Proyecto Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

redBorder Malware EndPoint

Autor: Adrián Rodríguez García
Tutor: Pablo Nebrera Herrera

Dep. Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2016

Proyecto Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

redBorder Malware EndPoint

Autor:
Adrián Rodríguez García

Tutor:
Pablo Nebrera Herrera
Profesor titular

Dep. Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2016

Proyecto Fin de Grado: redBorder Malware EndPoint

Autor: Adrián Rodríguez García

Tutor: Pablo Nebrera Herrera

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

RedBorder Malware EndPoint ha sido mi primer contacto con el mundo laboral. Estoy satisfecho con esta experiencia, que me ha ayudado a aprender y a superarme día a día.

En primer lugar, quisiera agradecer a mi padre el sacrificio realizado para poner a mi disposición todos los medios necesarios para estudiar, que me han permitido llegar hasta aquí.

Agradecer también a todo el equipo de redBorder. En especial a Carlos Jiménez, por el trato recibido por su parte, por su dedicación y por su esfuerzo para que este proyecto sea a día de hoy una realidad.

A Verónica, por hacer todo el trayecto universitario conmigo y hacerlo mucho más fácil y llevadero, por haber estado en los buenos y malos momentos y por ayudarme cada vez que lo he necesitado.

Por último, quiero dar las gracias a mis amigos, los que durante toda esta etapa me han tenido que soportar tanto dentro como fuera de la universidad. Por todos ellos, ¡¡Venga a ustedes!!

Adrián Rodríguez García

Sevilla, 2016

Resumen

Los cibercriminales son una de las principales amenazas a las que se enfrentan las empresas día tras día en el ciberespacio. Incluso compañías con directrices estrictas en materia de ciberseguridad pueden quedar expuestas. Por este motivo, es necesario implantar medidas de seguridad en todos los rincones de las empresas, incluido los equipos de trabajo, herramienta tan necesaria como vulnerable a ciberataques.

De esta conclusión nace redBorder Malware EndPoint, incluido dentro del módulo de Malware del portfolio de productos de redBorder.

redBorder Malware EndPoint es un sistema de ciberseguridad para equipos finales, con inteligencia artificial, que detecta y evita, en tiempo real, amenazas, malware y ataques dirigidos a empresas. Trabaja en cooperación con la aplicación de redBorder Malware para reportar y bloquear malware en base a un sistema de reputación.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN.

"En Internet hay lobos y corderos. Si no eres lobo, te toca ser cordero."

- Chema Alonso –

1.1 Motivación

Hoy en día, la seguridad cibernética es una de las mayores preocupaciones en el sector empresarial, debido a los continuos ataques contra compañías y organizaciones para el propósito de obtener información. Este tipo de situaciones ocurren cada vez con mayor frecuencia e incluso existen organizaciones de cibercriminales que se benefician de estas actividades.

Por este motivo, es necesario tener una herramienta anti-malware que además de incluir el análisis del tráfico de red (NAT) y herramientas de seguridad cibernética específicos (IPS, IDS, Mail Gateway, etc.), incluya protección para EndPoints[1]. Los EndPoints son pilares fundamentales en la detección de amenazas avanzadas, malware o ataques dirigidos y mantener, gracias a la inteligencia artificial, toda la infraestructura de red segura.

Este proyecto, cubrirá la necesidad de protección para EndPoints y se integrará dentro de redBorder Malware, donde convivirá con los demás sistemas que este alberga.

1.2 Objetivo

El propósito general de este proyecto es el desarrollo de un sistema de ciberseguridad para equipos finales que detecte en tiempo real amenazas avanzadas, malware, ataques dirigidos, y que reporte dicha información al sistema central de redBorder Malware para practicar un análisis, y permitir de este modo tomar decisiones para resolver cualquier incidencia.

Una vez realizado el desarrollo y la integración del proyecto dentro de la plataforma de redBorder Malware, se realizarán pruebas de concepto y de rendimiento para demostrar el óptimo funcionamiento del mismo como herramienta anti-malware.

1.3 Metodología

Antes de la realización del proyecto se realizará una introducción a redBorder Malware y todos los elementos que intervienen dentro del sistema. Adicionalmente se hablará de Apache Kafka, Riak KV y de Google Rapid Response. Todo ello es tratado en la sección 2. A continuación, se introducirá el sistema en la sección 3 y se explicará la arquitectura del EndPoint y de todos los módulos que lo componen en la sección 4. Seguidamente se comenzará con el diseño, implementación y desarrollo, entrando en detalle en cada uno de los módulos que componen el EndPoint y sus funcionalidades. Todo ello, explicado en la sección 4. Una vez desarrollado, en la sección 5, se integrará dentro de la aplicación de malware de redBorder. Por último, en la sección 6, se realizarán diferentes pruebas de concepto, rendimiento y fiabilidad para demostrar el óptimo funcionamiento del sistema, finalizando con la sección 7 de presupuesto.

[1] Equipos finales de una red, haciendo referencia a puestos de trabajo en ámbito empresarial.

2 CONTEXTUALIZACIÓN.

2.1 Introducción a redBorder Malware

redBorder es un sistema de seguridad y monitorización de redes en tiempo real y altamente escalable que permite disponer de una gran visibilidad de todo lo que ocurre en las redes monitorizadas, procurando un gran control de la infraestructura y de los servicios ofrecidos en ésta.

Este sistema ofrece varios módulos de servicios que pueden ser contratados por los clientes. Entre ellos se encuentra redBorder Malware, que es un sistema de ciberseguridad para detectar malware y evitar la propagación del mismo y cualquier daño que pueda causar en la red.

