerated	string[python]
DeviceName	string[python]
InitiatingProcessFileName	string[python]
ActionType	string[python]
InitiatingProcessAccountName	string[python]
InitiatingProcessCommandLine	string[python]
dtype:	object

Figura 14. Tipos de datos para cada campo, aquí la situación cambia con respecto a la aplicación de usuarios, debido a las cantidades muy altas, pedimos en menos tiempo los datos, por tanto, será más difícil el análisis de un rango de tiempo extendido, veremos este caso en particular posteriormente en el entrenamiento.

```

for i in data_actividad_de_equipos.columns:
    frecuencias = data_actividad_de_equipos[i].value_counts()
    desviacion_estandar = frecuencias.std()
    media = frecuencias.mean()
    coeficiente_de_variacion = (desviacion_estandar/media)
    print(f"Coeficiente de variacion para {i} es: ", coeficiente_de_variacion)
✓ 0.9s Python

```

Coeficiente de variacion para TimeGenerated es: 1.29874415668393
 Coeficiente de variacion para DeviceName es: 0.5376929490372007
 Coeficiente de variacion para InitiatingProcessFileName es: 6.940023894111443
 Coeficiente de variacion para ActionType es: 2.9079381280190413
 Coeficiente de variacion para InitiatingProcessAccountName es: 10.801649153152137
 Coeficiente de variacion para InitiatingProcessCommandLine es: 22.632903090320386

Figura 15. Variabilidad de cada campo, es mayor a 1 en todos los campos excepto en *DeviceName*, lo cual, es entendible debido a que posiblemente tengamos pocos firewalls, interpretamos además, una variación muy alta, mucho más alta en comparación de la aplicación de usuarios.

EDA Actividad de tráfico

Primero empezamos analizando el volumen de los datos, utilizando la consola de Log Analytics en Azure Sentinel, en la **Figura 16** podemos ver el resultado.

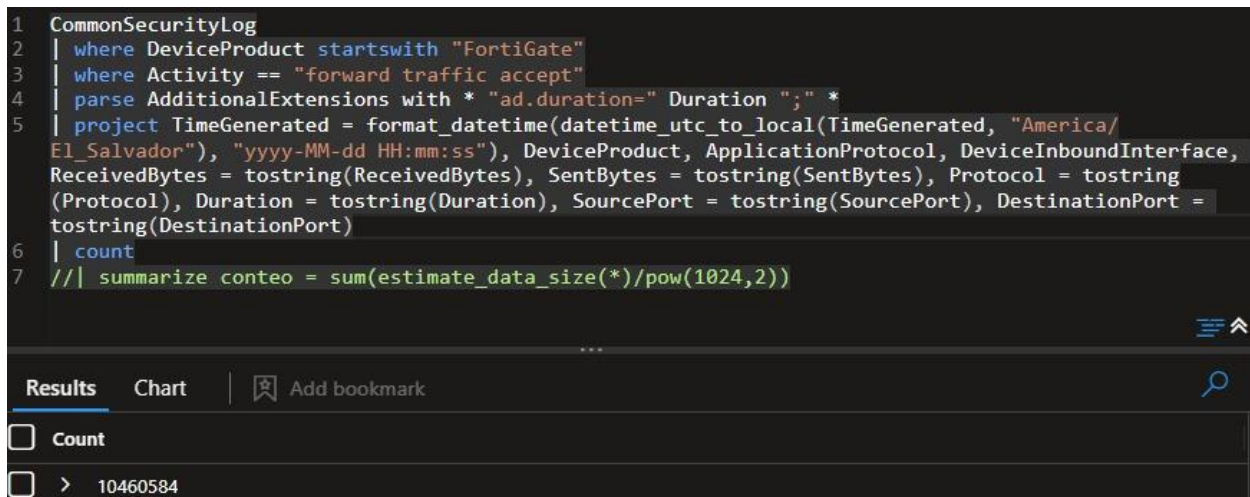


Figura 16. Volumen de datos en un día para la aplicación de actividad de tráfico.

Y terminamos con el peso de los datos, esto es muy importante, debido a que estamos ocupando API, y si se tiene restricción de Bytes por petición es muy importante tomarlo en cuenta, lo vemos en la Figura 17.

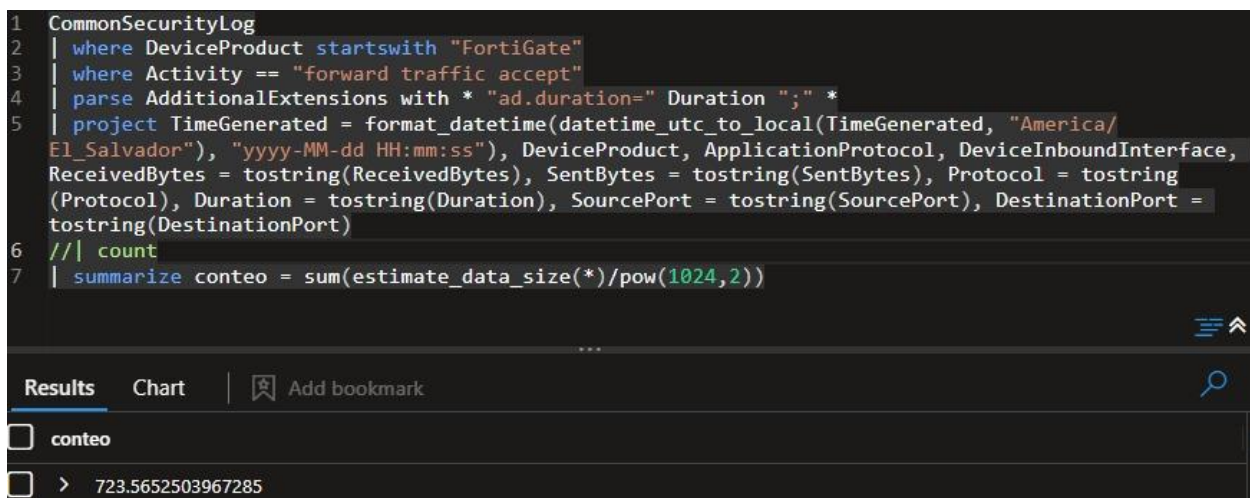


Figura 17. Volumen de datos en un día para la aplicación de actividad de tráfico.

Podemos observar que la cantidad de información para esta aplicación es con diferencia, mucho más grande que las anteriores, quiere decir que nuestro entrenamiento dependerá mucho del hardware, con cantidades de información así, esto será un limitante, con estas primicias ya podemos iniciar con la evaluación de nuestro dataset, con un enfoque de variabilidad de cada campo, primero veremos el tipo de datos en la Figura 18 y luego veremos la variabilidad de cada campo en la Figura 19.

```
data_actividad_de_trafico = datos_de_queries(query_actividad_de_trafico_aceptado, 2, 0.5, 1660)
✓ 1m 25.1s Python

print(data_actividad_de_equipos.dtypes)
✓ 0.0s Python

TimeGenerated          string[python]
DeviceName             string[python]
InitiatingProcessFileName string[python]
ActionType             string[python]
InitiatingProcessAccountName string[python]
InitiatingProcessCommandLine string[python]
dtype: object
```

Figura 18. Tipos de datos para cada campo, debido al gran volumen de datos, aquí solo podemos ir pidiendo datos cada media hora, por eso es muy importante el primer análisis de volumen y cantidad.

```
for i in data_actividad_de_trafico.columns:
    frecuencias = data_actividad_de_trafico[i].value_counts()
    desviacion_estandar = frecuencias.std()
    media = frecuencias.mean()
    coeficiente_de_variacion = (desviacion_estandar/media)
    print(f"Coeficiente de variacion para {i} es: ", coeficiente_de_variacion)
✓ 2.9s Python

Coeficiente de variacion para TimeGenerated es: 0.6226418103522979
Coeficiente de variacion para DeviceProduct es: 1.188843775119696
Coeficiente de variacion para ApplicationProtocol es: 23.843141744162825
Coeficiente de variacion para DeviceInboundInterface es: 2.311913355554881
Coeficiente de variacion para ReceivedBytes es: 82.39913493540344
Coeficiente de variacion para SentBytes es: 79.47893679097559
Coeficiente de variacion para Protocol es: 1.0499547033186243
Coeficiente de variacion para Duration es: 73.27567312782124
Coeficiente de variacion para SourcePort es: 8.95832677862956
Coeficiente de variacion para DestinationPort es: 33.90628197164668
```

Figura 19. Variabilidad de cada campo.

Entrenamiento

Como venimos mencionando, este trabajo pretende ser una solución completa de Threat Hunting, siendo moldeable, modular y adaptativa a las situaciones que puedan presentarse según la aplicación que se necesite. Por tanto, partimos de dos primicias: Entrenamientos con datos con volumen alto, variabilidad baja en sus campos y Entrenamiento con datos con volumen bajo y con variabilidad alta en sus campos, esto para Isolation Forest, para el entrenamiento de la red neuronal recurrente LSTM utilizaremos un solo camino de entrenamiento.

