



Trabajo Fin De Máster

I Edición Máster en análisis de malware, reversing y bug bounty.

Curso 2020-2021

Detección de compromiso en sistemas Linux basado en comportamiento

Autor:

Jorge Gastón Puig

Tutor:

José Torres

Índice general

Índice general	IV
Índice de figuras	V
Introducción	1
Objetivo	2
Estado del arte	3
Diseño	7
Obtención de métricas	9
Modelado del sistema	13
Detección de anomalías mediante PCA	14
Detección de anomalías mediante Isolation Forest	16
Detección de anomalías mediante GMM	17
Gestión de eventos	19
Arquitectura SW de la herramienta	23
Diagrama de clases	23
Diagrama de actividad	26
Librerías y recursos utilizados	28
Prueba de concepto	29

Repositorio	29
Entorno utilizado	29
Ejecución	30
Emulación de un comportamiento anómalo	30
Aumento del número de procesos	32
Aumento de la carga de la CPU	32
Aumento del uso de la memoria RAM	35
Incremento del número de conexiones TCP/UDP	35
Conclusiones de la prueba de concepto	35
Siguientes pasos	39
Conclusiones	41
Bibliografía	41

Índice de figuras

1.	Bloques principales de la herramienta	7
2.	Resultado análisis PCA	14
3.	Análisis BIC	18
4.	Email con la información de los modelos	21
5.	Email con la notificación del reentrenamiento	22
6.	Diagrama de clases de la herramienta	23
7.	Diagrama de actividad de la herramienta	26
8.	Almacenamiento de logs	27
9.	Errores detectados por GMM de forma constante	31
10.	Notificación de anomalía al incrementar el número de procesos	33
11.	Notificación debido al aumento de la carga de la CPU	34
12.	Notificación de anomalía debido al número de conexiones	36

Introducción

En los últimos meses los ciberataques se han multiplicado y todo hace indicar que dicho aumento continuará en el corto-medio plazo. Un nuevo modelo de industria donde la digitalización es el eje principal del desarrollo, unido a nuevos modelos de trabajo derivados de la Covid-19, así como una sociedad que cada día integra y demanda más tecnología en su día a día son el escenario perfecto para que los ciberdelincuentes vean multiplicarse las posibilidades de sacar rendimientos a sus actividades delictivas.

Este entorno hace necesario que las empresas e instituciones cuenten con los procesos de bastionado o hardening adecuados para evitar que estos ataques tengan éxito y consigan acceso a sus sistemas. No obstante, ningún sistema es 100% seguro, por lo que adicionalmente a las medidas de bastionado es necesario implantar herramientas que permitan la detección del malware, de tal manera que se bloqueen los posibles ataques y se apliquen las medidas correctivas necesarias.

Uno de los inconvenientes de los sistemas de detección de malware a día de hoy es que se basan en detectar IOCs (indicadores de compromiso) o patrones de comportamiento ya descubiertos y analizados con anterioridad y puestos a disposición de la comunidad. Estos sistemas son eficaces por lo tanto detectando amenazas ya descubiertas y analizadas, pero son vulnerables a nuevos ataques.

Algo común a cualquier tipo de malware es que una vez infectado el sistema objetivo, de una manera u otra altera su comportamiento, lo cual se verá reflejado en ciertos parámetros medibles del sistema:

- Ransomware: Los accesos a ficheros se verán aumentados, por lo que es de preveer que los descriptores de ficheros abiertos aumenten.

- Spyware: Se aumentarán tanto los accesos a ficheros para obtener la información como las conexiones con el servidor donde se envíe dicha información.
- Botnets: Creará conexiones con el servidor command and control, y aumentará el uso de la CPU cuando tenga que ejecutar alguna tarea.

Los sistemas de detección de malware tradicionales se basan en detectar las características del malware para detectarlo y bloquearlo. Para ello se sirven de IOCs conocidos, como pueden ser hashes, ficheros abiertos, conexiones de red, etc; o bien implementan técnicas de análisis heurístico que permiten detectar malware desconocido o del que se conocen pocos detalles, basándose por ejemplo en análisis de código estático y dinámico en entornos controlados como sandboxes. Ambas aproximaciones tienen en común que se utilizan el comportamiento del malware para detectarlo.

Objetivo

Basándose en la afirmación anterior de que todo malware modifica el comportamiento del sistema infectado, en este trabajo se propone desarrollar una herramienta de detección de malware para sistemas Linux que permita detectar malware mediante la monitorización de la actividad del sistema. Para ello se identificarán los principales parámetros de caracterizan el funcionamiento del sistema, para generar estadísticas que representen el funcionamiento normal, y a continuación modelar el sistema mediante técnicas de machine learning. Dicho modelado permitirá predecir los valores de dichos indicadores y compararlos con los reales del sistema, de manera que se generen alertas en caso de que existan desviaciones entre los valores predichos y los reales.

Estado del arte

Existen diferentes tipos de herramientas de seguridad cuyo objetivo es proteger a las redes y sus dispositivos de los ataques. Aunque el objetivo final es el mismo, cada tipo de herramienta utiliza una serie de técnicas y tiene alcances diferentes. En este apartado se va a hacer un breve resumen de los diferentes tipos de herramientas que podemos encontrar, explicando brevemente su cometido. Asimismo, se van a listar algunas soluciones concretas para entornos Linux.

Endpoint Protection Platform

Los Endpoint Protection Platform (o EPP) son las herramientas que habitualmente conocemos como antivirus. Su función es prevenir y detectar ataques, así como actividad maliciosa en los sistemas. Las técnicas utilizadas por estos sistemas son diversas, como pueden ser comprobación de IOCs conocidos o análisis de comportamiento del sistema. Pueden hacer uso de recursos en red para no tener que almacenar localmente la base de datos con todos los IOCs conocidos.

Intrusion Detection System

Los Intrusion Detection System (IDS) son sistemas de detección de intrusiones, cuyo objetivo es detectar accesos no autorizados a un ordenador o red. Para ello, estas herramientas monitorizan y analizan el tráfico de red para generar alarmas que alerten a los administradores de la red sobre los eventos detectados. Los IDS utilizan dos métodos diferentes para la detección de estos eventos:

- Detección basado en firma: Mediante este método, el tráfico analizado se compara con patrones conocidos de ataque (o firmas), de manera que el sistema es capaz de detectar

dichos ataques. Estos patrones han de estar previamente almacenados, por lo que si el IDS no está actualizado o el ataque es nuevo y no está registrado en su base de datos, el sistema no será capaz de detectarlo.

- **Heurística:** Este método consiste en detectar los ataques mediante la caracterización estadística de la actividad de la red, como puede ser el ancho de banda utilizado, los puertos, protocolos, etc.

Los IDS son herramientas pasivas en el sentido que por sí solos no bloquean ni mitigan el ataque o acceso no autorizado, sino que se limitan a generar una alarma que deberá ser gestionada por los administradores de la red.

Host-Based Intrusion Detection System - HIDS

Los HIDS son sistemas similares a los IDS desde el punto de vista que buscan detectar anomalías que indiquen un riesgo para la máquina local, y en caso de detectarlo lo bloquean. La diferencia es que mientras que los IDS actúan a nivel local de una máquina, los HIDS despliegan agentes locales en las diferentes máquinas de la red y hacen uso de un servidor centralizado que analiza en tiempo real los datos enviados por los diferentes agentes locales. Otra de las características de los HIDS es que analizan los eventos de los dispositivos en los que hay agentes locales, el lugar de analizar el tráfico de red que va a través de ellos. Esto implica que los HIDS son capaces de detectar un mayor número de amenazas, aunque por contra dependen del funcionamiento de un servidor central, y requieren una gestión más compleja.

Network Intrusion Detection System - NIDS

Los NIDS son herramientas que analizan el tráfico de red para detectar amenazas tales como ataques de denegación de servicio, escaneadores de puertos o intentos de entrar en un ordenador, analizando el tráfico en la red en tiempo real. Para ello analizan los paquetes de red tanto entrante como saliente.

Intrusion Prevention System

Los Intrusion Prevention System (IPS) al igual que los IDS son sistemas que monitorizan y analizan el tráfico de red para detectar anomalías, pero a diferencia de los IDS, los IPS son capaces de llevar a cabo acciones correctivas para evitar el ataque, como pueden ser descartar paquetes o bloquear las conexiones sospechosas modificando las reglas del firewall. Al igual que los IDS también generan las alarmas correspondientes.

Endpoint Detection and Response - EDR

Son herramientas que monitorizan tanto la red como los sistemas locales o endpoints para protegerlos de amenazas. Realizan monitorización y análisis continuo y utilizan diferentes técnicas para ofrecer una respuesta rápida ante amenazas complejas.

Los EDR combinan los antivirus tradicionales con técnicas más avanzadas como la inteligencia artificial, big data o sandboxing para detectar actividad maliciosa en los sistemas y bloquear y prevenir los ataques. Además también permiten integrarse con herramientas antimalware como SIEM. El objetivo de estos sistemas es el mismo que el de los antivirus tradicionales, en el sentido de que persiguen bloquear amenazas, sin embargo el hecho de usar técnicas más avanzadas hace que sean capaces de detectar amenazas más complejas como 0-day o APTs.

En definitiva, los EDR son sistemas que aúnan diferentes herramientas de seguridad para dar una solución global.

Security Information and Event Management

Los sistemas SIEM o sistema de gestión de eventos e información de seguridad son soluciones que centralizadas abarcan tanto la gestión de la información de seguridad como la gestión de eventos. La tecnología SIEM proporciona un análisis en tiempo real de las alertas de seguridad generadas por los distintos dispositivos de la red, y al mismo tiempo recoge los logs de dichos sistemas. Con dicha información los sistemas SIEM detectan eventos de seguridad que puedan suponer actividad maliciosa y que pueden ser causadas por un incidente de seguridad. En resumen, son soluciones que se integran con otras herramientas para la detección de amenazas.

Rootkits detectors

Los detectores de rootkit o rootkit scanners son herramientas que, como su propio nombre indica, persiguen detectar rootkits en la máquina.