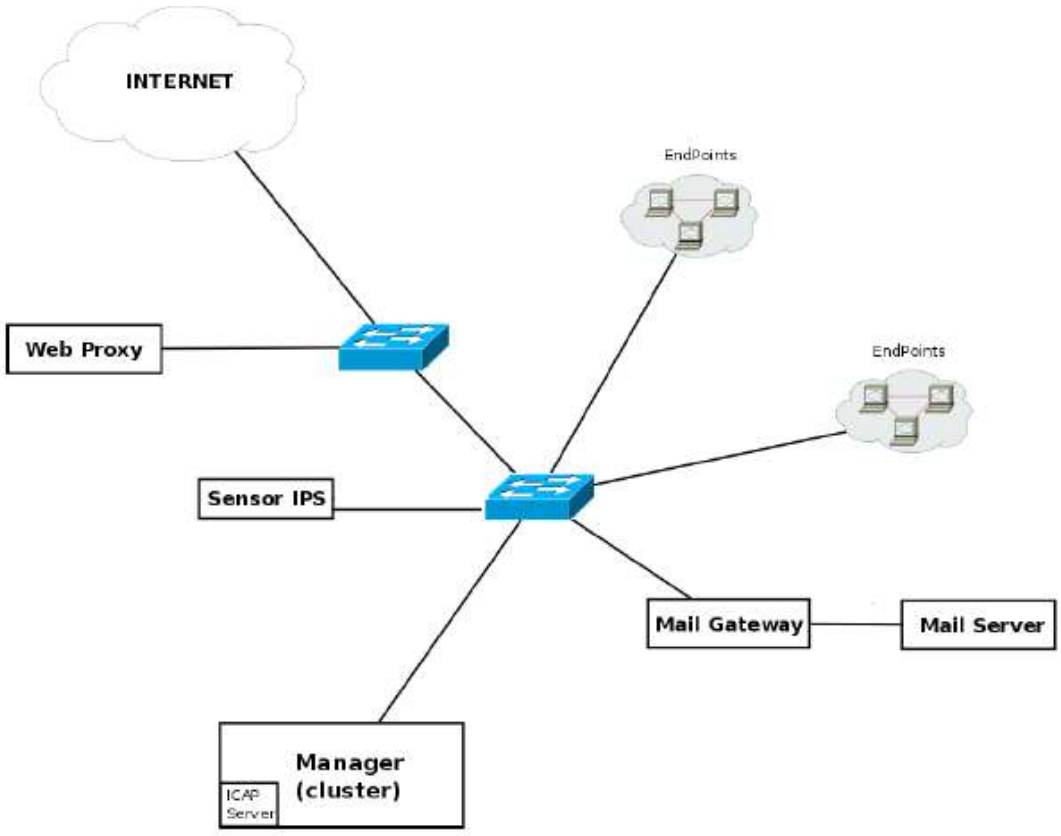


Figura 1 Arquitectura de redBorder Malware

Tal y como se muestra en la anterior imagen, el escenario está formado por una red conmutada con 5 elementos principales:

- **Manager (clúster):** elemento central en el cual se recibe, se trata y se visualiza toda la información emitida por los diferentes elementos. En su interior se incluye el servidor ICAP, que recibe las consultas/peticiones del proxy web.

- **IPS:** elemento por el cual pasa el tráfico de red que se va a analizar y, que funcionando como un IDS/IPS, es capaz de detectar patrones de tráfico correspondientes a posibles ataques y/o Malware. Una vez registrado con un manager, le envía los resultados, así como los ficheros detectados mediante los protocolos HTTP, FTP, SMTP y SMB.

- **Mail Gateway:** elemento por el cual pasa el tráfico de correo (SMTP). Funciona como un sencillo MTA[2], ya que cada vez que recibe un email lo reenvía, por una parte, al servidor de correo correspondiente y, por la otra, una copia de los emails con adjuntos al manager, junto con datos adicionales del email (ej. Dirección remitente/destinatario...).

- **Mail Server:** servidor de correo electrónico utilizado únicamente para la recepción del correo interceptado por el Mail Gateway y autorizado para su reenvío.

- **EndPoints:** elementos finales de la red, donde se instalará este proyecto con el fin de detectar en tiempo real amenazas, malware y ataques a través del análisis de información del sistema, reportando la misma al manager.

Cabe mencionar que en uno de los puentes de red se ha configurado un SPAN[3] a nivel local para habilitar la interfaz de red del puente que lo une con el puente más externo para copiar todo el tráfico hacia/desde Internet en otra interfaz. En esa interfaz de destino es donde se conecta la interfaz del IDS/IPS.

Destacar que redBorder implementa más servicios de los mencionados aquí, pero se ha optado por introducir sólo redBorder Malware para comprender mejor el ecosistema y funcionamiento de este proyecto.

2.2 Tecnologías utilizadas

2.2.1 Apache Kafka

Servicio distribuido, particionado y con posibilidad de replicación que ofrece la funcionalidad de un sistema de colas de mensajes con el paradigma publicador/subscritor. Los eventos generados por los distintos elementos de redBorder Malware pasan por este servicio en el Manager, a fin de desacoplar la obtención de datos de su procesamiento.
El EndPoint usa Apache Kafka para enviar información relativa a eventos del sistema al Manager y así informar en tiempo real de lo sucedido en los equipos finales de la red.