Isolation Forest

Se ha creado la función de entrenamiento *isolation_forest_function* que contiene las fases de preparación de datos, transformación de datos y entrenamiento de datos, podemos verlo en el **Script 3**, para la predicción de los datos también tenemos una función *predicción_anomalías_IF*, lo vemos en el **Script 4**.

```
# Funcion para el entrenamiento de un modelo de Isolation Forest

def isolation_forest_function(data_copy,train_mode,features,categorical_features,model_name):

    # Estos datos los extraemos para poder hacer un analisis de los resultados posteriormente

    # Convertimos a formato de fecha la columna correspondiente y posteriormente poder sacar los datos de tiempo necesarios

    data_copy["TimeGenerated"] = pd.to_datetime(data_copy["TimeGenerated"])

    # Extraemos la hora del registro de tiempo

    data_copy["hour"] = data_copy["TimeGenerated"].dt.hour

    # Extraemos el dia de la semana de tiempo, yendo del 0 al 6 correspondientes de lunes a domingo

    data_copy["day_of_week"] = data_copy["TimeGenerated"].dt.day_of_week

    # Extraemos el dia de la semana de tiempo, yendo del 0 al 6 correspondientes de lunes a domingo

    data_copy["date"] = data_copy["TimeGenerated"].dt.date


    # Entrenamos segun el modo establecido en los parametros

    # Para el modo de entrenamiento en volumen altos

    if train_mode == "big_data":


        # Hacemos el tratamiento de las variables

        for clave, valor in categorical_features.items():


            # Se factoriza la variable tipo ID
```

```
if clave == "id_feature":

    data_copy[valor[1]] = pd.factorize(data_copy[valor[0]])[0]
    break

# Se crea una columna con el numero de ocurrencias de los valores de cada campo
data_copy[valor] = data_copy.groupby(clave)[clave].transform("count")

# Se verifica si existe ya el modelo entrenado
if os.path.isfile(f"models/{model_name}.joblib"):

    # Cargamos el modelo entrenado
    model = load(f"models/{model_name}.joblib")

    # Llamamos la funcion de prediccion de anomalias para Isolation Forest
    anomalias, no_anomalias = prediccion_anomalias_IF(data_copy, model, features)

else:

    # Entrenamos modelo si no existe en el directorio
    # Entrenamos con un porcentaje de anomalias de 1%
    model = IsolationForest(contamination=0.01)
    model.fit(data_copy[features])

    # Hacemos la prediccion en nuestros datos
    anomalias, no_anomalias = prediccion_anomalias_IF(data_copy, model, features)

    # Guardamos el modelo
    dump(model, f"models/{model_name}.joblib")

# Para el modo de entrenamiento en volúmenes bajos
elif train_mode == "small_data":
```

```
# Hacemos una copia de los datos originales para poder visualizar todas las características
# con los resultados
data_copy_original = data_copy.copy()
# Sacamos los dummies de los campos
# Es posible de esta forma por ser un volumen bajo de datos
# Hay poca variabilidad y ademas, es algo que el hardware permite procesar
data_copy = pd.get_dummies(data_copy, columns=categorical_features)

# Preparamos un arreglo para las columnas de los dummies
columns_dummies = []

# Extraemos solo las columnas creadas de los dummies
for col in data_copy.columns:

    for feature in features:

        if feature in col:
            columns_dummies.append(col)

# Extraemos las características a entrenar
features = ["hour", "day_of_week"] + columns_dummies

# Entrenamos modelo si no existe en el directorio
# Entrenamos con un porcentaje de anomalías de 1%
model = IsolationForest(contamination=0.01)
model.fit(data_copy[features])

# Se verifica si existe ya el modelo entrenado
if os.path.isfile(f"models/{model_name}.joblib"):

    # Cargamos el modelo entrenado
```



```
model = load(f"models/{model_name}.joblib")

# Llamamos la funcion de prediccion de anomalias para Isolation Forest
anomalias, no_anomalias = prediccion_anomalias_IF(data_copy, model, features)

else:

    # Entrenamos modelo si no existe en el directorio
    # Entrenamos con un porcentaje de anomalias de 1%
    model = IsolationForest(contamination=0.01)
    model.fit(data_copy[features])

    # Hacemos la prediccion en nuestros datos
    anomalias, no_anomalias = prediccion_anomalias_IF(data_copy, model, features)\
    # Guardamos el modelo
    dump(model, f"models/{model_name}.joblib")

    # Extraemos las columnas que no estan en el dataset de entrenamiento
    columnas_faltantes_copia = [col for col in data_copy_original.columns if col not in
data_copy.columns]
    # Agregamos las columnas y los datos al dataset de entrenamiento
    data_copy = pd.concat([data_copy, data_copy_original[columnas_faltantes_copia]], axis=1)

    # Creamos diccionario para guardar metricas de evaluacion del modelo
    metricas_evaluacion_modelo = {}

    # Guardamos el porcentaje de las anomalias
    metricas_evaluacion_modelo["porcentaje_de_anomalias"] =
(len(anomalias)/(len(anomalias)+len(no_anomalias)))*100

    # Guardamos el MSE de datos anomalos y no anomalos
    metricas_evaluacion_modelo["mse_datos_no_anomalos"] =
mean_squared_error(no_anomalias["anomia_puntuacion"], np.zeros(len(no_anomalias)))
```

```

    metricas_evaluacion_modelo["mse_datos_anomalos"] =
mean_squared_error(anomalias["anomia_puntuacion"], np.ones(len(anomalias)))

    # Retornamos el dataset de entrenamiento, el grupo de anomalías, no anomalias y las metricas de
    # evaluacion del modelo

    return data_copy, anomalias, no_anomalias, metricas_evaluacion_modelo

```

Script 3. Script de Python para realizar el entrenamiento de un modelo de Isolation Forest. Con los parámetros *data_copy* para pasar el dataframe de datos a entrenar o predecir, *train_mode* para indicar el tipo de volumen de datos que se entrenaran, *features* un arreglo de los campos que se entrenaran, *categorical_features* un diccionario de los campos en su nomenclatura convertida (para datos de volumen bajo o alto) y *model_name* que indicara el nombre del modelo que se utilizara.

```

# Funciones para la prediccion de las anomalias con el modelo de Isolation Forest
def prediccion_anomalias_IF(data_copy, model, features):

    # Creamos columna de anomalia y guardamos las predicciones
    data_copy["anomalia"] = model.predict(data_copy[features])
    # Convertimos a 0 y 1 los resultados obtenidos, 0 para datos normales y 1 para anomalias
    data_copy["anomalia"] = data_copy["anomalia"].map({1:0,-1:1})

    # Extremos la puntuacion de la anomalia
    data_copy["anomalia_puntuacion"] = model.decision_function(data_copy[features])

    # Guardamos un grupo de datos anomalos
    anomalias = data_copy[data_copy["anomalia"]==1]
    # Guardamos un grupo de datos no anomalos
    no_anomalias = data_copy[data_copy["anomalia"]==0]

    # Retornamos el grupo de anomalias y no anomalias
    return anomalias, no_anomalias

```

Script 4. Script de Python para realizar las predicciones. Con los parámetros, *data_copy* siendo el dataset que se ocupara para las predicciones, *model* es el modelo para realizar las predicciones y *features* para las predicciones.

La arquitectura resumida de nuestra función de entrenamiento es la siguiente:

1. Conversión columna de *TimeGenerated* a formato de fecha, extrayendo *hour*, *day_of_week* y *date*.
2. Entrenamiento del modelo según *train_mode*:
 - a. Big Data:
 - i. Tratamiento de variables categóricas, factorización de *id_feature* y contar las ocurrencias de valores en cada campo.
 - ii. Verificación si el modelo ya existe
 1. Si: Cargar el modelo y hacer la predicción de los datos.
 2. No: Entrenar el modelo, predecir las anomalías y guardamos el modelo entrenado.
 - b. Small Data
 - i. Creación de dummies para variables categóricas.
 - ii. Extraer las columnas de los dummies.
 - iii. Verificación si el modelo ya existe
 1. Si: Cargar modelo y predecir anomalías
 2. No: Entrenar el modelo, predecir las anomalías y guardamos el modelo entrenado.
 - iv. Creación final de dataset de salida con todas las columnas.
3. Guardamos las métricas de evaluación.
4. Retornamos los resultados.

La arquitectura resumida de nuestra función de predicción es la siguiente:

1. Predicción de los datos con *predict*.
2. Mapeamos anomalías y no anomalías.
3. Creamos grupo de anomalías y no anomalías.
4. Retornamos los resultados.

LSTM

Se ha creado la función de entrenamiento *isolation_forest_function* que contiene las fases de preparación de datos, transformación de datos y entrenamiento de datos, podemos verlo en el **Script 5**, para la predicción de los datos también tenemos una función *prediccion_anomalias_TS*, lo vemos en el **Script 6**.