Herramientas de seguridad para Linux

A continuación se muestra un listado de algunas de las herramientas de seguridad para Linux que podemos encontrar en la actualidad. Sin ser un listado exhaustivo de herramientas, se pretende mostrar el gran abanico de posibilidades existentes, tanto en tipos de herramientas como en su objetivo.

- ClamAV: Software antivirus con licencia GPL que a su vez incluye varias herramientas de detección de amenazas que pueden ejecutarse en paralelo. Es una de las herramientas más extendidas.
- Rkhunter: Es una herramienta Unix de código libre con licencia GPL diseñada para detectar rootkits, backdoors y exploits locales. Para ello compara los hashes SHA-1 de los principales ficheros de la máquina con el valor esperado.
- OSSEC: es un HIDS de código libre que realiza tareas de análisis de logs, detección de rootkits, generación de alertas y respuesta ante amenazas.
- WAZUH: Es un HIDS que surge como una derivación de OSSEC y que se integra con la pila ELK.
- Linux Malware Detect (LMD): Herramienta que permite escanear equipos en busca de malware y reportarlo.
- OpenVas: Escáner de vulnerabilidades que permite detectar vulnerabilidades analizando equipos de red y servidores.

Diseño

Para el desarrollo de la herramienta se propone el diagrama de la imagen 1, en la que se muestran los cuatro bloques principales que la componen:

- Obtención de métricas.
- Modelado del sistema.
- Monitorización y gestión de eventos.
- Reaprendizaje.

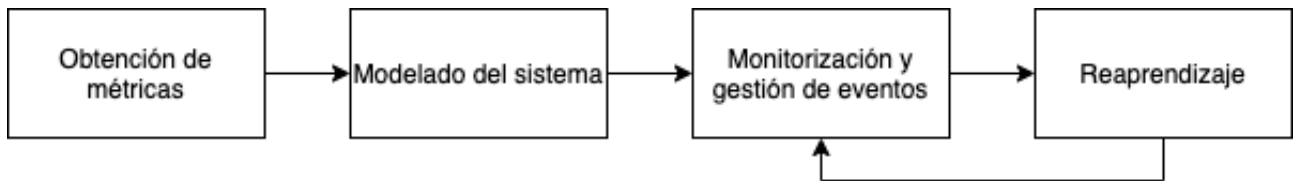


Figura 1: Bloques principales de la herramienta

Obtención de métricas

En primer lugar es necesario listar las métricas que servirán para caracterizar el sistema y obtener datos durante un cierto periodo temporal. Dichas métricas serán parámetros característicos del sistema que puedan verse afectados por la presencia de malware, tal y como la carga de la CPU, los usuarios conectados, los puertos abiertos, número de conexiones SSH o número de descriptores abiertos entre otros.

Modelado del sistema

Los datos obtenidos del apartado anterior servirán como entrada para modelar el sistema mediante algoritmos de machine learning. Esta fase de modelado incluye también el análisis de componentes principales o PCA. Es habitual en los dataset utilizados en problemas de machine learning que las variables de entrada estén correladas entre ellas. Esto implica que hay información redundante que empeora el rendimiento del modelo, y que por lo tanto conviene eliminar para reducir la carga computacional. El PCA permite realizar este estudio y reducir las dimensiones del problema. Además de realizar el PCA en esta fase se evaluarán diferentes alternativas a la hora de modelar el sistema para finalmente escoger la que mejor se adecue al problema y permita obtener mejores resultados.

Monitorización y gestión de eventos

Una vez el sistema esté modelado será capaz de predecir los valores esperados para las métricas que nos interesen. De esta manera, el sistema comparará el valor real leído del sistema con el predicho por el modelo. Cuando se detecte alguna anomalía en estos parámetros, el sistema enviará algún tipo de notificación en la que se indique la detección de dicha anomalía, así como los parámetros que la han provocado.

Reaprendizaje

Es de prever que el modelo del sistema se vea modificado a lo largo del tiempo, en el caso por ejemplo de que se instalen nuevos servicios o se desconecten otros. Es por ello que es necesaria una fase de reaprendizaje del modelo, el cual se llevará a cabo con los datos obtenidos durante la fase de monitorización.

La aplicación se ha desarrollado en Python3, debido a las ventajas que provee a la hora de portarla a diferentes arquitecturas y sistemas, así como por la existencia de librerías como numpy, pandas y scikit-learn que facilitan el desarrollo de aplicaciones de machine learning.

Obtención de métricas

Las métricas son los indicadores que permitirán modelar el funcionamiento normal de nuestro sistema y las que se monitorizarán para detectar posibles anomalías que correspondan con ataques al sistema. De manera periódica la herramienta realiza una lectura de cada estas métricas. Para su lectura, en algunos casos se obtienen mediante utilidades del sistema (comandos como “ps” o “netcat”), en otros mediante la librería “psutil”, y en otros procesando ficheros del sistema. A continuación se detallan los diferentes indicadores utilizados así como el proceso para obtenerlos.

Timestamp

Para cada lectura de datos que realice la aplicación, almacenará un timestamp que será el número de segundos transcurrido desde las 00:00 horas de ese mismo día. Este valor es importante debido a que los ciclos de uso de una máquina habitualmente suele ser de un día. Es de suponer que en horas punta del uso de la máquina, existan varios usuarios conectados, así como una mayor probabilidad de tener fallos de autenticación, mayor carga de la CPU, etc. Sin embargo en horas de menos uso, lo normal es que ocurra al contrario. Para obtener el timestamp se hace uso de la librería “datetime” de Python.

Número de descriptores de fichero

Para la lectura del número de descriptores de fichero abiertos procesará la salida del siguiente comando:

```
lsof | wc -l
```


Número de procesos en ejecución

El número de procesos se obtendrá procesando la salida del siguiente comando:

```
$ ps aux | wc -l
```

En Python se puede ejecutar un comando del sistema y obtener su salida mediante “os.popen()”. En el resto de métricas que se indique el comando utilizado, se hace uso de dicha función para obtener el valor.

Porcentaje de uso de la CPU

Para obtener porcentaje de uso de la CPU se utiliza el método `cpu_percent` de la librería “psutil”:

```
cpu_usage = psutil.cpu_percent(2)
```

Se calcula el porcentaje de uso durante 2 segundos para evitar variabilidad.

Carga media de la CPU

Para obtener la carga media de la CPU se utiliza el método `cpu_getloadavg()` de la librería “psutil”:

```
load1, load5, load15 = psutil.getloadavg()
cpu_load = (load15 / os.cpu_count()) * 100
```

Se calcula la carga de la CPU durante los últimos 15 minutos.

Uso de la memoria RAM

Al igual que en el caso anterior, para obtener el uso de la memoria RAM se utiliza la librería “psutil”, mediante la siguiente función:

```
ram = psutil.virtual_memory()[2]
```

Número de usuarios activos

El número usuarios conectados se obtiene mediante la salida de:

```
$ who | wc -l
```

Número de conexiones TCP/UDP

Para la obtención del número de conexiones TCP/UDP se ejecuta el siguiente comando:

```
$ netstat -t -u | wc -l
```

Número de conexiones SSH

La utilidad “netstat” se utiliza también para obtener el número de conexiones SSH activas, procesando la salida del comando de la siguiente manera:

```
$ netstat -tna | grep 'ESTABLISHED.*sshd' | wc -l
```

Bytes enviados y recibidos

Para obtener los bytes transmitidos y recibidos, se leen los datos de los ficheros “/sys/class/net/eth0/statistics/tx_bytes” y “/sys/class/net/eth0/statistics/rx_bytes”. En este caso no se guarda el valor total, si no que se utiliza la diferencia con respecto a la anterior lectura.

Número de fallos de autenticación

Para la detección de logins fallidos se procesa el fichero “/var/log/auth.log”. Al igual que en el caso anterior, no se considera el número total de logins fallidos, si no la diferencia con la lectura anterior.

Modelado del sistema

Una vez obtenidas las métricas que caracterizan a nuestro sistema es necesario modelarlo. En el propio diseño de la herramienta se incluye una fase de reaprendizaje, sin embargo, es necesario un proceso de recogida de métricas para poder realizar un análisis y ajuste previo del modelo. En algunos modelos es necesario establecer ciertos parámetros de configuración, o un umbral a partir del cual una predicción se considere anomalía. Dichos parámetros se determinarán en este análisis previo.

De entre los diferentes algoritmos de Machine Learning, los que más se ajustan al problema que se pretende resolver mediante este proyecto son aquellos algoritmos no supervisados cuyo objetivo es detectar anomalías. En este caso no contamos con datos etiquetados, ya que todas las medidas que toman corresponden con un funcionamiento normal del sistema. Por otro lado no se ha identificado un parámetro claro del sistema que se pueda predecir en base al resto de métricas, por lo tanto los algoritmos de regresión tampoco parecen los más adecuados.

Para la detección de anomalías se propone combinar tres modelos diferentes, los cuales se listan y detallan a continuación:

- Detección de anomalías mediante análisis de componentes principales.
- Detección de anomalías mediante Isolation Forest.
- Detección de anomalías mediante GMM (Gaussian Mixture Modelling).

A lo largo de las siguientes secciones se van a mostrar los detalles para la generación de cada uno de los modelos utilizados por la herramienta. Explicar la teoría detrás de estos algoritmos queda fuera del alcance de este proyecto, sin embargo sí que es interesante detallar algunos aspectos de cada modelo que hay que tener en cuenta a la hora de utilizarlos.

Detección de anomalías mediante PCA

El análisis de componentes principales (o PCA, principal component analysis) es una técnica que permite reducir la dimensionalidad de un conjunto de datos. Es habitual que entre todas las métricas capturadas haya información redundante, por ejemplo es de esperar que haya cierta relación entre los bytes transmitidos y recibidos. El hecho de que haya información redundante en el vector de entrada hace que se añada complejidad al problema a tratar y por lo tanto sea necesaria más capacidad de cómputo. El PCA permite reducir la dimensionalidad del problema al mismo tiempo que se mantiene la información de los datos de entrada.