2.2.2 Google Rapid Response (GRR)

GRR es un framework para dar una respuesta rápida remota a incidente forenses. La estructura es cliente/servidor, donde el servidor se encuentra en el Manager y el cliente en los EndPoints. De esta forma, dentro de redBorder Malware se usa para parar/reiniciar redBorder Malware EndPoint, bloquear los equipos en cuanto se detecte algún tipo de riesgo para red y obtener información de un sistema final para su análisis forense.

[2] Mail Transfer Agent (Agente de Transferencia de correo) se encarga de recibir y reenviar correos.
[3] Modo de configuración de una interfaz de red en un puente de red que permite enviar copias de paquetes a otra interfaz de red que esta siendo monitorizada.

2.2.3 Riak

Servicio de almacenamiento de objetos fiable y de alta disponibilidad. Es utilizado sobre todo para el almacenamiento profundo en Druid. Todos los ficheros que se envían desde los EndPoints para ser analizados en busca de malware, usan el servicio de Riak para almacenar en Druid.

2.2.4 Druid

Es un sistema de almacenamiento y análisis de información que permite la consulta y exploración de grandes cantidades de datos en tiempo real.

2.2.5 Windows Manager Instrumentation (WMI)

Windows Management Instrumentation (WMI) es la infraestructura de datos y operaciones de gestión en sistemas operativos basados en Windows. Permite automatizar tareas administrativas y suministrar datos de gestión a otras partes del sistema operativo y productos.

En este proyecto se ha usado para gestionar en tiempo real los cambios producidos en el sistema de fichero de Windows y para poder gestionar los cambios en los procesos.

2.2.6 Python

Lenguaje de programación interpretado usado para la realización de este proyecto. La elección de este lenguaje es debido a la enorme potencia que contienen sus librerías para el sistema operativo Windows.

3 DESCRIPCIÓN DEL SISTEMA.

3.1 Introducción a redBorder Malware EndPoint.

redBorder Malware EndPoint es un sistema de ciberseguridad, para equipos finales de una red, con inteligencia artificial que detecta en tiempo real amenazas, malware o ataques. Una vez detectada cualquier anomalía o amenaza se reporta a la unidad central de la plataforma redBorder Malware (manager), con capacidad de análisis, para estudiar y posteriormente determinar la acción a ejercer sobre un EndPoint.

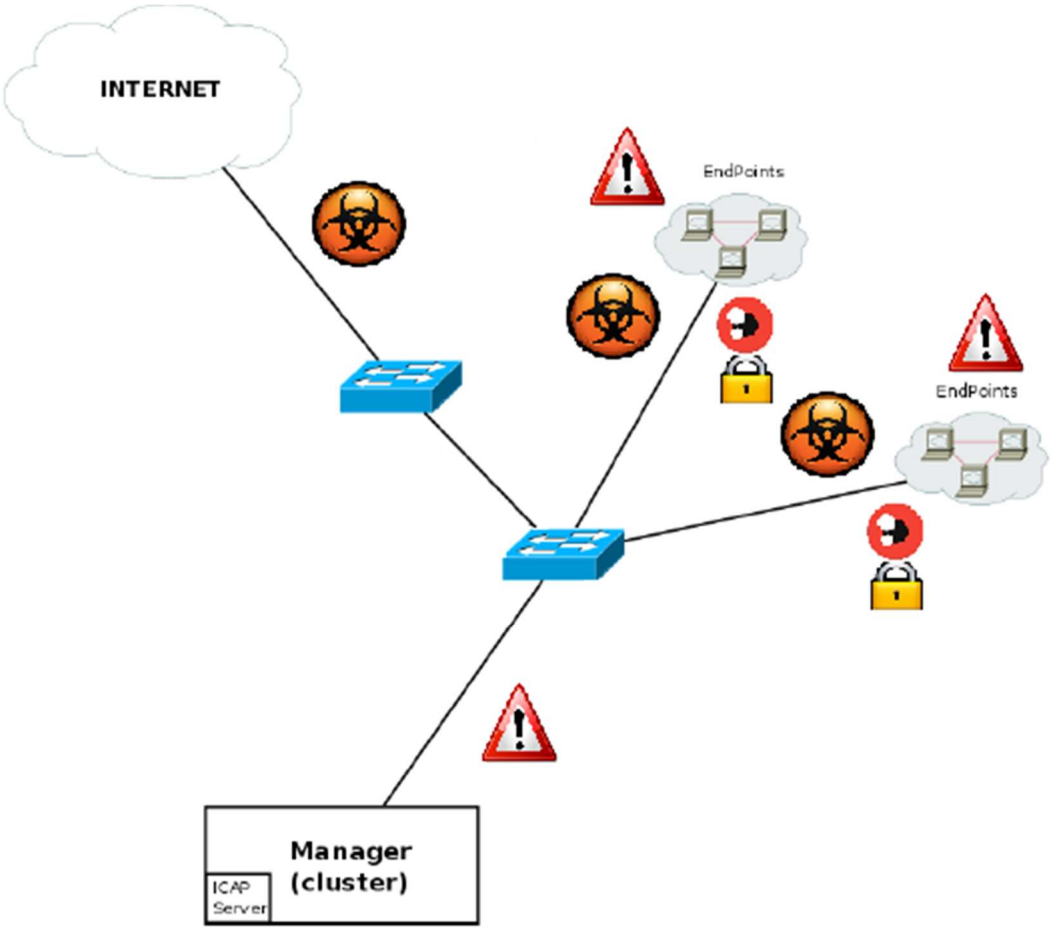


Figura 2 Detección de anomalías

Este sistema, consigue un alto nivel de seguridad en EndPoints debido a que monitoriza de forma constante los procesos que se crean o destruyen, la red y cualquier cambio en el sistema de ficheros. La información obtenida se envía al manager para un análisis mediante IOC[4] a la vez que se almacena de forma local, creando así un histórico de eventos para un posible análisis forense. Adicionalmente, cualquier cambio en el sistema de ficheros es detectado y el fichero implicado subido a Riak para analizarlo mediante cargadores[5] en busca de

[4] Indicadores de compromisos (Indicators Of Compromises) son un conjunto de reglas que permiten detectar incidencias o anomalías.
[5] Conjunto de motores de análisis malware que emiten una valoración sobre una posible infección malware en un fichero.

posibles infecciones malware.