Funcion para el entrenamiento de una red neuronal recurrente LSTM

def temporal_sequential_function(data_copy, features, model_name):

funcion para crear la secuencia de logs que procederemos a entrenar

def secuencias_de_logs(datos, tamaño_secuencia):

Creamos arreglo para guardar las secuencias

secuencias = []

Creamos la secuencia segun el tamaño de secuencia previamente definido

for i in range(len(datos) - tamaño_secuencia):

secuencia_actual = datos[i:i+tamaño_secuencia]

secuencias.append(secuencia_actual)

return np.array(secuencias)

Convertimos a formato de fecha la columna correspondiente y posteriormente poder sacar los datos de tiempo necesarios

data_copy["TimeGenerated"] = pd.to_datetime(data_copy["TimeGenerated"])

Extraemos la hora del registro de tiempo

data_copy["hour"] = data_copy["TimeGenerated"].dt.hour

Extraemos el día de la semana de tiempo, yendo del 0 al 6 correspondientes de lunes a domingo

data_copy["day_of_week"] = data_copy["TimeGenerated"].dt.day_of_week

Extraemos el día de la semana de tiempo, yendo del 0 al 6 correspondientes de lunes a domingo

data_copy["date"] = data_copy["TimeGenerated"].dt.date

Iniciamos el scaler para tener nuestras características en 0 y 1

scaler = MinMaxScaler()

```
# Hacemos tratamiento a nuestras características
for clave, valor in features.items():

    # Para las variables categoricas
    if clave == "categorical":

        for i in range(len(valor)):

            # Convertimos a valores numericos
            data_copy[valor[i]] = LabelEncoder().fit_transform(data_copy[valor[i]])

    elif clave == "numeric":

        # Ocupamos el scaler para hacerl los valores entre 0 y 1
        data_copy[valor] = scaler.fit_transform(data_copy[valor])

# Verificamos si modelo ya existe
if os.path.isfile(f"models/{model_name}.keras"):

    # Definimos tamaño de la secuencia
    tamaño_secuencia = 10

    # Creamos las secuencias de datos para poder ocupar el datasets con el modelo ya entrenado
    x = secuencias_de_logs(data_copy[features["categorical"] + features["numeric"]],
                           tamaño_secuencia)

    # Cargamos el modelo
    model = load_model(f"models/{model_name}.keras")

    # Extraemos los datos relevantes de las predicciones, incluyendo error de reconstrucción del modelo
    y el umbral de anomalías
    anomalías, no_anomalías, secuencias_anomalías, secuencias_normales, error_reconstrucción,
    umbral_anomalías = prediccion_anomalías_TS(x, model)
```

else:

Definimos tamaño de la secuencia

tamaño_secuencia = 10

Creamos las secuencias de datos para poder ocupar el dataset y entrenar nuestro modelo

*x = secuencias_de_logs(data_copy[features["categorical"] + features["numeric"]],
tamaño_secuencia)*

Construimos el modelo secuencial LSTM

model = Sequential()

Añadimos capa de 64 unidades LSTM

model.add(LSTM(64, input_shape=(x.shape[1],x.shape[2]), return_sequences=False))

Repetimos el vector de entrada

model.add(RepeatVector(x.shape[1]))

Añadimos otra capa de 64 unidades LSTM

model.add(LSTM(64, return_sequences=True))

Aplicamos una capa densa a cada uno de los pasos de tiempo

model.add(TimeDistributed(Dense(x.shape[2])))

Compilamos el modelo

model.compile(optimizer="adam", loss="mse")

Entrenamos el modelo

model.fit(x, x, epochs=10, batch_size=64)

Modificar para el siguiente entrenamiento a .keras

model.save(f"models/{model_name}.keras")

*# Extraemos los datos relevantes de las predicciones, incluyendo error de reconstrucción del modelo
y el umbral de anomalías*

```

    anomalias, no_anomalias, secuencias_anomalas, secuencias_normales, error_reconstruccion,
    umbral_anomalias = prediccion_anomalias_TS(x, model)

```

```

# Retornamos los valores

```

```

    return data_copy, anomalias, no_anomalias, secuencias_anomalas, secuencias_normales,
    error_reconstruccion, tamaño_secuencia, umbral_anomalias

```

Script 5. Script de Python para realizar el entrenamiento de un modelo de LSTM. Con los parámetros *data_copy* para pasar el dataframe de datos a entrenar o predecir, *features* un arreglo de los campos que se entrenaran y *model_name* que indicara el nombre del modelo que se utilizara.

```

# Funciones para la prediccion de las anomalias con el modelo de LSTM

```

```

def prediccion_anomalias_TS(x, model):

```

```

    # hacemos las predicciones

```

```

    predicciones_actividad_trafico = model.predict(x)

```

```

    # Calculamos el error de reconstruccion

```

```

    error_reconstruccion = np.mean(np.square(x - predicciones_actividad_trafico), axis=(1,2))

```

```

    # Calculamos el umbral de anomalias

```

```

    umbral_anomalias = np.mean(error_reconstruccion) + 2 * np.std(error_reconstruccion)

```

```

    # Calculamos las secuencias de datos normales y anomalias

```

```

    secuencias_anomalas = np.where(error_reconstruccion > umbral_anomalias)[0]

```

```

    secuencias_normales = np.where(error_reconstruccion <= umbral_anomalias)[0]

```

```

    # Extraemos el grupo de anomalias y no anomalias

```

```

    anomalias = x[secuencias_anomalas]

```

```

    no_anomalias = x[secuencias_normales]

```

```

    # Retornamos los resultados

```

return anomalias, no_anomalias, secuencias_anomalias, secuencias_normales, error_reconstruccion, umbral_anomalias

Script 6. Script de Python para realizar las predicciones. Con los parámetros, *x* siendo el dataset (Secuencia de los datos creada previamente) que se ocupara para las predicciones y *model* es el modelo para realizar las predicciones.

La arquitectura resumida de nuestra función de entrenamiento es la siguiente:

1. Creamos función de creación de las secuencias de entrenamiento para nuestro modelo.
2. Conversión columna de *TimeGenerated* a formato de fecha, extrayendo *hour*, *day_of_week* y *date*.
3. Iniciamos el *Scaler* para poder hacer conversiones previas de nuestras variables numéricas.
4. Tratamiento de variables
 - a. Categóricas: Convertir a valores numéricos con *LabelEncoder*.
 - b. Numéricas: Aplicamos el *Scaler*.
5. Verificación si el modelo ya existe
 - a. Si: Definir la secuencia de los datos, cargar el modelo y hacer la predicción de los datos.
 - b. No: Definir la secuencia de los datos, construir el modelo LSTM, entrenar el modelo, predecir las anomalías y guardamos el modelo entrenado.
6. Retornamos los resultados.

La arquitectura resumida de nuestra función de predicción es la siguiente:

5. Predicción de los datos con *predict*.
6. Mapeamos anomalías y no anomalías.
7. Creamos grupo de anomalías y no anomalías.
8. Retornamos los resultados.

Análisis de resultados

Es muy importante tomar en cuenta que los resultados y las evaluaciones las tenemos que hacer para datos no supervisados, además, ya hemos planteado una problemática anteriormente, sobre la variabilidad excesiva los campos de un dataset, por tanto, en el análisis de actividades de usuario, veremos dos tipos de análisis, un análisis con datos de volumen mayor y un análisis con datos de volumen alto, esto, para el modelo de Isolation Forest, además es importante también aclarar que este mismo enfoque para la actividad de equipos no es posible con el actual entorno de trabajo, debido a que el hardware limite usar datasets muy variables, por ende, no se ha podido realizar el análisis con dummies en datos de volumen bajo para la actividad de equipos.

Análisis de actividades de usuarios

En el **Script 7** veremos la implementación para volumen alto para la actividad de usuarios.

```
-----  
# Caracteristicas categoricas para el entrenamiento, ademas de la caracteristica tipo ID  
categorical_features = {  
    "EventID": "EventID_frecuencia",  
    "Process": "Process_frecuencia",  
    "id_feature": ["Account", "Account_num"]  
}  
  
# Caracteristicas de entrenamiento iniciales  
features = ["Account_num", "Process_frecuencia", "EventID_frecuencia", "hour", "day_of_week"]  
  
# Hacemos una copia de los datos originales  
data_actividad_de_usuarios_copy = data_actividad_de_usuarios.copy()  
  
# Entrenamiento del modelo, prediccion de los datos y extraccion de los resultados y evaluaciones  
data_actividad_de_usuarios_copy, anomalias_actividad_de_usuarios,  
no_anomalias_actividad_de_usuarios, metricas_del_modelo =  
isolation_forest_function(data_actividad_de_usuarios_copy, "big_data", features, categorical_features,  
"usuarios_bigData_IF")  
-----
```

Script 7. Script para realizar el entrenamiento del modelo, las predicciones de los datos y la extracción de los resultados, esto, para un análisis de volumen alto.

Evaluación del modelo, en la **Figura 20** podemos observar las métricas del porcentaje de anomalías en los datos, el MSE para datos anómalos y no anómalos; además, de un gráfico de la distribución de las puntuaciones de anomalías en la **Figura 21** y un gráfico de mapa de calor de las anomalías en la **Figura 22**.

```
print(metricas_del_modelo)
✓ 0.0s Open 'metricas_del_modelo' in Data Wrangler
Python
{'porcentaje_de_anomalias': 1.5717892722879279, 'mse_datos_no_anomalous': 0.022031044334469434, 'mse_datos_anomalous': 1.0333388665403915}
```

Figura 20. Podemos extraer el valor de 1.57% de anomalías en todo el dataset, siendo un valor muy acertado debido al parámetro de 1% que hemos puesto en el entrenamiento, además el MSE para valores no anómalos nos deja muy claro que los datos normales son más fáciles de predecir debido a su valor muy cercano a 0, pasa lo contrario con el MSE de los valores anómalos, debido a que son más difíciles de separar, su valor es mas alto.