A partir de los datos de entrada, el análisis PCA genera una serie de componentes que no están correlacionadas entre sí. Cada una de estas componentes se denomina componente principal, y se obtiene como una combinación lineal de las variables originales. Cada componente contiene una parte de la información de los datos de entrada, la cual se ve reflejada en la proporción de varianza explicada. Si un problema tiene 'n' variables de entrada, el análisis PCA puede generar hasta 'n' componentes. En ese caso, a partir de los componentes se podrán reconstruir los datos originales, ya que la varianza acumulada de todos los componentes será 1. Si se reduce el número de componentes a usar, la varianza acumulada será menor que 1 y por lo tanto empezará a haber cierto error en los datos reconstruidos. En la imagen 2 se muestran los gráficos con la varianza explicada de cada componente (2a) y la varianza acumulada del modelo según el número de componentes usados (2b).

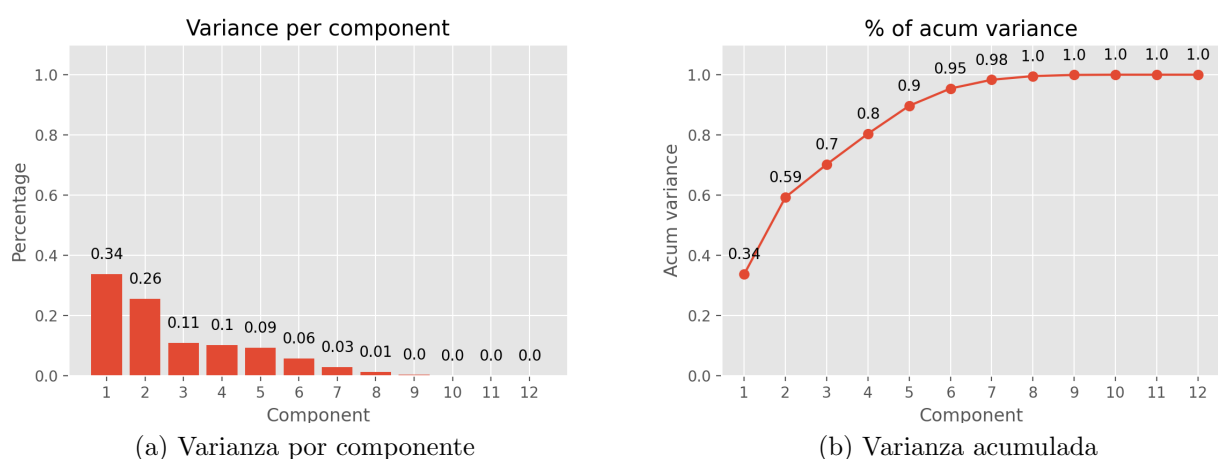


Figura 2: Resultado análisis PCA

La detección de anomalías que se propone en este proyecto consiste en realizar un PCA de los datos obtenidos para entrenar el modelo. Una vez obtenidos las componentes principales, se utilizan para transformar los datos obtenidos durante la monitorización, y se realiza de nuevo la transformación inversa. De esta manera, tendremos los datos reconstruidos por el modelo PCA y los datos originales, y por lo tanto podremos calcular el error de reconstrucción. Si el error obtenido es alto, será indicativo de que el modelo no es capaz de reconstruir correctamente los datos de entrada, y por lo tanto querrá decir que los datos son anómalos al no ajustarse al modelo. Se deben tener las siguientes consideraciones a la hora de implementar este modelo:

- Los valores de las variables de entrada de deben estandarizar de manera que todas tengan media 0 y desviación típica 1. Esto es necesario porque en caso contrario una variable podría dominar sobre el resto falseando el resultado final del modelo. En esa situación, el modelo explicaría muy bien una variable concreta, pero no el resto.
- Se debe establecer el número de componentes a utilizar, para lo cual se debe definir el umbral de varianza acumulada al que se quiere llegar. Dicho de otra forma, se debe definir la cantidad de información que se permite perder al modelo. En el caso de este proyecto, y a la vista de los resultados mostrados en la imagen 2, se ha optado por utilizar 8 componentes, ya que a partir de la octava componente la varianza a acumulada es 1. No obstante, ese habitual establecer límites de varianza acumulada en torno al 90 o 95 %, por lo que todavía existe margen para reducir el número de componentes.
- Por último, se debe definir la forma en la que se calcula el error, así como el nivel de error a partir del cual se considera una anomalía. Se puede optar por dos estrategias. Una de ellas sería analizar el error individual de cada una de las variables reconstruidas, y establecer el número de variables que han de superar dicho error para considerar los datos como anomalía. La otra estrategia es hacer un cálculo del error acumulado de todas las variables. En este caso, se ha optado por la segunda opción, calculando el error según la siguiente expresión:

$$error = \sum_{i=1}^n \frac{abs(x_i^2 - m_i^2)}{x_i^2}$$

Siendo x el valor de la variable leída, m el valor reconstruido, y n el número de variables. Se ha establecido que un error acumulado mayor de 10 corresponde con una anomalía, si bien este valor debería ser ajustado en un entorno real en base a los resultados de las predicciones.

Detección de anomalías mediante Isolation Forest

Isolation Forest es un método no supervisado para identificar anomalías cuando los datos no están etiquetados, es decir, es un método que se ajusta perfectamente a la naturaleza del problema a resolver en este proyecto.

Un modelo Isolation Forest está formado por la combinación de múltiples árboles de decisión llamados isolation trees. Para generar los árboles de decisión, en primer lugar selecciona una muestra de una variable del conjunto de datos de forma aleatoria. A continuación se selecciona otro valor aleatorio entre los valores máximos y mínimos de dicha variable y se compara con el valor de la variable seleccionada. Si la variable es mayor o menor se crea una nueva rama y se repite el proceso anterior pero acotando el intervalo, siendo el nuevo máximo o mínimo el valor aleatorio seleccionado que ha generado la bifurcación de la rama. Este proceso se repite de forma recursiva hasta que no se puede ramificar más. Cuantas menos ramas haya necesitado el árbol para aislar al punto, más anómalo será.

Aunque es un modelo no supervisado, a la hora de generar el modelo es necesario indicar como parámetro de entrada el porcentaje de datos anómalos que existen. En este caso, como consideramos que todas las observaciones son correctas, previamente a generar el modelo se contaminan los datos de entrada añadiendo un 1 % de datos aleatorios:

```
def _add_contaminated_data(self, input_data):
    data = input_data.copy()

    mean_data = (np.trunc(data.max(axis=0)) + 1)
    rows_to_add = int(len(data.index) * self.contamination)
    for i in range(0, rows_to_add):
        wrong_data = mean_data * random.randint(10, 20)
```

```
wrong_df = pd.DataFrame([wrong_data], columns=data.columns)
data = data.append(wrong_df, ignore_index=True)
```

Una vez modelado el sistema, ante un conjunto de datos de entrada, la salida será un -1 en caso de que los valores sean anómalos. Además, el modelo también devuelve un valor llamado “anomaly score” que indica el nivel de evidencia de la anomalía, cuando más negativo más probabilidad de anomalía. Así pues, para detectar anomalías Isolation Forest podríamos utilizar ambos resultados. En este caso, el anomaly score únicamente se lee para incluirlo en los logs.

Detección de anomalías mediante GMM

Gaussian Mixture Model (GMM) es un modelo probabilístico en el que se considera que las observaciones siguen una distribución probabilística formada por la combinación de múltiples distribuciones normales (componentes). Ajustar un modelo GMM consiste en estimar los parámetros que definen la función de distribución de cada componente: la media y la matriz de covarianza. Una vez aprendidos los parámetros, se puede calcular la densidad de probabilidad que tiene cada observación de pertenecer a cada componente y al conjunto de la distribución. Observaciones con muy poca densidad de probabilidad pueden considerarse como anomalías.

Para este modelo se debe establecer el tipo de matriz de covarianza y el número de componentes a utilizar. Para obtener el valor óptimo de dichos valores se puede realizar un análisis BIC (Bayesian information criterion) [3]. En la imagen 3 se muestra el resultado de dicho análisis para los datos iniciales del modelo, siendo el número de componentes óptimo 18 y la matriz de covarianza de tipo “diag”. No obstante, para la matriz de covarianza “full” con 7 componentes el resultado es similar.

Tanto la matriz de covarianza como el número de componentes que mejor se ajuste a los parámetros de entrada puede variar a lo largo del tiempo según la distribución de los datos de entrada. Es por ello que antes de entrenar el modelo, la aplicación realiza un análisis para calcularlos. Dicha función se muestra a continuación:

```
def _GMMAnalysis(self, input_data):
    n_components = range(1, self._max_number_of_components)
```

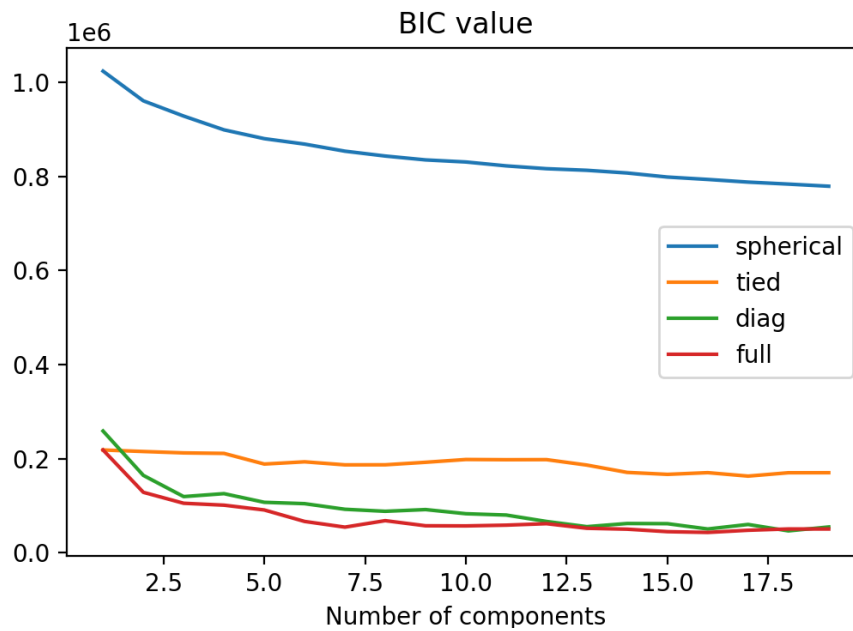



Figura 3: Análisis BIC

```

min_cov_type = None
min_cov_components = -1
min_cov = -1
for covariance_type in self._covariance_type:
    bic = []

    for i in n_components:
        model = self._GMMModel(input_data, i, covariance_type)
        model_data = model.bic(input_data)
        bic.append(model_data)
        if min_cov == -1 or model_data < min_cov:
            min_cov_type = covariance_type
            min_cov_components = i
            min_cov = model_data

    return min_cov_type, min_cov_components

```

Gestión de eventos

Una vez obtenidas las métricas y modelado el sistema, es necesario poner en marcha la aplicación realizando una monitorización en tiempo real del sistema y comprobando si los datos leídos son considerados por el modelo como válidos o anomalías. Para que el usuario disponga de información sobre el funcionamiento de la aplicación se ha implementado un mecanismo de notificaciones vía email. Las notificaciones se envían en las siguientes situaciones:

- Al detectarse una anomalía.
- Al reentrenar el modelo.