Este sistema ha sido instalado en EndPoints con sistemas operativos basados en Windows como un servicio mediante un ejecutable MSEXE. Por tanto, se inicia y apaga con el equipo y funciona con permisos de superusuario SYSTEM.

3.2 Funcionalidades principales

Las principales funcionalidades que este proyecto presenta son las siguientes:

- **Captura de procesos en tiempo real**: Los procesos son la ejecución de programas existentes en la memoria de un equipo. Por ello, es esencial detectar la creación y destrucción de los mismos, debido a que, si en nuestro equipo se introduce un malware, es de vital importancia detectar su ejecución cuanto antes para evitar o mitigar cualquier posible daño que pueda causar.

  La captura de procesos creados o destruidos se realiza mediante WMI, que además de permitir la captura en tiempo real, permite obtener datos del proceso.

Figura 3 Detección de procesos

- **Detección en tiempo real de modificaciones en el sistema de ficheros**: Los ficheros son la principal fuente de malware. Por este motivo, cualquier fichero que se cree, modifique o borre en el sistema de ficheros, será detectado, capturado y comparado contra una caché local para comprobar si es conocido o no. En caso de ser desconocido, se subirá mediante Riak al manager para ser analizado en busca de malware. De esta forma, se genera un control exhaustivo de todos los ficheros que circulan por la red.

  Como ocurre con los procesos, WMI permite detectar los cambios producidos en el sistema de ficheros y de forma adicional muestra información acerca del evento producido.

Figura 4 Captura de ficheros

- **Detección de las conexiones de red en tiempo real**: Monitorizar las conexiones de red es esencial para detectar ataques o amenazas y poder neutralizarlas. Es por este motivo, por el cual este sistema monitoriza todos los procesos y detecta cualquier conexión que realicen. De esta manera, se conocen situaciones comprometida en tiempo real y se puede actuar en consecuencia de forma rápida y contundente.

  Como se explicó anteriormente, WMI permite capturar en tiempo real la creación de procesos y adicionalmente ofrece información de los mismos. Entre esa información, se encuentran las conexiones de red que ese proceso tiene abiertas.

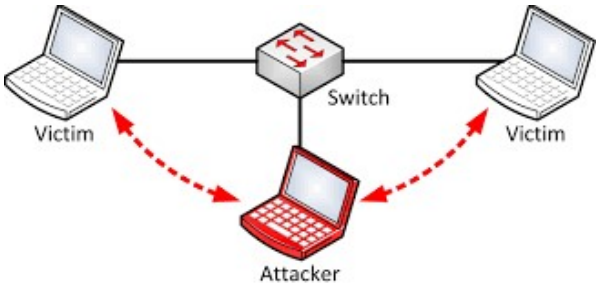


Figura 5 Detección de conexiones

- **Generación de un histórico de eventos:** En el entorno del sistema (sección 4.1.1) se crea un histórico que recoge en formato JSON todos los eventos generados a partir de las funcionalidades descritas anteriormente. Este registro se actualiza cada vez que se detecta un evento.

  En caso de infección o ataque en un EndPoint, este histórico es esencial para poder realizar un análisis forense y detectar las causas exactas por las que ocurrieron y poder llegar a la fuente primaria de propagación.

Figura 6 Histórico de eventos

4 ARQUITECTURA DEL SISTEMA.

4.1 Arquitectura del entorno

El servicio EndPoint de redBorder Malware consta de una arquitectura de entorno subdividida en dos partes muy claramente diferenciadas:

- **RedBorder Malware EndPoint**: Objeto de este proyecto, monitoriza los procesos, sistema de ficheros, web y enriquece la caché local mediante la información que se obtiene del servicio de reputación.