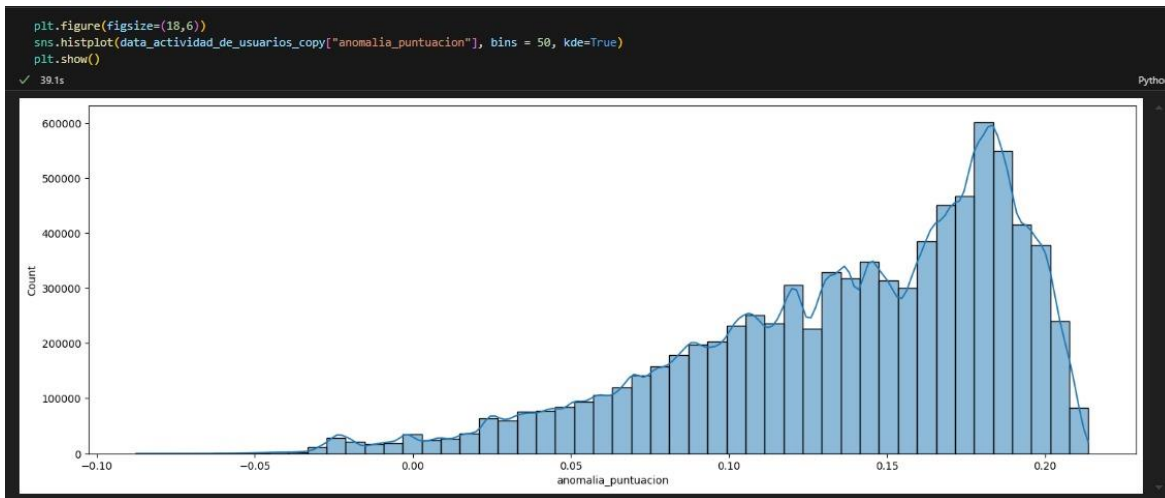


Figura 21. Distribución muy clara de los datos anómalos y no anómalos, siendo complemento del anterior análisis.

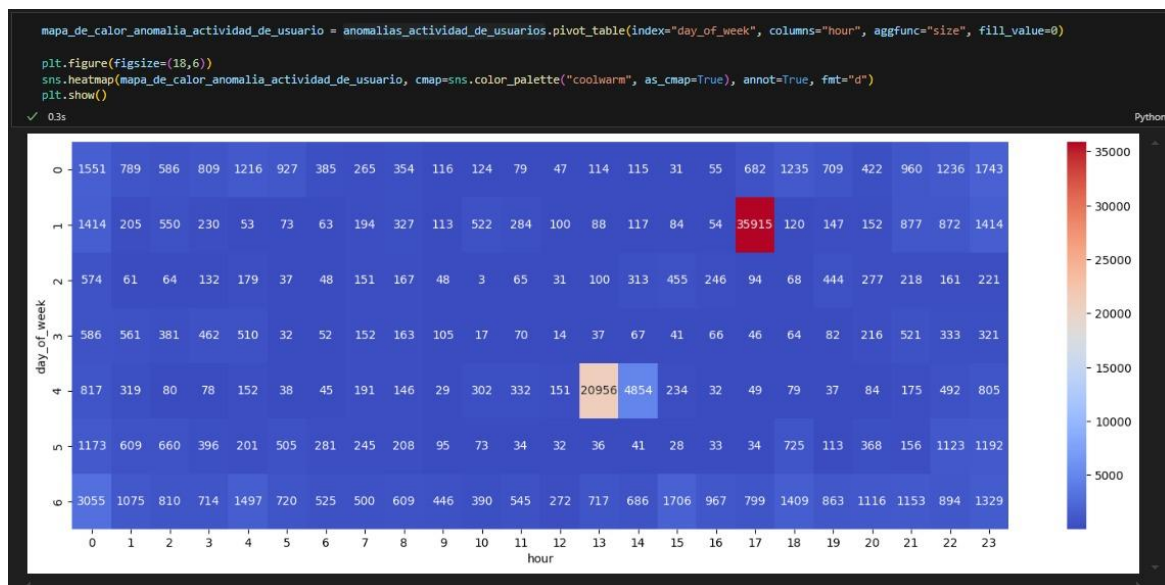


Figura 22. Aquí podemos observar que las anomalías se concentran mucho en puntos bien específicos de los días de la semana y las horas, veremos más adelante el comportamiento del modelo con volumen bajo de datos.

En el **Script 8** veremos la implementación para volumen bajo para la actividad de usuarios.

Variables categoricas

categorical_features = ["EventID", "Process", "Account"]

Variables para entrenamiento, posteriormente las ocuparemos con dummies

features = ["EventID_", "Process_", "Account_"]

Hacemos una copia de los datos originales


data_actividad_de_usuarios_recortado_copy = data_actividad_de_usuarios.copy()

Entrenamiento del modelo, prediccion de los datos y extraccion de los resultados y evaluaciones

data_actividad_de_usuarios_recortado_copy, anomalias_actividad_de_usuarios_recortado, no_anomalias_actividad_de_usuarios_recortado, metricas_del_modelo = isolation_forest_function(data_actividad_de_usuarios_recortado_copy, "small_data", features, categorical_features, "usuarios_smallData_IF")

Script 8. Script para realizar el entrenamiento del modelo, las predicciones de los datos y la extracción de los resultados, esto, para un análisis de volumen bajo.

Evaluación del modelo, en la **Figura 23** podemos observar las métricas del porcentaje de anomalías en los datos, el MSE para datos anómalos y no anómalos; además, de un gráfico de la distribución de las puntuaciones de anomalías en la **Figura 24** y un gráfico de mapa de calor de las anomalías en la **Figura 25**.



```
print(metricas_del_modelo)
```

✓ 0.0s Open 'metricas_del_modelo' in Data Wrangler Python

```
{'porcentaje_de_anomalias': 0.965387554862277, 'mse_datos_no_anomalos': 0.0009881431534365882, 'mse_datos_anomalos': 1.0064095229812222}
```

Figura 23. Podemos extraer el valor de 0.96% de anomalías en todo el dataset, siendo un valor muy acertado debido al parámetro de 1% que hemos puesto en el entrenamiento, además el MSE para valores no anómalos nos deja muy claro que los datos normales son más fáciles de predecir debido a su valor mucho más cercano a 0 que el anterior modelo, pasa lo contrario con el MSE de los valores anómalos, debido a que son más difíciles de separar, su valor es más alto, resultado muy parecido al anterior modelo.

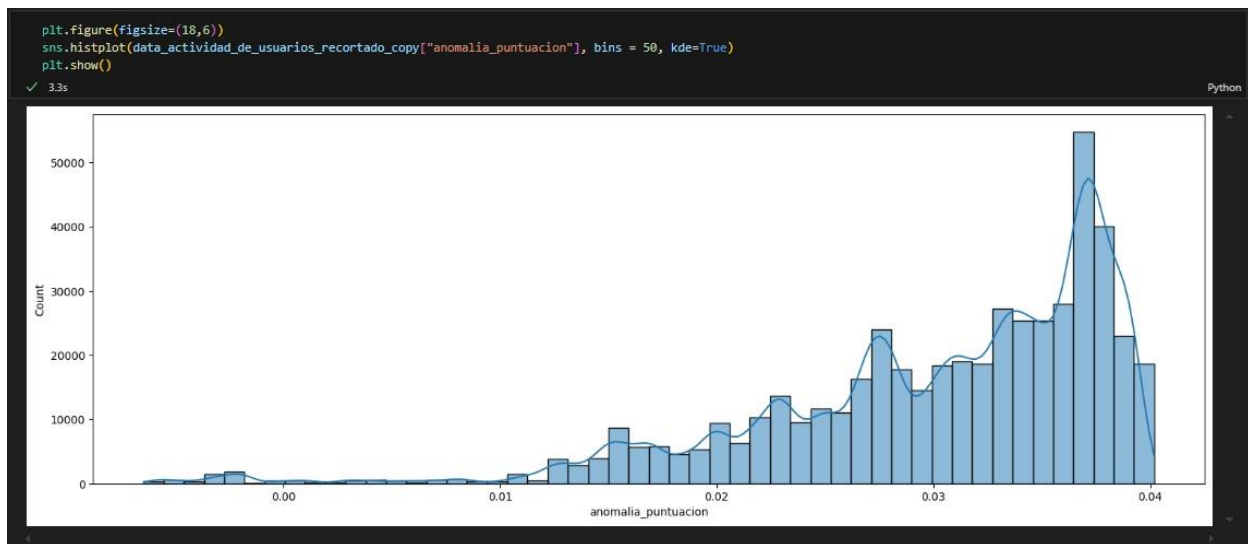


Figura 24. Distribución muy clara de los datos anómalos y no anómalos, siendo complemento del anterior análisis.