En el caso de que detectarse alguna anomalía la aplicación enviará una notificación al usuario informando de la situación, de manera que pueda analizar los datos que han dado lugar a la dicha situación. El mensaje con la notificación contiene la información de salida de los modelos en el cuerpo del mensaje y permite ver qué modelo ha detectado la anomalía. Para cada uno de los modelos, el mensaje contiene los parámetros de entrada así como la salida del modelo. A continuación se muestra un ejemplo del log del modelo GMM:

```
*** GMM PREDICTION REPORT ***
Input Data:
  timestamp  num_process  num_fds  num_conn  num_ssh  num_active_users  cpu_usage  cpu_load  ram  tx_bytes  rx_bytes
  failed_logins
0      57982         171    29174         1         0             1       0.0     5.0  37.1     807     3201
      0

Probability log of the prediction:
[-376.14083369]

Min Prob is :
-140.82360781587914

Anomaly Result:
True
```

En este caso se muestran los parámetros de entrada, el log de probabilidad de dichos parámetros, y el límite a partir del cual se considera anomalía. En el caso del ejemplo, el modelo ha clasificado los datos como anómalos. Para cada modelo la información es diferente. Por ejemplo en el caso de PCA, para los mismos datos de entrada, la salida es:

```
*** PCA PREDICTION REPORT ***
Input Data:
  timestamp  num_process  num_fds  num_conn  num_ssh  num_active_users  cpu_usage  cpu_load  ram  tx_bytes  rx_bytes
  failed_logins
0    57982      171    29174      1      0      1      0.0      5.0  37.1      807    3201
  0

Transformed Data:
  timestamp  num_process      num_fds  num_conn      num_ssh  num_active_users  cpu_usage  cpu_load      ram
  tx_bytes  rx_bytes  failed_logins
0  58645.637492  172.476587  27402.903641  1.049932  1.326948e-15      1.0  -0.004196  4.990863  38.239959
  806.544652  3199.107003      0.000175

Error:
  timestamp  num_process      num_fds  num_conn      num_ssh  num_active_users  cpu_usage  cpu_load      ram
  tx_bytes  rx_bytes  failed_logins
0  7.739847e+07  507.173127  1.002031e+08  0.102357  1.760790e-30      8.881784e-16  0.000018  0.091285  85.884494
  734.724721  12115.381492  3.052699e-08

Accumulated Error:
0.328814288135491

Anomaly Result:
False
```

En este caso se muestran los datos de entrada, los reconstruidos tras aplicar la transformación con los componentes calculados por el PCA, y el error de reconstrucción. Para el modelo Isolation Forest, se muestran los parámetros de entrada y el “score anomaly”, tal y como se muestra a continuación:

```
*** ISOLATION FOREST PREDICTION REPORT ***
Input Data:
  timestamp  num_process  num_fds  num_conn  num_ssh  num_active_users  cpu_usage  cpu_load  ram  tx_bytes  rx_bytes
  failed_logins
0    57982      171    29174      1      0      1      0.0      5.0  37.1      807    3201
  0

Anomaly Score for the input data:
[-0.56147187]

Anomaly Result:
False
```

Para un mismo vector de entrada, es probable que no todos los modelos obtengan el mismo resultado, por lo que en la notificación por mail se envía la información de todos ellos. En la imagen 4 se muestra a modo de ejemplo un mail con una notificación de anomalía detectada.

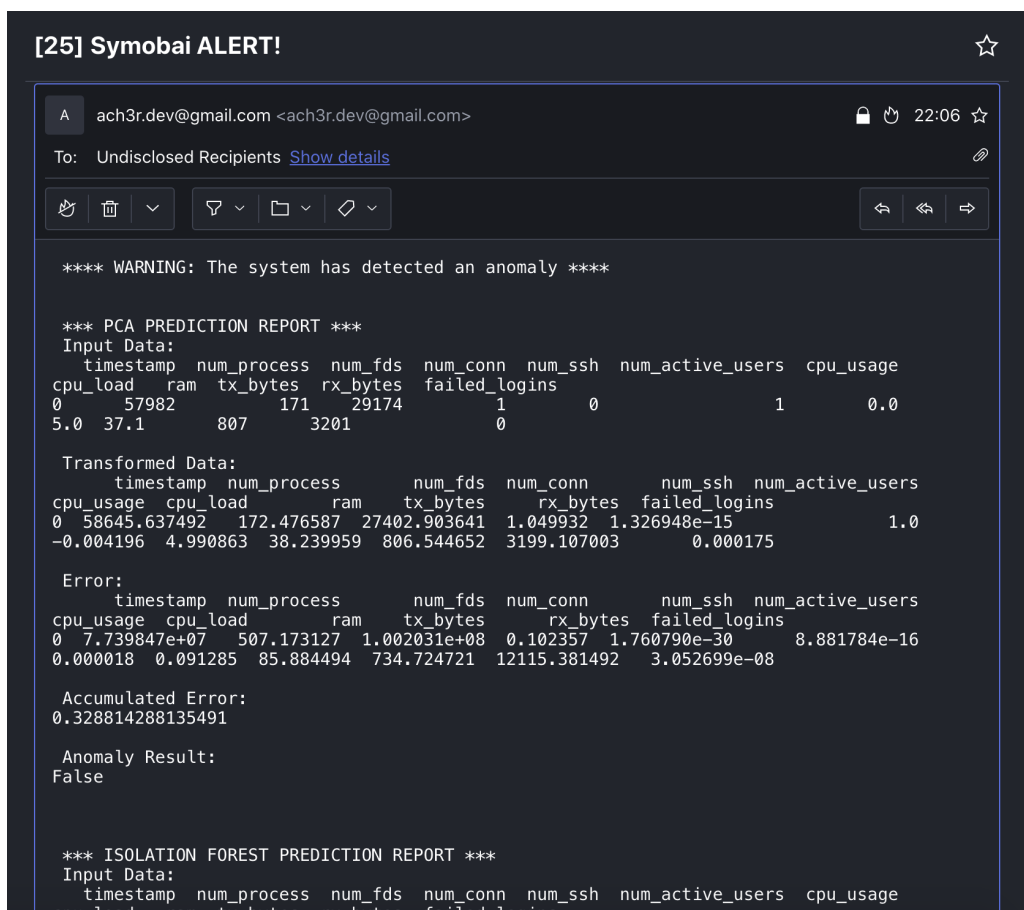


Figura 4: Email con la información de los modelos

Para evitar saturar la bandeja del correo del usuario, la aplicación envía una máximo una notificación de anomalía a la hora. No obstante, almacena en la máquina local un fichero de log para cada uno de las últimas 100 anomalías detectadas.

Además de la información de los modelos, la aplicación envía un informe adjunto con información del sistema. Dicho informe incluye la información obtenida por los siguientes comandos:

```
$ ps aux
$ pstree
$ netstat -a
$ vmstat
$ iostat
$ free
$ df
$ dmesg
```

De esta manera, el usuario de la aplicación puede analizar de forma rápida si la anomalía detectada se debe a alguna circunstancia conocida sin necesidad de tener que acceder a la máquina.

Adicionalmente, también se ha considerado interesante que la aplicación envíe una notificación cada vez que reentrene el modelo, en la que además se adjunte el fichero *.csv con las métricas que se han utilizado para entrenarlo. Así el usuario de la aplicación tendrá disponibles el histórico de datos obtenidos y podrá reproducir los modelos generados por el sistema offline. Se ha establecido un periodo temporal de 24 horas para realizar el reaprendizaje del modelo, por lo tanto se recibirá una notificación de este tipo al día. En la imagen 5 se muestra el contenido de la notificación que envía la aplicación en esta situación.

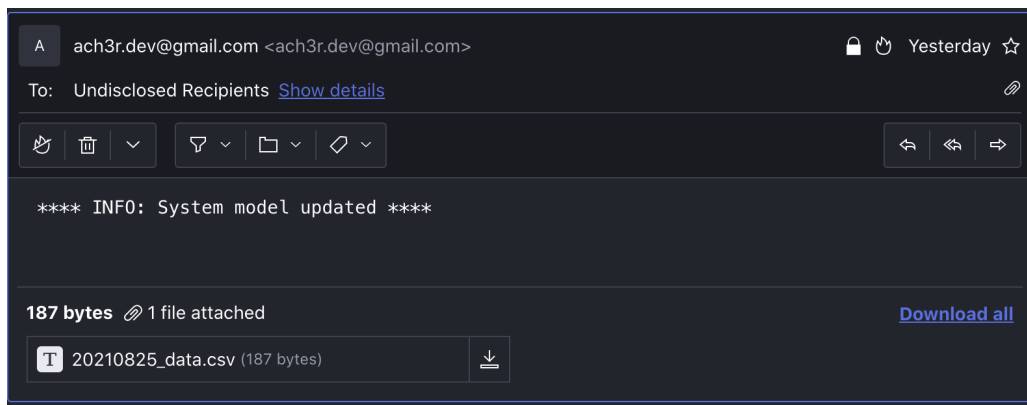


Figura 5: Email con la notificación del reentrenamiento

Arquitectura SW de la herramienta

En este capítulo se va a detallar el diseño de la herramienta mediante sus diagramas de clases y actividad. Sin pretender ser una documentación del software, el objetivo es explicar su funcionamiento y las diferentes decisiones tomadas para llegar a la solución final.