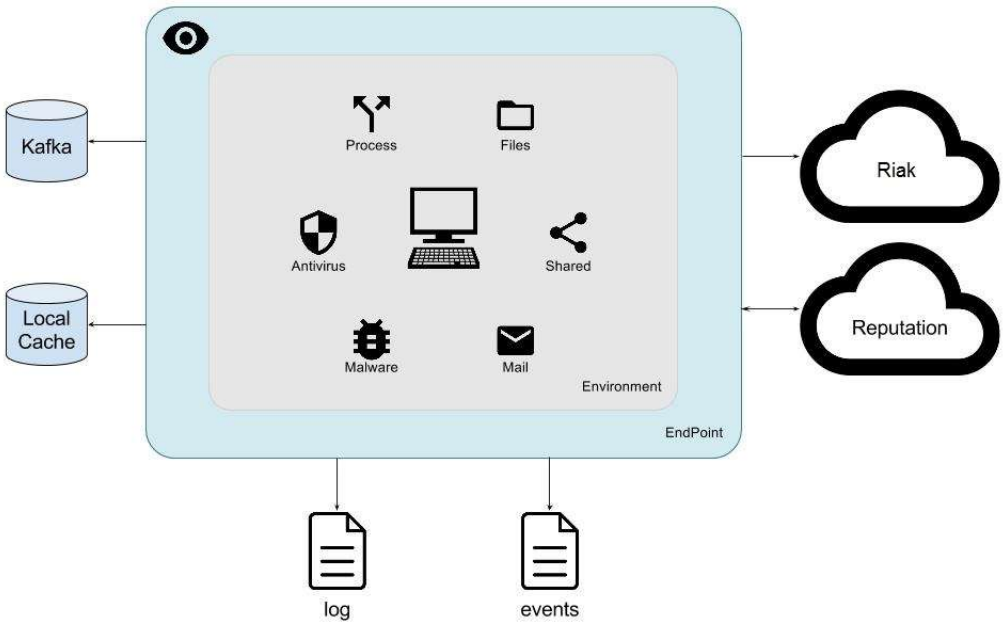


Figura 7 Arquitectura del entorno del agente EndPoint

- **GRR**: Consta de una arquitectura cliente – servidor, donde el servidor está instalado en el manager de redBorder Malware y el cliente en los EndPoints.

  GRR comunica su servidor y clientes mediante el protocolo de Google protobuf. El intercambio de mensajes siempre es en sentido servidor a cliente, donde el servidor lanza una petición al cliente para obtener cierta información y este último responde con dicha información.

  Este mecanismo, permite al manager obtener información sobre cualquier EndPoint e interactuar con redBorder Malware EndPoint (parar/reiniciar). Además, es un método esencial para ejercer acciones de contención frente a cualquier amenaza que pudiera surgir.

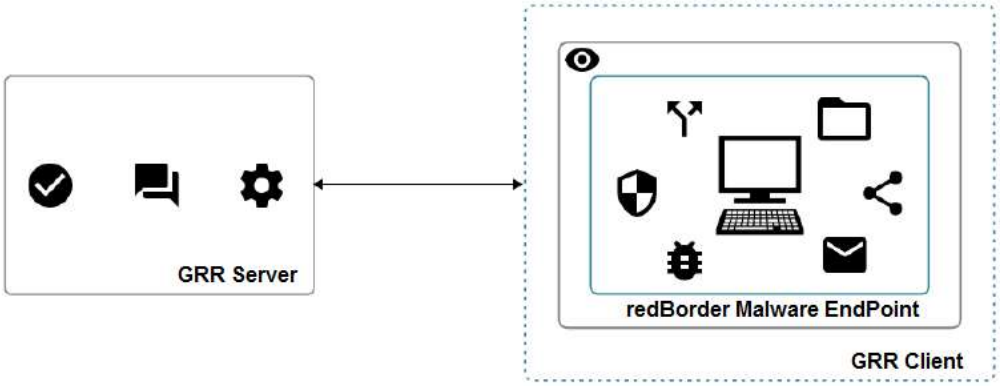


Figura 8 Arquitectura del entorno

4.1.1 Arquitectura de entorno de redBorder Malware EndPoint

El entorno, una vez desplegado, se encuentra en la ubicación "*%PROGRAMFILES%\redBorder Malware EndPoint*" y se compone de un total de cinco directorios:

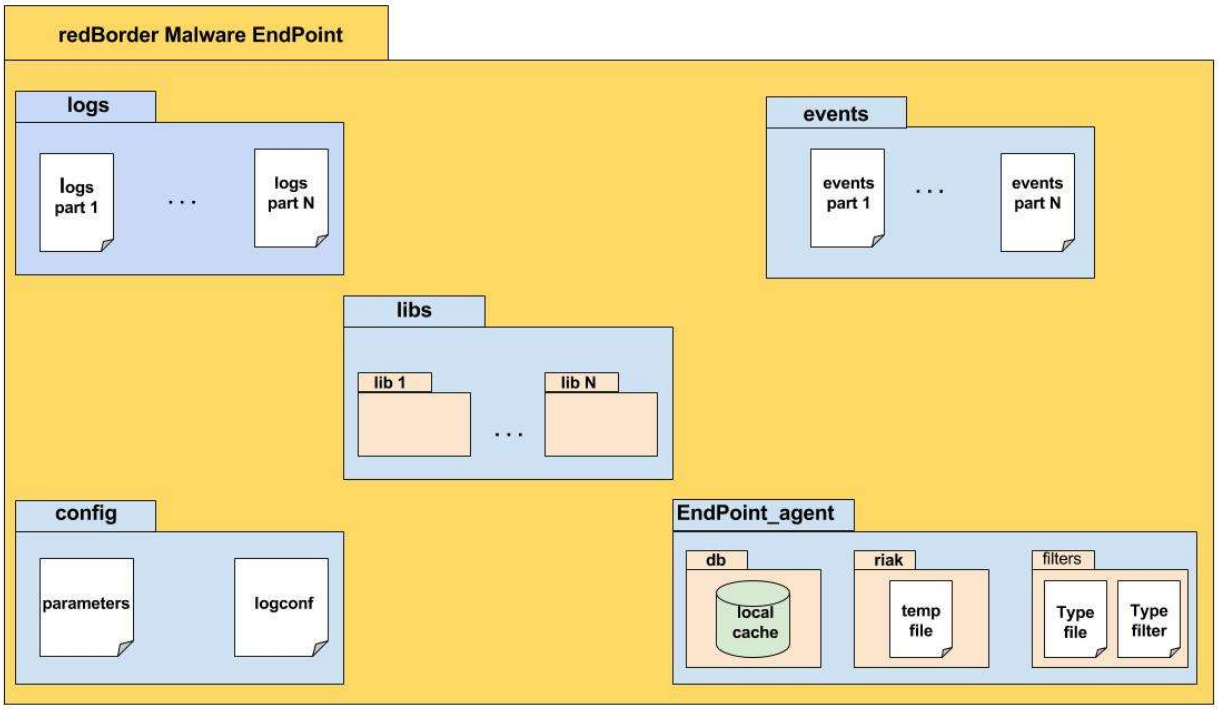


Figura 9 Directorios del entorno

- **Libs**: Se compone de un conjunto de librerías que se usan en el agente EndPoint y que están preparadas para ser cargadas. Son imprescindibles para el correcto funcionamiento del software. Las librerías que componen este directorio son las siguientes:
    - **Pywin32**: Es una extensión que permite usar las funcionalidades de varias API de Windows, entre ellas WMI.
    - **Psutil**: Librería que permite obtener información de los procesos activos y de la utilización del sistema (CPU, memoria, disco, red, etc.)

- **Kafka-Python**: Librería que permite la utilización de Apache Kafka para el envío y recepción de eventos usando sistemas distribuidos.
- **Boto**: Librería que se usa para subir ficheros a Riak desde un EndPoint.
- **Request**: Librería que incluye urllib3 y que ofrece todas las facilidades para usar servicios web de forma totalmente transparente.
- **PyYaml**: Librería que permite parsear formatos YAML en este lenguaje de programación.
- **Six**: Librería que permite usar funcionalidades de Python y suavizar su diferencia entre diferentes versiones.

- **Config**: En este directorio se encuentran los dos ficheros de configuración que existen para configurar el agente EndPoint.
  - **Logconf**: Fichero que se usa para configurar, por un lado, los logs de aplicación y por otro, el histórico de eventos generados en las diferentes funcionalidades del proyecto.

```
[loggers]
keys=root, binnacle, application

[handlers]
keys=streamHandler, fileHandlerBinnacle, fileHandlerApplication

[formatters]
keys=preciseFormatter, simpleFormatter

[logger_root]
level=INFO
handlers=streamHandler
qualname=root


[logger_binnacle]
handlers=fileHandlerBinnacle
qualname=binnacle

[logger_application]
handlers=fileHandlerApplication
qualname=application


[handler_fileHandlerApplication]
class=handlers.RotatingFileHandler
level=INFO
formatter=preciseFormatter
args=(%(application)s, 'a', 52428800, 5)

[handler_streamHandler]
class=StreamHandler
level=INFO
formatter=preciseFormatter
args=()

[handler_fileHandlerBinnacle]
class=handlers.RotatingFileHandler
level=INFO
formatter=simpleFormatter
args=(%(binnacle)s, 'a', 52428800, 10)


[formatter_preciseFormatter]
format=[%(levelname)s] : %(asctime)s - %(filename)s : %(lineno)
d -> %(message)s

[formatter_simpleFormatter]
format=%(message)s
```

Figura 10 Fichero logconf

Como se puede observar en la anterior imagen, existen los logs de salida para el histórico de eventos (binnacle) y para los logs de aplicación (application). Adicionalmente, se observa como cada logs lleva asignado su manejador (handler), en los cuales se indica el tipo de logs (INFO, DEBUG o ERROR), el tipo de formato y la clase (class). Destacar, que la clase elegida ha sido la rotativa o cíclica, es decir, que completado cierto tamaño de fichero de logs (50 Mb en este caso), se cree otro fichero de logs para continuar hasta un máximo de ficheros (5 en caso de los logs de aplicación y 10 en caso del histórico de eventos). Una vez completado el máximo de ficheros permitidos, se empezará a sobreescribir el primero.

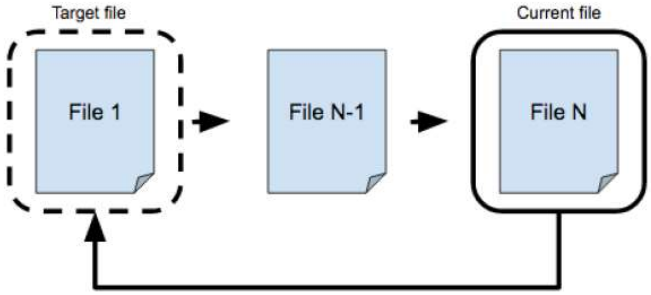


Figura 11 Ficheros cíclicos o rotativos

- **Parameters**: Fichero de configuración en formato YAML que contiene todos los parámetros necesarios para poder configurar de forma correcta el agente EndPoint y que de esta forma pueda comunicarse con el manager de redBorder Manager mediante el sistema de reputación (sección 4.2.1), Riak o Apache Kafka.

```
---
kafka_server: kafka.redborder.cluster:9092
s3_bucket: malware
s3_key: "/mdata/input/"
access_key: <ACCESS_KEY>
secret_key: <SECRET_KEY>
reputation_server: rb-reputation.redborder.cluster
file_size: 134217728
version_cache: v1
domain_id: 1
score: 1
client_id: C.91c40668ddea3975
```

Figura 12 Fichero parameters

  - **Kafka_server**: Parámetro que indica un alias del servidor donde enviar los eventos junto al puerto destino. En este caso, la IP correspondiente al alias se encuentra en el fichero "hosts", que corresponde con un balanceador que se encarga de redirigir los eventos según unos criterios al manager adecuado dentro del clúster.
  - **S3_bucket**: Indica el bucket, dentro de Riak instalado en el clúster de manager de redBorder Malware, donde se subirán los ficheros.
  - **S3_key**: Parámetro que indica la ruta dentro del bucket de Riak donde subir los ficheros.
  - **Access_key** y **secret_key**: En su conjunto, son dos parámetros que forman el equivalente a "usuario", "contraseña" para poder acceder a Riak.