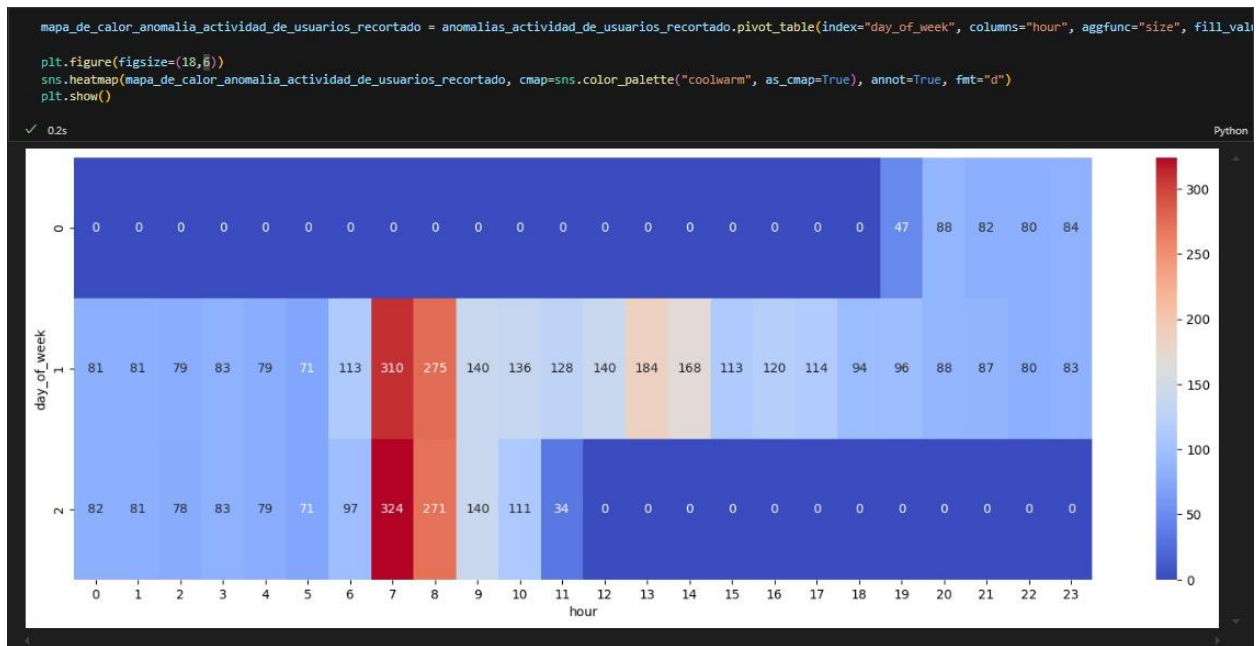


Figura 25. Vemos las anomalías más dispersas que el anterior modelo, esto puede ser consecuencia de cómo estamos tratando la conversión de las variables categorías, en este caso utilizamos dummies, por lo tanto, cada variación de los campos tiene relevancia en el entrenamiento, en cambio, cuando hacemos un conteo de las variables por campo como en el anterior modelo, es posible que datos variaciones poco frecuentes no sean tomadas en cuenta, por tanto, puede estar ocurriendo un sesgo de datos.

Análisis de actividades de equipos

En el **Script 9** veremos la implementación para volumen alto para la actividad de equipos.

```
# Características categoricas para el entrenamiento, ademas de la característica tipo ID
categorical_features = {
    "InitiatingProcessFileName" : "InitiatingProcessFileName_frecuencia",
    "ActionType" : "ActionType_frecuencia",
    "InitiatingProcessAccountName" : "InitiatingProcessAccountName_frecuencia",
    "InitiatingProcessCommandLine" : "InitiatingProcessCommandLine_frecuencia",
    "id_feature" : ["DeviceName", "DeviceName_num"]
}

# Características de entrenamiento iniciales
features =
["InitiatingProcessFileName_frecuencia", "ActionType_frecuencia", "InitiatingProcessAccountName_frecuencia", "InitiatingProcessCommandLine_frecuencia", "DeviceName_num", "hour", "day_of_week"]

# Hacemos una copia de los datos originales
data_actividad_de_equipos_copy = data_actividad_de_equipos.copy()

# Entrenamiento del modelo, prediccion de los datos y extraccion de los resultados y evaluaciones
data_actividad_de_equipos_copy, anomalias_actividad_de_equipos,
no_anomalias_actividad_de_equipos, metricas_del_modelo =
isolation_forest_function(data_actividad_de_equipos_copy, "big_data", features, categorical_features,
"equipos_bigData_IF")
```

Script 9. Script para realizar el entrenamiento del modelo, las predicciones de los datos y la extracción de los resultados, esto, para un análisis de volumen alto.

Evaluación del modelo, en la **Figura 26** podemos observar las métricas del porcentaje de anomalías en los datos, el MSE para datos anómalos y no anómalos; además, de un gráfico de la distribución de las puntuaciones de anomalías en la **Figura 27** y un gráfico de mapa de calor de las anomalías en la **Figura 28**.

```
print(metricas_del_modelo)
✓ 0.0s Open 'metricas_del_modelo' in Data Wrangler Python
{'porcentaje_de_anomalias': 1.6573380304357934, 'mse_datos_no_anomalos': 0.009590642258231882, 'mse_datos_anomalos': 1.0276980797463446}
```

Figura 26. Podemos extraer el valor de 1.66% de anomalías en todo el dataset, siendo un valor muy acertado debido al parámetro de 1% que hemos puesto en el entrenamiento, además el MSE para valores no anómalos nos deja muy claro que los datos normales son más fáciles de predecir debido a su valor muy cercano a 0, pasa lo contrario con el MSE de los valores anómalos, debido a que son más difíciles de separar, su valor es más alto, misma situación que los anteriores modelos.

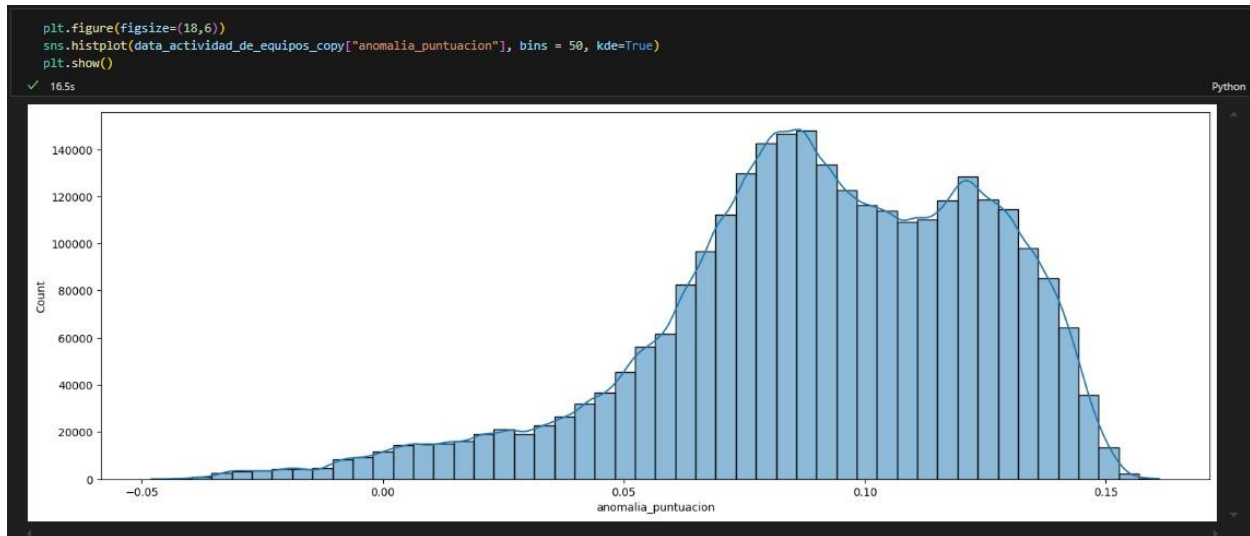


Figura 27. Distribución muy clara de los datos anómalos y no anómalos, siendo complemento del anterior análisis.

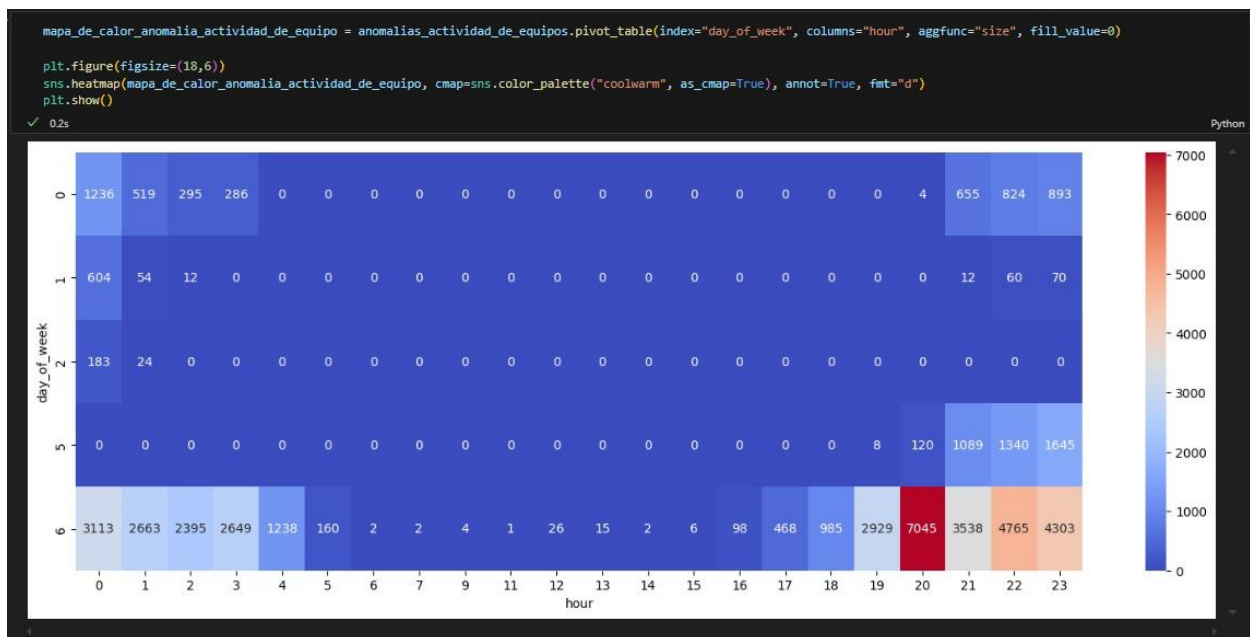


Figura 28. Aquí podemos observar que las anomalías se concentran mucho en puntos bien específicos de los días de la semana y las horas.