Diagrama de clases

En la imagen 6 se muestra el diagrama de clases de la herramienta, en este apartado se van a describir las clases implementadas, la funcionalidad que implementan y sus interfaces principales.

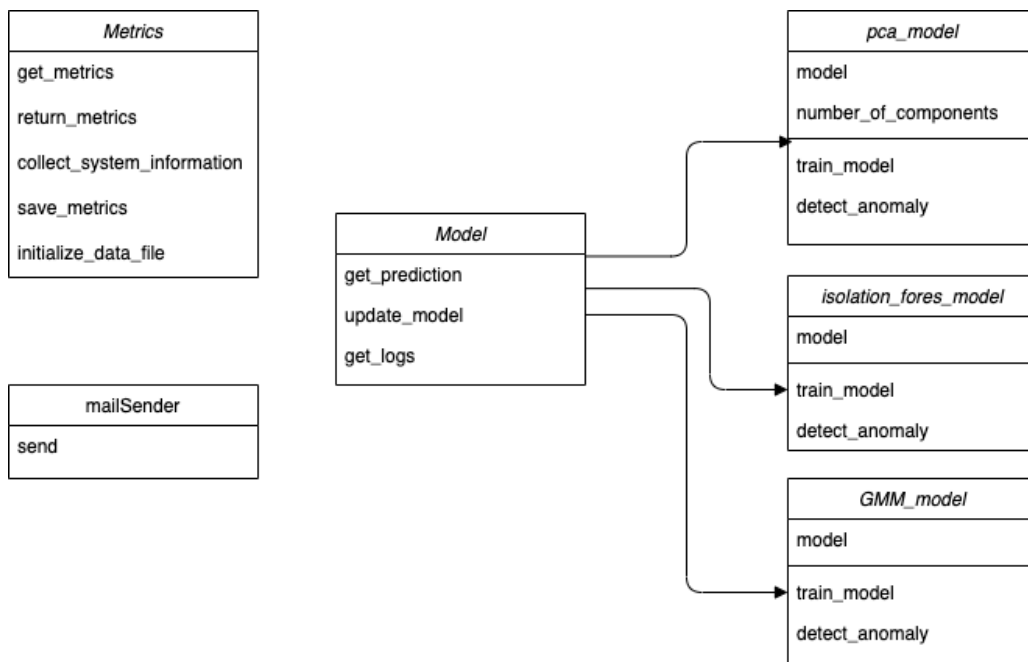


Figura 6: Diagrama de clases de la herramienta

En primer lugar la clase “Metrics” es la encargada de obtener las métricas del sistema, es

decir, su función es leer la información del sistema y dejarla disponible para que el resto de elementos de la herramienta pueda acceder a ella. Los interfaces son los métodos que la clase provee al resto de elementos para acceder a dicha información, y sirven para hacer un repaso de las principales funciones de dicha clase:

- **get_metrics**: Este método se encarga de leer los parámetros del sistema.
- **return_metrics**: Devuelve las métricas leídas en la última ejecución de 'get_metrics'.
- **initialize_data_file**: Mediante este método se inicializa el fichero *.csv donde se van a almacenar las métricas. Se encarga de crear el fichero y añadirle la cabecera correspondiente.
- **save_metrics**: Mediante es método se almacenan los datos obtenidos.
- **collect_system_information**: Este método permite obtener información del sistema y generar un fichero de log con dicha información. Se utiliza cuando se detecta una anomalía para poder tener un informe del estado del sistema que aporte más información que las propias métricas y enviarlo al usuario.

La clase “model” se encarga del modelado del sistema. Como se ha comentado anteriormente, la solución propone combinar diferentes algoritmos de predicción para detectar si existe una anomalía o no. Esta clase es la encargada de agrupar las llamadas a cada uno de los modelos, haciendo transparente para el programa principal dichos algoritmos. De esta manera, si en el futuro se quisiera añadir un nuevo modelo, sería suficiente con añadir la clase correspondiente con dicho modelo, y modificar la clase “model” para integrarlo, sin necesidad de modificar el bucle principal. Los interfaces provistos por la clase model son:

- **update_model**: Se encarga de entrenar los modelos del sistema. Tiene como parámetro de entrada un fichero con los datos de las métricas.
- **get_prediction**: Tiene como entrada una muestra de las métricas obtenidas del sistema, y devuelve si dichos datos se corresponden con una anomalía o no.

- **get_logs**: Este método devuelve una cadena de texto con la información de salida de la última predicción hecha por cada uno de los modelos. Esta información se envía al usuario en caso de detectar una anomalía.

Las clases `pca_model`, `isolation_forest_model` y `GMM_model` son las clases que implementan cada uno de los modelos utilizados. En todos los casos proveen de los siguientes interfaces:

- **train_model**: Se encarga de entrenar el modelo. Tiene como parámetro de entrada un fichero con los datos de las métricas.
- **detect_anomaly**: Tiene como entrada una muestra de las métricas obtenidas del sistema, y devuelve si dichos datos se corresponden con una anomalía o no.
- **get_log**: Este método devuelve una cadena de texto con información del modelo y la predicción.

Como se puede observar, existe una relación uno a uno entre los métodos expuestos por la clase “Model” y los expuestos por cada una de las clases que implementan los modelos de cada algoritmo. Con esto se persigue que la clase “Model” permita abstraer al programa principal de los modelos concretos implementados. Adicionalmente, cada una de las clases que implementan los algoritmos de modelado exponen el atributo “**model**”. Esto permite tener accesible los modelos de forma externa en caso de que sea necesario, ya sea para obtener los logs o para analizar los parámetros característicos de cada modelo.

Por último, la clase “mailSender” permite enviar emails a modo de notificación de los eventos detectados por el sistema. Esta clase tiene únicamente un método llamado “**send**” que se encarga de enviar el mail. Cuando se crea la instancia de la clase “mailSender” se pasa como parámetro de entrada un fichero con los parámetros de configuración:

```
port: <SERVER PORT>
smtp_server: <SMTP SERVER ADDRESS>
sender: <MAIL SENDER ADDRESS>
password: <PASS>
defaultDest: <DEFAULT DESTINATION ADDRESS [optional]>
```


Diagrama de actividad

El diagrama de actividad se muestra en la imagen 7.