- **Reputation_server**: Alias del servidor donde se encuentra el servicio de reputación (sección 4.2.1) del cual se realimenta la caché local. La IP que se corresponde a este alias se encuentra en el fichero "hosts", al igual que las IP de Kafka o Riak.
- **File_size**: Parámetro que indica el tamaño máximo en bytes que un fichero puede tener para subirse a Riak. A partir de ese valor (128 Mb) el fichero no se subirá a la plataforma de malware, debido a que VirusTotal no analiza archivos superiores a ese tamaño, por tanto, no se analizarían de forma correcta.
- **Versión_cache**: Versión del sistema de reputación (sección 4.2.1), necesario para construir la URL de petición.
- **Domain_id**: Identificador de tipo numérico, que identifica de forma unívoca el dominio de red en el cual está un EndPoint.
- **Score**: Parámetro que indica una puntuación a partir de la cual los ficheros serán considerados como malware (se explica más detalladamente en la sección 4.2.1 dentro del sistema de reputación).
- **Client_id**: Identificador únivoco de un EndPoint asignado mediante GRR.

- **Logs**: Directorio en el cual se almacenan los logs de aplicación del agente EndPoint. Como se ha explicado anteriormente, se crearán un máximo de ficheros (cinco en este caso) de un tamaño determinado (50Mb particularmente), a partir de los cuales empezará a sobreescribirse el primero.

  Un ejemplo de logs de aplicación es la siguiente imagen:

```
[INFO] : 2016-06-03 13:22:08,126 - endpoint_agent.py : 167 ->
Created new folder : C:\Program Files\redBorder Malware EndPoint
\endpoint_agent\s3
[INFO] : 2016-06-03 13:22:08,126 - endpoint_agent.py : 191 ->
Created new folder : C:\Program Files\redBorder Malware EndPoint
\endpoint_agent\db
[INFO] : 2016-06-03 13:22:08,126 - Config.py : 29 -> Config
loaded from : C:\Program Files\redBorder Malware EndPoint\config
\parameters.yml
[INFO] : 2016-06-03 13:22:08,126 - FileManager.py : 26 -> Created
new file with key 'type_filter' and mode 'r' in path 'C:\Program
Files\redBorder Malware EndPoint\endpoint_agent\filters
\file_filters.yml'
[INFO] : 2016-06-03 13:22:08,126 - FileManager.py : 61 -> Readed
content from file with key: 'type_filter'
[INFO] : 2016-06-03 13:22:08,126 - CacheManager.py : 65 -> Init
cache client
[INFO] : 2016-06-03 13:22:08,142 - QueueManager.py : 18 ->
Created new queue with key: 'slow'
[INFO] : 2016-06-03 13:22:08,142 - QueueManager.py : 18 ->
Created new queue with key: 'file_event'
[INFO] : 2016-06-03 13:22:08,142 - QueueManager.py : 18 ->
Created new queue with key: 'kafka'
[INFO] : 2016-06-03 13:22:08,142 - QueueManager.py : 18 ->
Created new queue with key: 'file_capture'
[INFO] : 2016-06-03 13:22:08,142 - CacheManager.py : 93 ->
Started cache client!
```

Figura 13 Ejemplo de logs de aplicación

- **Events**: Directorio que almacena el histórico de eventos que el agente EndPoint produce a lo largo del tiempo. Al igual que los logs de aplicación, estos ficheros son cíclicos, creándose un número determinado de ficheros (diez en este caso) con un tamaño determinado (50Mb en particular).

  En la sección 4.3 se detallan y explican todos los eventos producidos en el agente EndPoint.

- **EndPoint_agent**: Son un conjunto de directorios que forman el agente de carga y contienen todo el código de la aplicación. Cabe destacar dos de los directorios:
    - **db**: Directorio donde se almacena la caché local del agente EndPoint. La caché local es una base de datos SQLite con dos campos (hash y score) que se realimenta con el sistema de reputación de redBorder Malware.

      Todos los ficheros detectados y capturados por el agente EndPoint son comparados contra la caché para dilucidar sin son conocidos o no y en caso de ser conocidos si es malware o está limpio. (La caché será tratada más en profundidad en la sección 4.2.1 de este proyecto).
    - **Riak**: Directorio donde se almacena una copia de todos los ficheros que son subidos a Riak. Estos ficheros son temporales, de forma que una vez subidos al manager, se destruyen.
    - **Filter**: Directorio donde se almacenan dos ficheros de configuración internos. Type file, es un fichero que describe las cadenas características de ciertos tipos de ficheros en una serie de bytes, permitiendo así conocer el tipo de un fichero. El otro, Type filter, es un fichero YAML que se configura en el manager y permite elegir que tipos de ficheros no se subirán a Riak para su análisis. Esto es debido porque quizás ficheros de tipo MP3, ODT, DOCX, etc, no son de interés para un determinado cliente.