Análisis de actividades de tráfico

En el **Script 10** veremos implementación para la actividad de tráfico.

Características de entrenamiento

```
features = {
    "categorical":
    ["DeviceProduct", "ApplicationProtocol", "DeviceInboundInterface", "SourcePort", "DestinationPort", "Protocol"],
    "numeric": ["ReceivedBytes", "SentBytes", "Duration", "hour", "day_of_week"]
}
```

Copia de los datos originales

```
data_actividad_de_trafico_copy = data_actividad_de_trafico.copy()
```

Entrenamiento del modelo, predicción de los datos y extracción de los resultados y evaluaciones

```
data_actividad_de_trafico_copy, anomalias_actividad_de_trafico, no_anomalias_actividad_de_trafico,
secuencias_anomalias, secuencias_normales, error_reconstruccion, tamano_secuencia =
temporal_sequential_function(data_actividad_de_trafico_copy, features, "trafico_TS")
```

Script 10. Script para realizar el entrenamiento del modelo, las predicciones de los datos y la extracción de los resultados.

Evaluación del modelo, en la **Figura 29** podemos observar la distribución del error de reconstrucción.

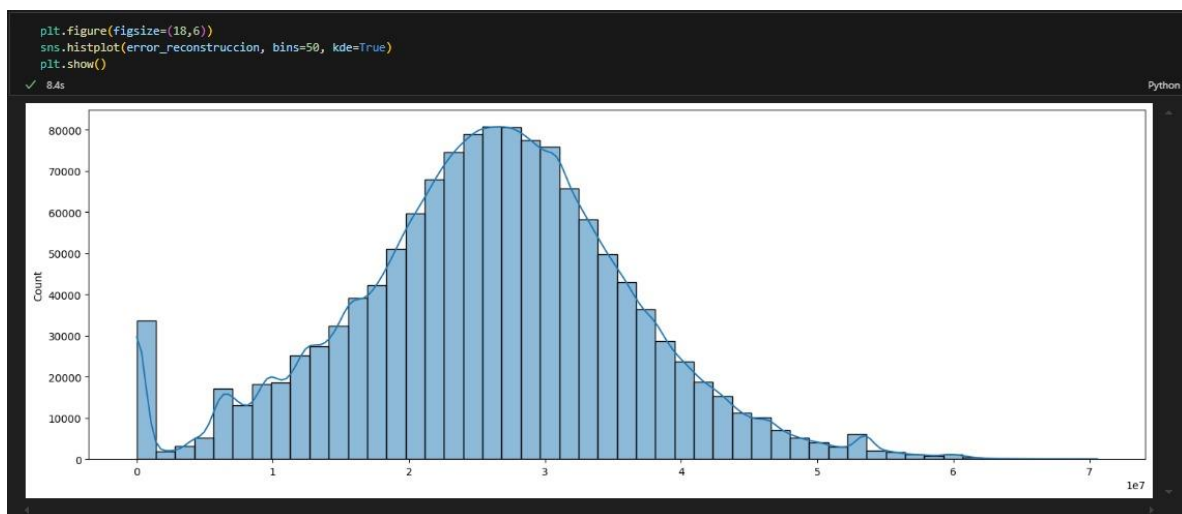


Figura 29. Nos indica que ambos extremos salen de la curva, por tanto, tendremos anomalías, en general muy bien el modelo, sigue el comportamiento normal de la gráfica con puntos de anomalías esperados.

Reporte de Power BI

Para poder tener una visualización de los datos mas general, mas visual y mas interactiva hemos realizado un reporte de power BI, para nuestras detecciones de anomalías en cada aplicación. Es importante destacar que, para este fin, se realizaron funciones distintas para poder ocupar los modelos entrenados y guardar los resultados en un CSV y acumular este proceso dentro de un rango de tiempo especificado. En el **Script 11 y 12** veremos las funciones para Isolation Forest y LSTM.

Funcion extraer los datos en un rango de tiempo establecido y guardarlo en un CSV

def

isolation_forest_prediction_function(data_copy,train_mode,features,categorical_features,model_name=""):

Convertimos a formato de fecha la columna correspondiente y posteriormente poder sacar los datos de tiempo necesarios

data_copy["TimeGenerated"] = pd.to_datetime(data_copy["TimeGenerated"])

Extraemos la hora del registro de tiempo

data_copy["hour"] = data_copy["TimeGenerated"].dt.hour

Extraemos el dia de la semana de tiempo, yendo del 0 al 6 correspondientes de lunes a domingo

data_copy["day_of_week"] = data_copy["TimeGenerated"].dt.day_of_week

Extraemos el dia de la semana de tiempo, yendo del 0 al 6 correspondientes de lunes a domingo

data_copy["date"] = data_copy["TimeGenerated"].dt.date

Entrenamos segun el modo establecido en los parametros

Para el modo de entrenamiento en volumenes altos

if train_mode == "big_data":

Hacemos una copia de los datos originales para poder visualizar todas las características

con los resultados

data_copy_original = data_copy.copy()

Hacemos el tratamiento de las variables

for clave, valor in categorical_features.items():

Se factoriza la variable tipo ID


```
if clave == "id_feature":

    data_copy[valor[1]] = pd.factorize(data_copy[valor[0]])[0]
    break

# Se crea una columna con el numero de ocurrencias de los valores de cada campo
data_copy[valor] = data_copy.groupby(clave)[clave].transform("count")

# Se verifica si existe ya el modelo entrenado
if os.path.isfile(f"models/{model_name}.joblib"):

    # Cargamos el modelo entrenado
    model = load(f"models/{model_name}.joblib")
    # Llamamos la funcion de prediccion de anomalias para Isolation Forest
    prediccion_anomalias_IF(data_copy, model, features)

else:

    print("No existe modelo para cargarlo y hacer las predicciones del dataset")

# Extraemos columnas originales mas el resultado de las anomalias
data_copy = data_copy[data_copy_original.columns.to_list() + ["anomia", "date"]]

elif train_mode == "small_data":

    # Para el modo de entrenamiento en volúmenes bajos
    data_copy_original = data_copy.copy()
    # Sacamos los dummies de los campos
    # Es posible de esta forma por ser un volumen bajo de datos
    # Hay poca variabilidad y además, es algo que el hardware permite procesar
    data_copy = pd.get_dummies(data_copy, columns=categorical_features)
```

```
# Preparamos un arreglo para las columnas de los dummies
columns_dummies = []

# Extraemos solo las columnas creadas de los dummies
for col in data_copy.columns:
    for feature in features:
        if feature in col:
            columns_dummies.append(col)

# Extraemos las características a entrenar
features = ["hour", "day_of_week"] + columns_dummies

# Entrenamos modelo si no existe en el directorio
# Entrenamos con un porcentaje de anomalías de 1%
model = IsolationForest(contamination=0.01)
model.fit(data_copy[features])

# Hacemos la predicción en nuestros datos
prediccion_anomalias_IF(data_copy, model, features)

# Agregamos columnas originales y el resultado de las anomalías
columnas_faltantes_copia = [col for col in data_copy_original.columns if col not in
data_copy.columns]
data_copy = pd.concat([data_copy, data_copy_original[columnas_faltantes_copia]], axis=1)
data_copy = data_copy[data_copy_original.columns.to_list() + ["anomia", "date"]]

# Retornamos solo el dataset para guardar en el CSV
return data_copy
```