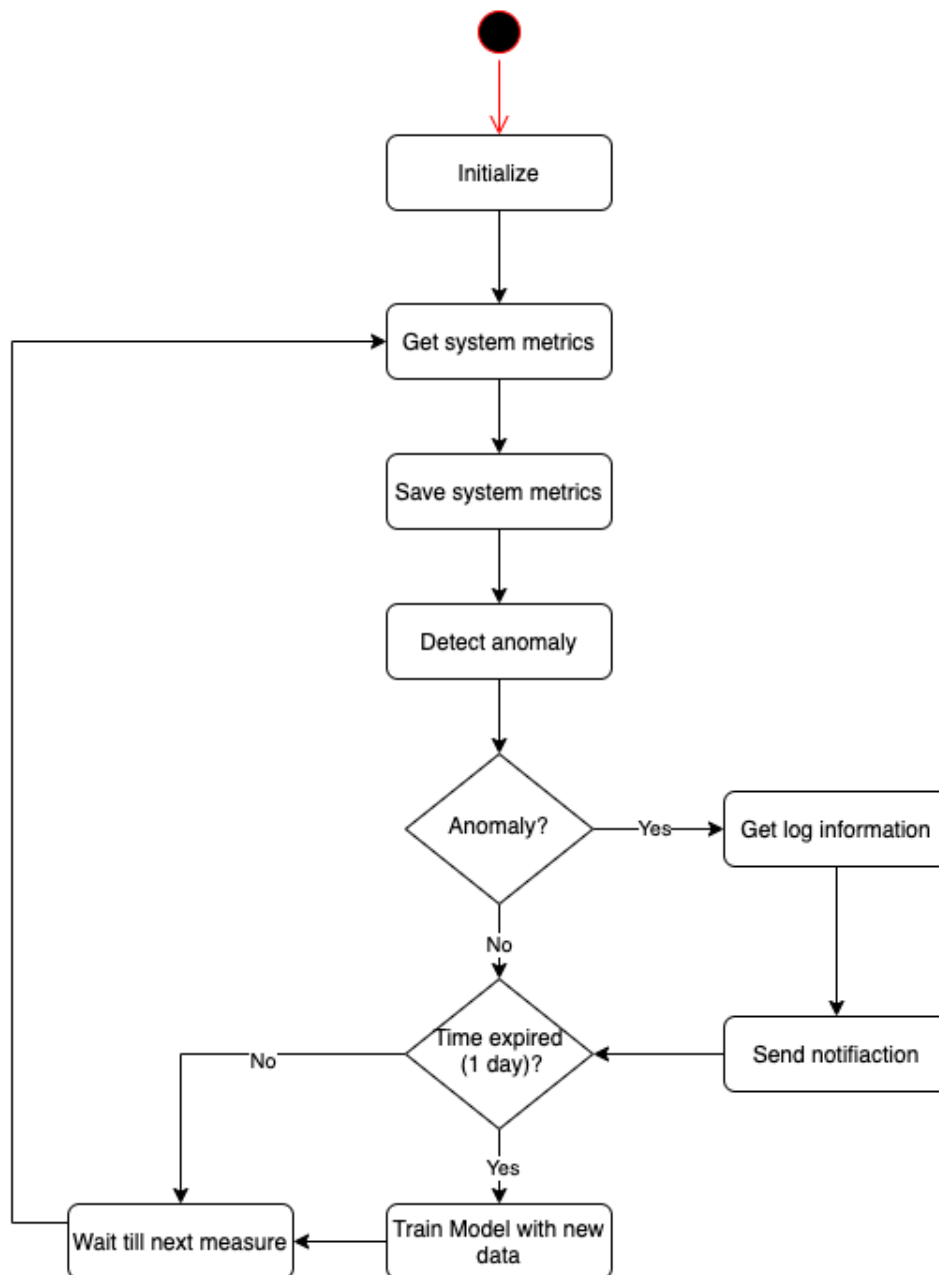


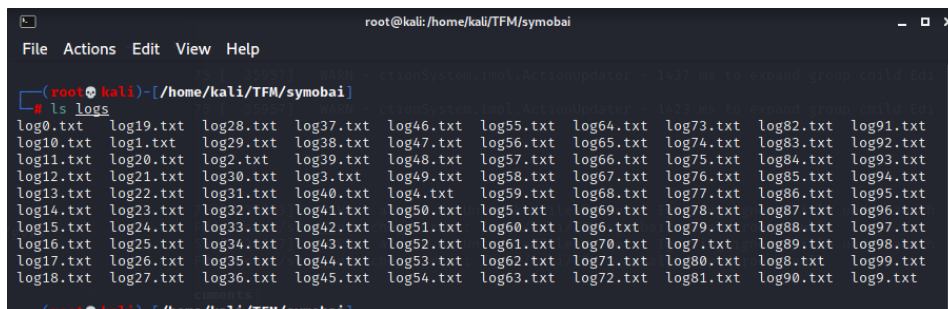
Figura 7: Diagrama de actividad de la herramienta

En primer lugar la aplicación realiza una serie de tareas de inicialización. Entre estas tareas se encuentran crear los modelos y entrenarlos con los datos de entrada iniciales, crear las instancias de las diferentes clases a utilizar, crear el fichero donde se almacenarán las métricas, así como

inicializar una serie de variables auxiliares.

A continuación la aplicación entra entre el bucle principal. En dicho bucle se realiza la lectura de métricas del sistema y su almacenamiento. Almacenar las métricas permitirá reentrenar los modelos a posteriori con datos actualizados.

Las métricas obtenidas del sistema se utilizan como entrada de los modelos, de forma que como salida se obtiene si los datos obtenidos se corresponden con una anomalía o no. En caso de que los modelos detecten que los datos son anómalos, se obtienen los logs de los modelos, así como la información del estado del sistema y se envía un mail al usuario. Para evitar saturar la bandeja de entrada del usuario, se establece un máximo de un mail a la hora, no obstante este parámetro sería fácilmente configurable. Adicionalmente a la notificación por mail, la herramienta almacena los logs de las últimas 100 anomalías detectadas, tal y como se muestra en la imagen 8. De esta manera el usuario puede analizar los logs de las anomalías detectadas aunque no las reciba por mail.



```
root@kali: /home/kali/TFM/symbai
File Actions Edit View Help
(root@kali) - /home/kali/TFM/symbai
# ls logs
log0.txt log19.txt log28.txt log37.txt log46.txt log55.txt log64.txt log73.txt log82.txt log91.txt
log10.txt log1.txt log29.txt log38.txt log47.txt log56.txt log65.txt log74.txt log83.txt log92.txt
log11.txt log20.txt log2.txt log39.txt log48.txt log57.txt log66.txt log75.txt log84.txt log93.txt
log12.txt log21.txt log30.txt log3.txt log49.txt log58.txt log67.txt log76.txt log85.txt log94.txt
log13.txt log22.txt log31.txt log40.txt log4.txt log59.txt log68.txt log77.txt log86.txt log95.txt
log14.txt log23.txt log32.txt log41.txt log50.txt log5.txt log69.txt log78.txt log87.txt log96.txt
log15.txt log24.txt log33.txt log42.txt log51.txt log60.txt log6.txt log79.txt log88.txt log97.txt
log16.txt log25.txt log34.txt log43.txt log52.txt log61.txt log70.txt log7.txt log89.txt log98.txt
log17.txt log26.txt log35.txt log44.txt log53.txt log62.txt log71.txt log80.txt log8.txt log99.txt
log18.txt log27.txt log36.txt log45.txt log54.txt log63.txt log72.txt log81.txt log90.txt log9.txt
```

Figura 8: Almacenamiento de logs

A continuación se comprueba el tiempo que ha transcurrido desde la última actualización del modelo. En caso de ser mayor de un día, se reentrena el modelo con los últimos datos obtenidos y se envía un mail al usuario con los datos que se ha utilizado para el entrenamiento, y que se corresponden con las métricas leídas durante el último día.

Tras enviar la notificación, o si no ha sido necesario reentrenar el modelo, se establece un tiempo de espera hasta la siguiente lectura de métricas. Este tiempo se ha establecido en 20 segundos, pero al igual que antes es fácilmente modificable.

Librerías y recursos utilizados

La herramienta se ha desarrollado en Python 3, haciendo uso de la librería scikit-learn para los algoritmos de machine learning. A continuación se muestra la lista de librerías instaladas en el entorno virtual utilizado para ejecutar la aplicación:

```
# pip list
Package           Version
-----
joblib             1.0.1
numpy              1.21.2
pandas             1.3.2
pip                21.2.4
psutil             5.8.0
python-dateutil    2.8.2
pytz               2021.1
scikit-learn       0.24.2
scipy              1.7.1
setuptools         57.0.0
six                1.16.0
sklearn            0.0
threadpoolctl      2.2.0
wheel              0.36.2
```

Para la obtención de métricas del sistema, se ha combinado la librería “psutil”, lectura de los ficheros del sistema, y ejecución de comandos del sistema y procesado su salida mediante “os.popen”. Por su parte, para obtener la información del estado del sistema que se envía al detectar una anomalía, se ha implementado un script en bash llamado “getSystemInfo.sh”. De esta forma el usuario de la aplicación puede modificar dicho script y así recolectar la información que le resulte de mayor interés. Esta estrategia no se ha seguido con las métricas, debido a que son la entrada de los modelos y por lo tanto requieren de cierto cuidado para asegurar que la aplicación funciona correctamente. Modificar el vector de entrada de los modelos sin un análisis previo podría falsear el resultado final.

Prueba de concepto

Repositorio

La herramienta desarrollada se ha denominado Symobai, que es un acrónimo de “System Model based on Artificial Intelligence”. El código de la aplicación puede encontrarse en el siguiente repositorio: <https://github.com/jrggaston/symobai>

Entorno utilizado

Para probar la herramienta se han utilizado dos entornos diferentes. Por un lado una máquina Ubuntu con las siguientes características:

```
$ uname -a
Linux ubuntu2004 5.8.0-41-generic #46~20.04.1-Ubuntu SMP Mon Jan 18
 17:52:23 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:   Ubuntu 20.04.2 LTS
Release:      20.04
Codename:     focal
```

Y otra máquina Kali con las siguientes características:

```
$ uname -a
Linux kali 5.10.0-kali7-amd64 #1 SMP Debian 5.10.28-1kali1
 (2021-04-12) x86_64 GNU/Linux
```

```
$ lsb_release -a
No LSB modules are available.
Distributor ID: Kali
Description:    Kali GNU/Linux Rolling
Release:        2021.2
Codename:       kali-rolling
```

Ejecución

Para lanzar la aplicación en primer lugar es necesario crear el entorno virtual e instalar las librerías utilizadas. Para ello hay que ejecutar los siguientes comandos:

```
$ python3 -m venv venv
$ source venv/bin/activate
(venv) $ pip install -r requirements.txt
(venv) $ nohup python3 src/main.py &
```

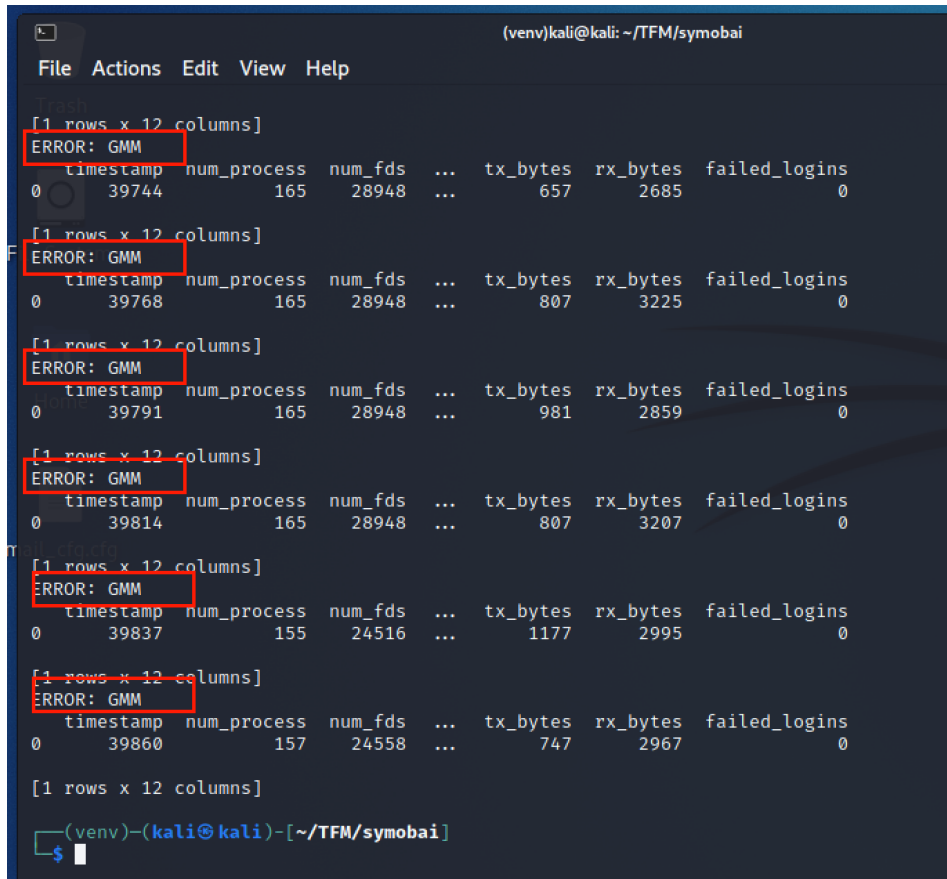
Se recomienda ejecutar el comando con permisos de root, ya que tanto para obtener las métricas como para crear el informe con el estado del sistema la herramienta accede a recursos que requieren dichos permisos.