Script 11. Script para extraer datos necesarios para el reporte y poder guardar un CSV del modelo de Isolation Forest.

```
# Funcion para el entrenamiento de una red neuronal recurrente LSTM y poder guardar en CSV
def temporal_sequential_prediction_function(data_copy, features, model_name):

    # funcion para crear la secuencia de logs que procederemos a entrenar
    def secuencias_de_logs(datos, tamaño_secuencia):

        # Creamos arreglo para guardar las secuencias
        secuencias = []

        # Creamos la secuencia segun el tamaño de secuencia previamente definido
        for i in range(len(datos) - tamaño_secuencia):
            secuencia_actual = datos[i:i+tamaño_secuencia]
            secuencias.append(secuencia_actual)

        return np.array(secuencias)

    # Convertimos a formato de fecha la columna correspondiente y posteriormente poder sacar los datos de tiempo necesarios
    data_copy["TimeGenerated"] = pd.to_datetime(data_copy["TimeGenerated"])

    # Extraemos la hora del registro de tiempo
    data_copy["hour"] = data_copy["TimeGenerated"].dt.hour

    # Extraemos el día de la semana de tiempo, yendo del 0 al 6 correspondientes de lunes a domingo
    data_copy["day_of_week"] = data_copy["TimeGenerated"].dt.day_of_week

    # Extraemos el día de la semana de tiempo, yendo del 0 al 6 correspondientes de lunes a domingo
    data_copy["date"] = data_copy["TimeGenerated"].dt.date

    # Iniciamos el scaler para tener nuestras características en 0 y 1
    scaler = MinMaxScaler()

    # Hacemos una copia de los datos originales para poder visualizar todas las características
    # con los resultados
    data_copy_original = data_copy.copy()
```

```
# Hacemos tratamiento a nuestras características
for clave, valor in features.items():

    # PArá las variables categorías
    if clave == "categorical":

        for i in range(len(valor)):

            # Convertimos a valores numericos
            data_copy[valor[i]] = LabelEncoder().fit_transform(data_copy[valor[i]])

    elif clave == "numeric":

        # Ocupamos el scaler para hacerl los valores entre 0 y 1
        data_copy[valor] = scaler.fit_transform(data_copy[valor])

# Verificamos si modelo ya existe
if os.path.isfile(f"models/{model_name}.keras"):

    # Definimos tamaño de la secuencia
    tamaño_secuencia = 10

    # Creamos las secuencias de datos para poder ocupar el datasets con el modelo ya entrenado
    x = secuencias_de_logs(data_copy[features["categorical"] + features["numeric"]],
                           tamaño_secuencia)

    # Cargamos el modelo
    model = load_model(f"models/{model_name}.keras")

    # Extraemos los datos relevantes de las predicciones, incluyendo error de reconstrucción del modelo y el umbral de anomalías
    anomalías, no_anomalías, secuencias_anomalías, secuencias_normales, error_reconstrucción,
    umbral_anomalías = prediccion_anomalías_TS(x, model)
```

```

# Creamos campo de anomalia con todos los valores a 0
data_copy_original["anomalia"] = 0

# Extraemos las anomalias y las guardamos en su indice correspondiente
for i in secuencias_anomalas:
    data_copy_original.loc[i:i+tamano_secuencia-1, "anomalia"] = 1

else:
    print("No existe modelo para cargarlo y hacer las predicciones del dataset")

# Retornamos los valores
return data_copy_original

```

Script 12. Script para extraer datos necesarios para el reporte y poder guardar un CSV del modelo de LSTM.

En el **Script 13** veremos la aplicación de esta estrategia, de esta forma podemos replicarla a las demás aplicaciones.

```

# Variables para controlar los tiempos de las funciones de extraccion de datos y el bucle del guardado de
datos
dias = 90
horas_fin = dias * 24
acumulacion_horas = 0
paso_horas = 20
fin_paso = 20

for i in range(0, horas_fin, paso_horas):
    data_actividad_de_usuarios = datos_de_queries(query_actividad_de_usuario, fin_paso, paso_horas,
acumulacion_horas)

categorical_features = ["EventID", "Process", "Account"]

```

```
features = ["EventID_", "Process_", "Account_"]

data_actividad_de_usuarios = isolation_forest_prediction_function(data_actividad_de_usuarios,
"small_data", features, categorical_features)

acumulacion_horas = acumulacion_horas + fin_paso

if i == 0:
    data_actividad_de_usuarios.to_csv('usuarios_prueba/data_actividad_de_usuarios_smallData.csv',
index=False)
else:
    data_actividad_de_usuarios.to_csv('usuarios_prueba/data_actividad_de_usuarios_smallData.csv',
mode='a', header=False, index=False)

print(acumulacion_horas)
if acumulacion_horas >= horas_fin:
    break

del data_actividad_de_usuarios
```

Script 13. Creamos un ciclo para poder guardar los datos en un CSV y poder ocuparlo para elaborar reporte de power BI.

En la **Figura 30, 31, 32 y 33** veremos el resultado del reporte de power BI.

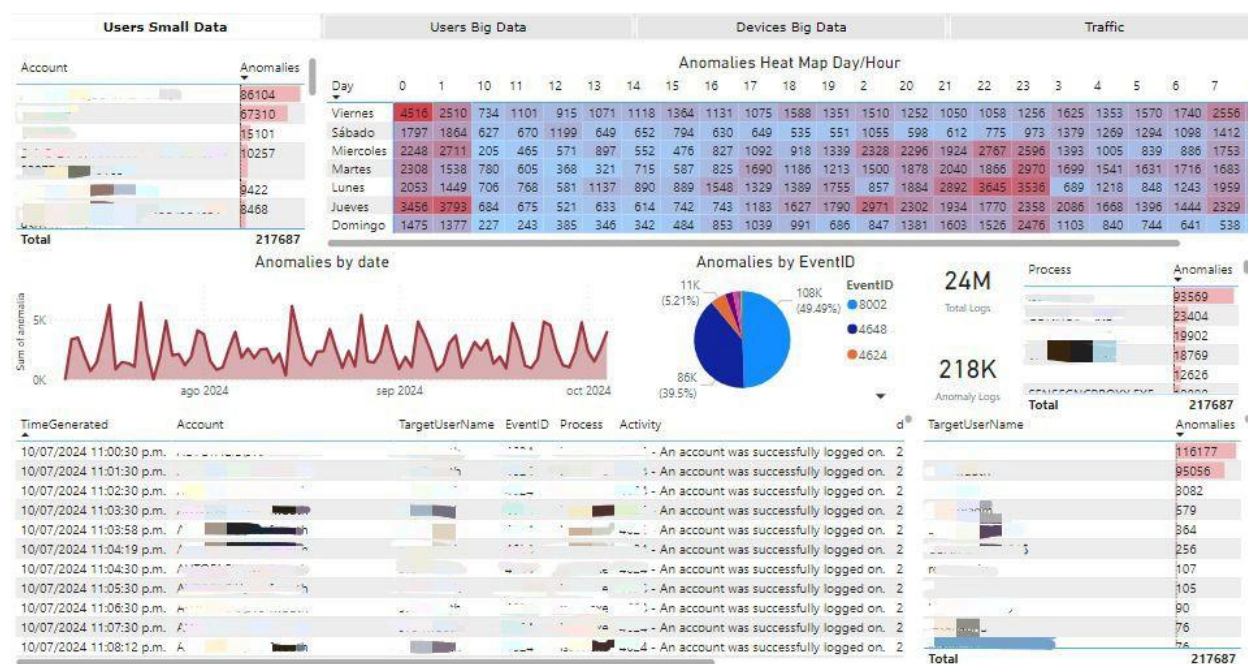


Figura 30. Pestaña de actividad de usuarios para volumen bajo de datos.

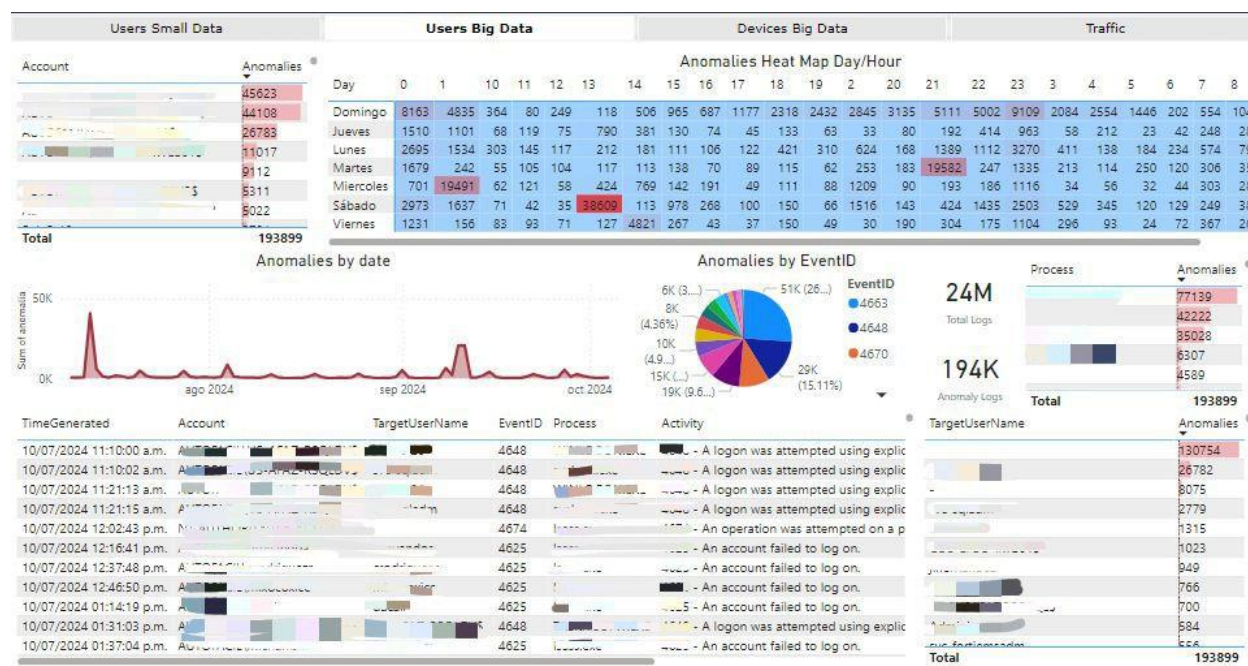


Figura 31. Pestaña de actividad de usuarios para volumen alto de datos.