La primera vez que se lance la aplicación, se entrenarán los modelos con los datos del fichero “dataset/init_data.csv”. Una vez que la aplicación esté ejecutándose durante 24 horas, reentrenará los modelos online con los datos obtenidos durante su ejecución.

Emulación de un comportamiento anómalo

En esta sección se van a emular diferentes comportamientos anómalos del sistema que podrían ser causados por la actividad maliciosa de un malware. Dicho comportamientos se han emulado mediante programas o scripts ad-hoc, o mediante la utilizad “stress-ng”. El objetivo es ver si los diferentes modelos del sistema detectan dicho comportamiento y la herramienta envía la notificación correspondiente.

Antes de realizar las pruebas es importante tener un modelo suficientemente estable, de manera que se asegure que en condiciones normales la aplicación no envía falsos positivos. Por ello antes de realizar ningún ataque se dejó la aplicación ejecutándose durante unas 72 horas, para analizar el comportamiento de la aplicación con el modelo original y con los modelos reentrenados. Tras tener la aplicación ejecutándose durante varios días en una máquina sin apenas uso, se observó que el modelo GMM en algunos casos detectaba anomalías de forma constante (imagen 9).



```
(venv)kali@kali: ~/TFM/symbai
File Actions Edit View Help

[1 rows x 12 columns]
ERROR: GMM
timestamp num_process num_fds ... tx_bytes rx_bytes failed_logins
0 39744 165 28948 ... 657 2685 0

[1 rows x 12 columns]
ERROR: GMM
timestamp num_process num_fds ... tx_bytes rx_bytes failed_logins
0 39768 165 28948 ... 807 3225 0

[1 rows x 12 columns]
ERROR: GMM
timestamp num_process num_fds ... tx_bytes rx_bytes failed_logins
0 39791 165 28948 ... 981 2859 0

[1 rows x 12 columns]
ERROR: GMM
timestamp num_process num_fds ... tx_bytes rx_bytes failed_logins
0 39814 165 28948 ... 807 3207 0

[1 rows x 12 columns]
ERROR: GMM
timestamp num_process num_fds ... tx_bytes rx_bytes failed_logins
0 39837 155 24516 ... 1177 2995 0

[1 rows x 12 columns]
ERROR: GMM
timestamp num_process num_fds ... tx_bytes rx_bytes failed_logins
0 39860 157 24558 ... 747 2967 0

[1 rows x 12 columns]

(venv)-(kali@kali)-[~/TFM/symbai]
$
```

Figura 9: Errores detectados por GMM de forma constante

Esto posiblemente se deba a que el modelo GMM asume que los datos de entrada siguen distribuciones probabilísticas formadas por la combinación de distribuciones normales. En este caso, como la máquina donde se recogieron los datos no tenía actividad, los valores capturados en el tiempo resultaban ser constantes en el tiempo y por lo tanto el modelo GMM no se ajusta correctamente. Probablemente, aplicar el modelo GMM en un sistema con usuarios y/o servicios interactuando de forma activa con él tenga resultados correctos. Por esta razón, para realizar

las pruebas se decidió no tener en cuenta la salida del modelo GMM, aunque sí se incluyó en los logs. Adicionalmente, para no falsear los resultados, entre ataque y ataque se reiniciaba la aplicación.

Aumento del número de procesos

La primera prueba consiste en aumentar el número de procesos del sistema. Para ello se ejecutó el siguiente script de Python:

```
import os
pid = os.fork()
if (pid > 0):
    print(pid)
    for i in range (0,5):
        os.fork()
while True:
    pass
```

Tras ejecutar este script el número de procesos se incrementó al igual que la carga y el uso de la CPU. El modelo Isolation Forest detectó la anomalía y envió la notificación (imagen 10).

Como se aprecia en la imagen 10, el score anomaly a la hora de generar la notificación tenía un valor menor que -0,7.

Aumento de la carga de la CPU

La siguiente prueba consiste en aumentar la carga de la CPU. Para ello se ha implementado un sencillo programa en C que comprueba si un número es primo mediante un bucle en el que para cada número 'n', realiza la operación 'n' módulo 'm', donde 'm' toma valores entre 2 y 'n' - 1.

```
#include <stdio.h>
#include <time.h>

void main() {
```

[48] Symobai ALERT!

☆

Transformed Data:

timestamp	num_process	num_fds	num_conn	num_ssh	num_active_users	cpu_usage
0	46289.0	198.0	42354.0	2.0	3.781510e-14	1.0
437.5	87.4	3672.0	6385.0	0.0		100.0

Error:

timestamp	num_process	num_fds	num_conn	num_ssh	num_active_users	cpu_usage
0	46289.0	198.0	42354.0	2.0	3.781510e-14	1.0
437.5	87.4	3672.0	6385.0	0.0		100.0

Accumulated Error:
3.2001155411212415e-13

Anomaly Result:
False

*** ISOLATION FOREST PREDICTION REPORT ***

Input Data:

timestamp	num_process	num_fds	num_conn	num_ssh	num_active_users	cpu_usage
0	46289	198	42354	2	0	1
437.5	87.4	3672	6385	0		100.0

Anomaly Score for the input data:
[-0.71553146]

Anomaly Result:
True

103.81 KB 1 file attached

Download all

log0.txt (103.81 KB)

Download

Figura 10: Notificación de anomalía al incrementar el número de procesos

```

time_t start, end;

int i, num = 1, primes = 0;

double elapsed_time = 0;

time(&start);
while (num <= 10000000) {
    i = 2;
    while (i <= num) {
        if(num % i == 0)
            break;
        i++;
    }
    if (i == num)
    {

```



```

        primes++;

        printf("%d prime numbers calculated\n",primes);
    }

    num++;
}

time(&end);

elapsed_time = difftime (end,start);
}

```

Tras ejecutar el programa anterior, el sistema indica que ha detectado la anomalía, y en el log indica que el modelo que la ha detectado ha sido el Isolation Forest, tal y como se muestra en la imagen 11. En este caso se detecta la anomalía transcurridos algunos minutos. Esto seguramente se deba a que la variable que ha dado lugar a la anomalía ha sido la carga de la cpu, y dicha variable se calcula haciendo una media de los últimos 15 minutos. Al igual que en el caso anterior, el score anomaly tenía un valor menor que -0,7.

[46] Symobai ALERT!
☆

Transformed Data:

timestamp	num_process	num_fds	num_conn	num_ssh	num_active_users	cpu_usage
0	45291.0	167.0	43011.0	2.0	7.365203e-15	1.0
50.0	86.9	40014.0	10966.0	0.0		60.9

Error:

timestamp	num_process	num_fds	num_conn	num_ssh	num_active_users
0	8.194178e-14	3.391568e-15	1.675423e-15	2.220446e-14	0.0
0.0	9.640644e-15	3.613116e-16	5.807387e-15	9.913194e-15	NaN

Accumulated Error:
0.7500000000001249

Anomaly Result:
False

*** ISOLATION FOREST PREDICTION REPORT ***

Input Data:

timestamp	num_process	num_fds	num_conn	num_ssh	num_active_users	cpu_usage
0	45291	167	43011	2	0	2
50.0	86.9	40014	10966	0		60.9

Anomaly Score for the input data:
[-0.70584207]

Anomaly Result:
True

100.04 KB 1 file attached
[Download all](#)

log0.txt (100.04 KB)

Figura 11: Notificación debido al aumento de la carga de la CPU

Aumento del uso de la memoria RAM

Otra prueba interesante es aumentar el uso de la RAM. En este caso, para estresar el sistema se ha utilizado la utilidad “stress-ng”. Para ello se ha ejecutado el siguiente comando:

```
$ stress-ng --vm-bytes $(awk '/MemAvailable/{printf "%d\n", $2 * 0.9;}') < /proc/meminfo)k --vm-keep -m 5
```

Ante este ataque la aplicación no detectó la anomalía tras más de una hora de monitorización. A continuación se incrementó la intensidad del ataque con el siguiente comando, pero tampoco fue detectado por la aplicación en un periodo de tiempo razonable.

```
$ stress-ng --vm-bytes $(awk '/MemAvailable/{printf "%d\n", $2 * 0.9;}') < /proc/meminfo)k --vm-keep -m 10
```

En ambos casos, el score anomaly de Isolation Forest estaba cercano al -0.7 que marca el umbral para detectar la anomalía, pero no le llegó a sobrepasar.

Incremento del número de conexiones TCP/UDP

Para simular un incremento anómalo del número de conexiones TCP/UDP, al igual que en el caso anterior se va a utilizar la herramienta “stress-ng”, ejecutando para ello el siguiente comando:

```
stress-ng -m 1 --sock 10
```

En este caso, la aplicación detectó también la anomalía de nuevo mediante el algoritmo de Isolation Forest, tal y como se muestra en la imagen 12.