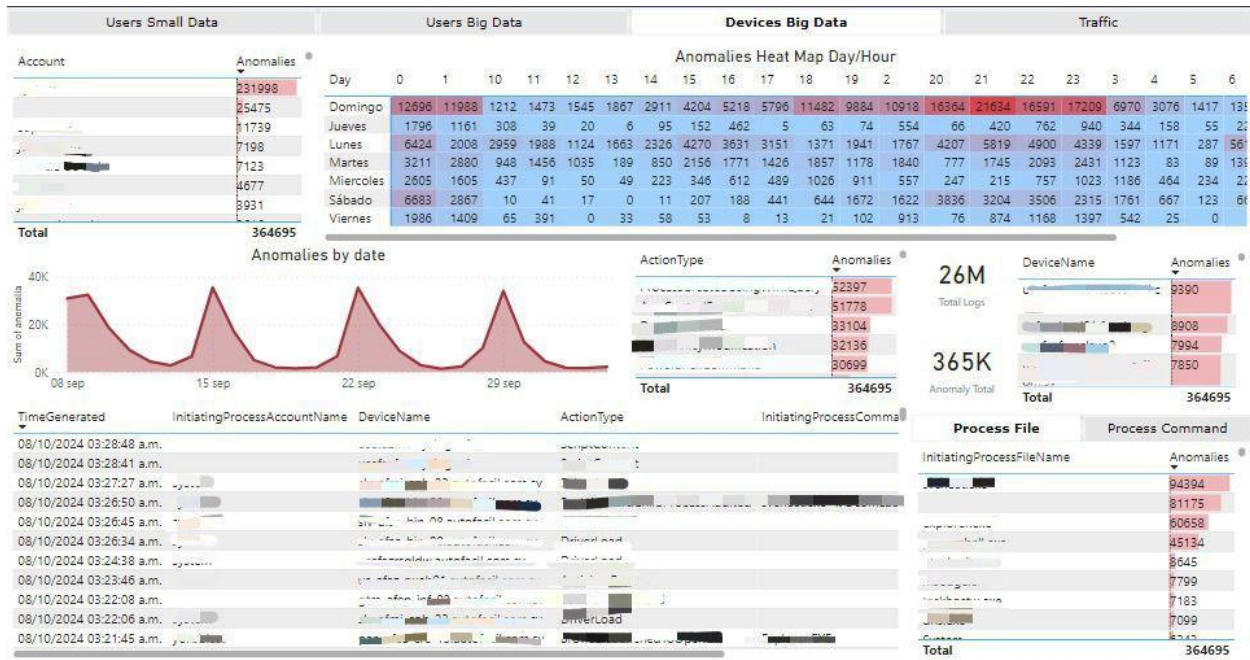


Figura 32. Pestaña de actividad de equipos para volumen alto de datos.

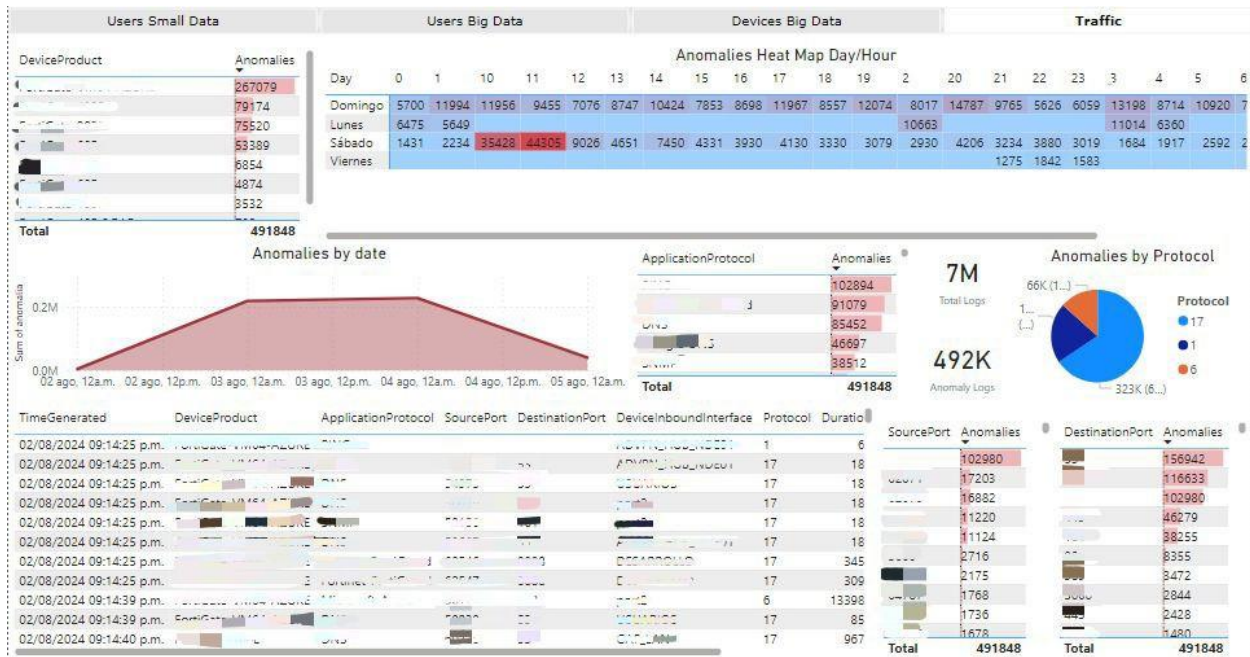


Figura 33. Pestaña de actividad de usuarios para volumen bajo de datos.

CONCLUSIONES

El proyecto ha demostrado la aplicación de las técnicas de Machine Learning y Deep Learning aplicadas al Threat Hunting para detectar amenazas avanzadas que pueden evadir los métodos de seguridad tradicionales. A través del análisis exhaustivo de actividades de usuarios, equipos y tráfico de red, se logró identificar anomalías con alta precisión, lo que puede permitir un análisis muy profundo de las acciones anómalas y las raíces de cada entidad relacionada. El uso de modelos no supervisados como Isolation Forest y LSTM resultó crucial para la detección de patrones atípicos en grandes volúmenes de datos.

Se evidenció que el Big Data es una herramienta fundamental para el desarrollo de soluciones de ciberseguridad adaptativas y escalables, analizar entornos seguros y buscar amenazas o comportamientos anómalos para poder prevenir y tomar acciones. La integración de tecnologías de análisis avanzado con plataformas como Azure Sentinel demostró ser un enfoque poderoso y muy acertado para poder implementar estas soluciones como complemento en un entorno de nube.

Este trabajo pone de relieve la importancia de integrar tecnologías emergentes en la protección de la información y demuestra cómo las arquitecturas basadas en la nube ofrecen un entorno flexible y robusto para la caza de amenazas.

Los resultados también evidencian un gran rango de mejora, recordemos que hemos utilizado datos sin etiquetas, por tanto, los análisis han girado alrededor de las técnicas de datos no supervisados, esto nos comunica la importancia de poder analizar los resultados y etiquetarlos para poder mejorar los modelos entrenados y hacerlos cada vez mas certeros.

Finalmente, esta investigación establece una base sólida para futuros desarrollos en el campo del Threat Hunting, inspirando la implementación de estas técnicas avanzadas en el ámbito de la ciberseguridad. El proyecto destaca el valor de combinar análisis de datos avanzados con estrategias proactivas de defensa cibernética, implementando las bases para un enfoque más dinámico y efectivo en la protección de activos digitales en un panorama de amenazas cada vez más complejo y sofisticado.

REFERENCIAS BIBLIOGRÁFICAS

1. Ashutosh Tripathi. (2021). RNN vs LSTM. <https://ashutoshtripathi.com/2021/07/02/what-is-the-main-difference-between-rnn-and-lstm-nlp-rnn-vs-lstm/>
2. Axelsson, S. (2000). Intrusion Detection Systems: A Survey and Taxonomy. <https://citeseerx.ist.psu.edu/documentrepid=rep1&type=pdf&doi=7a15948bdcb530e2c1deedd8d22dd9b54788a634>
3. Bianco, D. (2021). The Pyramid of Pain. <https://detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html>
4. Cybersecurity Ventures. (2022). The World Will Store 200 Zettabytes Of Data By 2025. <https://cybersecurityventures.com/the-world-will-store-200-zettabytes-of-data-by-2025/>
5. Dhriaj K & James Skelton. (2024). Anomaly Detection Using Isolation Forest in Python. <https://www.digitalocean.com/community/tutorials/anomaly-detection-isolation-forest>
6. IBM. (2023). Cost of a Data Breach Report. <https://www.ibm.com/reports/data-breach>
7. ISACA. (2019). COBIT 2019 Framework: Introduction and Methodology. https://community.mis.temple.edu/mis5203sec003spring2020/files/2019/01/COBIT-2019-Framework-Introduction-and-Methodology_res_eng_1118.pdf
8. Lee, R. M., & Lee, R. (2018). Threat Hunting: Open Source Tools and Techniques.
9. Pallavi Pandey. (2020). Outlier Detection using Isolation Forest. <https://machinelearninggeek.com/outlier-detection-using-isolation-forests/>
10. Rodriguez, R. (2022). The Threat Hunting Project. <https://knowledgehubmedia.com/the-2022-threat-hunting-report/>
11. Ruba Elhafiz. (2023). Modeling an Intrusion Detection Using Recurrent Neural Networks. https://www.sciencedirect.com/science/article/pii/S2307187723000135?ref=pdf_download&fr=RR-2&rr=8d2bc2434c9d741a