Conclusiones de la prueba de concepto

Las conclusiones de las pruebas de concepto llevadas a cabo son las siguientes:

- De todas las pruebas realizadas, solo en un caso no se ha detectado la anomalía. No obstante, en dicha el score anomaly estaba cercano al umbral de forma continua. Este resultado abre la opción a utilizar la evolución del score anomaly como detector de

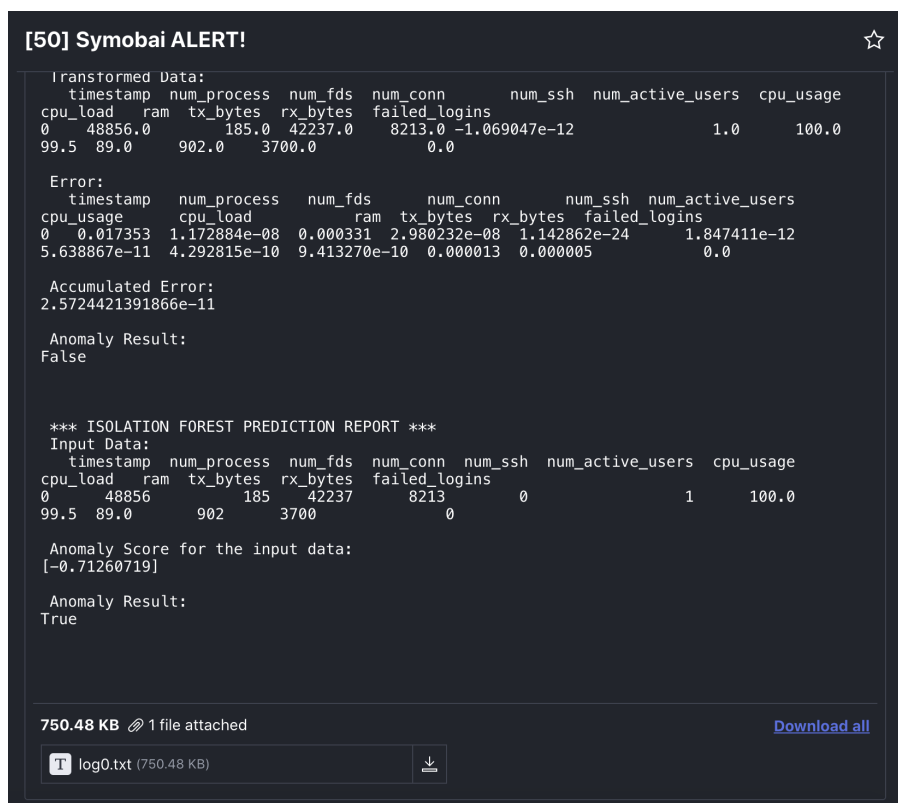


Figura 12: Notificación de anomalía debido al número de conexiones

eventos, y no únicamente la salida del modelo. Esta aproximación requeriría un análisis profundo del valor del score anomaly en diferentes situaciones, pero permitiría aumentar la sensibilidad del modelo.

- Analizando los datos de entrada que han generado las notificaciones, en todos los casos además de incrementarse el valor de la métrica que se intentaba estresar, se incrementaba también la carga y el uso de la CPU de forma colateral, lo cual también sería normal en un sistema real. No se puede concluir que un incremento aislado en alguna de las métricas sea detectado por los modelos, aunque sería una prueba fácilmente realizable sin necesidad de una máquina real, ya que se podría hacer generando el modelo offline. La carga de la cpu se ha mostrado como la métrica que más influencia tenía en el algoritmo de Isolation Forest.
- Como efecto colateral del punto anterior, en ninguno de los casos la respuesta del sistema ha sido inmediata. La anomalía se detectaba debido a la desviación de más de una métrica,

y era necesario cierto tiempo para que los valores evolucionaran. Según cómo de agresivo sea el ataque, el tiempo de respuesta era mayor o menor. En cualquier caso, en todas las situaciones la herramienta generó la alerta antes de que el sistema estuviera saturado. Observando el log de la herramienta se puede observar que el score anomaly del algoritmo Isolation Forest va incrementándose poco a poco.

- El método de detección de anomalías mediante PCA no ha dado el resultado esperado, ya que en ningún caso ha detectado la anomalía. No solo eso, si no que el error de reconstrucción en todas las situaciones ha sido bajo. Para tener un mejor resultado del modelo PCA, posiblemente sea necesario reducir el número de componentes.

Siguientes pasos

A lo largo de este proyecto se ha desarrollado una prueba de concepto de una herramienta de detección de anomalías en sistemas Linux basada en modelar el sistema mediante técnicas de machine learning. Si bien la prueba de concepto ha mostrado que esta aproximación puede ser útil a la hora de detectar anomalías derivadas de actividad maliciosa, la herramienta está lejos de poder ser llevada a un entorno real de producción. En este apartado se listan algunas de las líneas de desarrollo para seguir construyendo la aplicación.

- En primer lugar sería necesario realizar un análisis con diferentes tipos de malware para obtener los parámetros del sistema más afectados ante diferentes ataques. Si bien los algoritmos de machine learning se basan en el aprendizaje automático, el éxito de su resultado radica en la correcta elección de los parámetros de entrada mediante los cuales se modela el sistema. En este caso se han seleccionado métricas típicas, pero un malware suficientemente sofisticado podría pasar inadvertido.
- Obtener datos etiquetados como entrada para modelar el sistema, realizando una monitorización de sistemas infectados. Esto permitiría por un lado realizar un mejor ajuste de los modelos utilizados en el proyecto. Por ejemplo, se podrían elegir seleccionar componentes específicas del PCA, o ajustar los niveles de probabilidad de error. Por otro lado, tener datos etiquetados permitiría también utilizar otros modelos supervisados.
- En este caso tanto la recogida de las métricas, como el entrenamiento de los modelos, como la detección de las anomalías se ha realizado en la máquina que se quería monitorizar. Otra alternativa podría ser tener un servidor dedicado para realizar el modelado, de manera que la máquina a monitorizar únicamente se encargaría de obtener las métricas y enviarlas.

Esto permitiría tener una mayor capacidad de cómputo, e implantar la herramienta en sistemas con poca capacidad de procesamiento, como por ejemplo sistemas embebidos.

- Adquisición de datos distribuida. Existen arquitecturas que se componen de varios nodos con comportamientos similares. En ese caso, se podrían crear modelos con los datos agregados de los diferentes nodos.
- En la línea anterior, nos encontramos escenarios en los que hay diferentes instancias de un mismo elemento. Por ejemplo en un entorno industrial en el que haya varias líneas de producción idénticas donde cada línea cuenta con, por ejemplo, un robot, tendríamos 'n' robots haciendo las mismas tareas. Se podría asumir que al ser el mismo robot haciendo lo mismo el modelo del sistema debería ser el mismo. En este escenario podríamos comparar los modelos generados por las diferentes instancias como método de detección de anomalías. En el entorno actual en el que cada vez hay más productos conectados, una compañía podría plantearse obtener las métricas de sus productos para generar dichos modelos y ofrecer un servicio de detección de anomalías como valor añadido.
- Un paso más allá sería una vez detectada la anomalía, tener otro servicio que realizara un análisis del sistema para intentar detectar el proceso causante de la anomalía. De esta manera se podría dotar a la herramienta incluso de capacidad de actuación frente a eventos.
- Otra línea de desarrollo es utilizar redes neuronales en lugar de algoritmos de machine learning.
- Desde el punto de vista del desarrollo de la aplicación, como se ha comentado en algunos puntos anteriormente hay ciertos parámetros que se han establecido y que están hard-codeados. Si bien su modificación es muy sencilla, se deberían establecer una serie de ficheros de configuración a modificar por el usuario que incluyeran todos estos parámetros. Siguiendo el mismo concepto, se podría implementar un mecanismo mediante el cual la clase "Model" obtuviera los modelos a utilizar desde un fichero de configuración. De esta manera, el usuario podría añadir nuevas clases con nuevos modelos sin necesidad de modificar la herramienta, siempre que incluyeran los métodos definidos.

Conclusiones

Como conclusión principal de este trabajo fin de máster se puede extraer que la implementación e implantación de un sistema de detección de malware basado en el modelado del sistema mediante técnicas de machine learning puede ayudar a detectar actividad maliciosa en nuestro sistema. No obstante, un malware no deja de ser una aplicación software que convive con el resto de programas ejecutándose en la máquina, por lo tanto, la efectividad del sistema está supeditada al propio funcionamiento del malware.

Por otro lado, el modelado del sistema permite detectar comportamientos anómalos, pero el hecho de que un comportamiento sea anómalo no implica per se que se deba a actividad maliciosa. Por esta razón se debe ser cuidadoso a la hora de interpretar los eventos generados por el sistema. A la hora de desplegar una herramienta de este tipo será de vital importancia ajustar correctamente los modelos para que no generen falsos positivos.

En cuanto a los modelos utilizados, el algoritmo que mejores resultados a dado según las pruebas realizadas ha sido el Isolation Forest. Pese a ello, es interesante tener el resultado de diferentes modelos, ya que como se ha visto, no en todas las situaciones el modelos de Isolation Forest ha detectado la anomalía. El algoritmo GMM requiere de datos con mayor variabilidad que los utilizados en este proyecto, mientras que la detección de anomalías mediante PCA requeriría de un ajuste más preciso.

En definitiva, un sistema de este tipo puede ser un complemento interesante a antivirus y EDRs, ya que puede llegar a detectar actividad maliciosa antes que estos sistemas al no depender de IOCs, pero no puede plantearse como un sustituto.

Bibliografía

- [1] Kaspersky. ¿En qué consiste el análisis heurístico?
<https://www.kaspersky.es/resource-center/definitions/heuristic-analysis>
- [2] INCIBE. ¿Qué son y para qué sirven los SIEM, IDS e IPS?
<https://www.incibe.es/protege-tu-empresa/blog/son-y-sirven-los-siem-ids-e-ipss>
- [3] Bayesian information criterion
<https://www.immagic.com/eLibrary/ARCHIVES/GENERAL/WIKIPEDI/W120607B.pdf>
- [4] Principal Component Analysis
<https://www.cienciadedatos.net/documentos/py19-pca-python.html>
- [5] Gaussian Mixture Model
<https://www.cienciadedatos.net/documentos/py23-deteccion-anomalias-gmm-python.html>
- [6] Isolation Forest
<https://www.cienciadedatos.net/documentos/py22-deteccion-anomalias-isolation-forest-python.html>