4.1.2 Servicio de reputación de redBorder Malware

El servicio de reputación es el encargado de proporcionar una API REST para poder hacer consultas de una URL, IP o hash y obtener una puntuación y también pone a disposición listas de reputación para los elementos de redBorder Malware que lo soliciten.
Este servicio ofrece dos funcionalidades:

- **Listas**: Pueden ser de dos tipos, por un lado, existe la total, que ofrece el contenido íntegro de la caché remota, y por otro, la incremental, que permite obtener los últimos registros añadidos a la dicha caché.
- **Consulta específica**: Permite realizar una consulta por hash, URL o IP específica.

Los tipos de consultas que se pueden realizarse al servicio de reputación son las siguientes:

Method	Path	Function
GET	/reputation/vX/malware/total	Permite obtener el contenido íntegro de la caché remota.
GET	/reputation/vX/malware/total/:filter	Obtiene una lista total filtrada por hash, IP o URL.
GET	/reputation/vX/malware/incremental	Devuelve el identificador de la última lista incremental liberada.
GET	/reputation/vX/malware/incremental/:id	Permite obtener la lista incremental especificada a través del ID.
GET	/reputation/vX/malware/incremental/:id/:filter	Permite obtener la lista incremental especificada a través del ID filtrada por hash, IP o URL.
POST	/reputation/vX/malware/query	Permite realizar una consulta de un hash, IP o URL concretos.
POST	/reputation/vX/malware/status	Permite saber en qué estado se encuentra un hash.

Tabla 1 Consultas al servicio de reputación

Destacar que cuando se inicia una lista incremental, ésta comienza en la revisión 0.0 y cuando se libera el primer incremento pasará a ser la 1.0. Por tanto, por cada incremento que exista, aumentará en una unidad de la versión del sistema de reputación.

Mencionar que para usar de forma correcta el servicio de reputación con listas incrementales ha de hacerse en dos pasos reflejados a continuación:

- Consultar la lista total, obteniendo así todo el contenido de la caché remota.
- Consultar de forma periódica el incremental, para posteriormente obtener el contenido del incremento.

4.2 Arquitectura del agente EndPoint

El agente EndPoint consta de una estructura modularizada en tres partes que permite cumplir las necesidades y funcionalidades para las cuales ha sido diseñado.

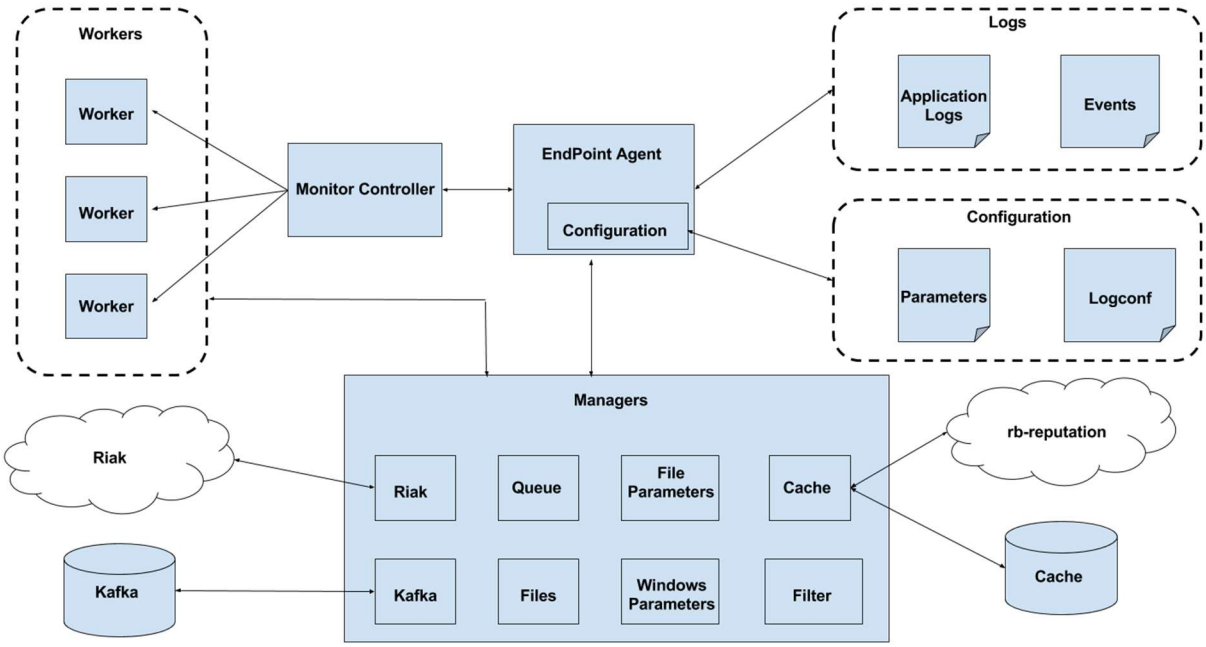


Figura 14 Arquitectura del agente EndPoint

La arquitectura consta de tres módulos:

- Módulo principal (EndPoint Agent), que ha su vez consta de un submódulo interno para su configuración.
- Módulo de gestores (Managers), que son un conjunto de submódulos independientes para manejar y utilizar los servicios del entorno de redBorder.
- Módulo controlador (Monitor Controller), encargado de manejar y gestionar los workers y tareas del agente EndPoint.

4.2.1 Módulo principal (EndPoint Agent)

Este módulo es el núcleo del agente EndPoint, debido a que se encarga de desplegar y comprobar el entorno mediante el submódulo de configuración. De igual forma, carga las librerías para su posterior uso, inicia y configura el módulo gestor e inicia el módulo controlador junto a las tareas, las cuales posteriormente son delegadas a éste.

La secuencia de arranque y configuración es la mostrada en la siguiente imagen:

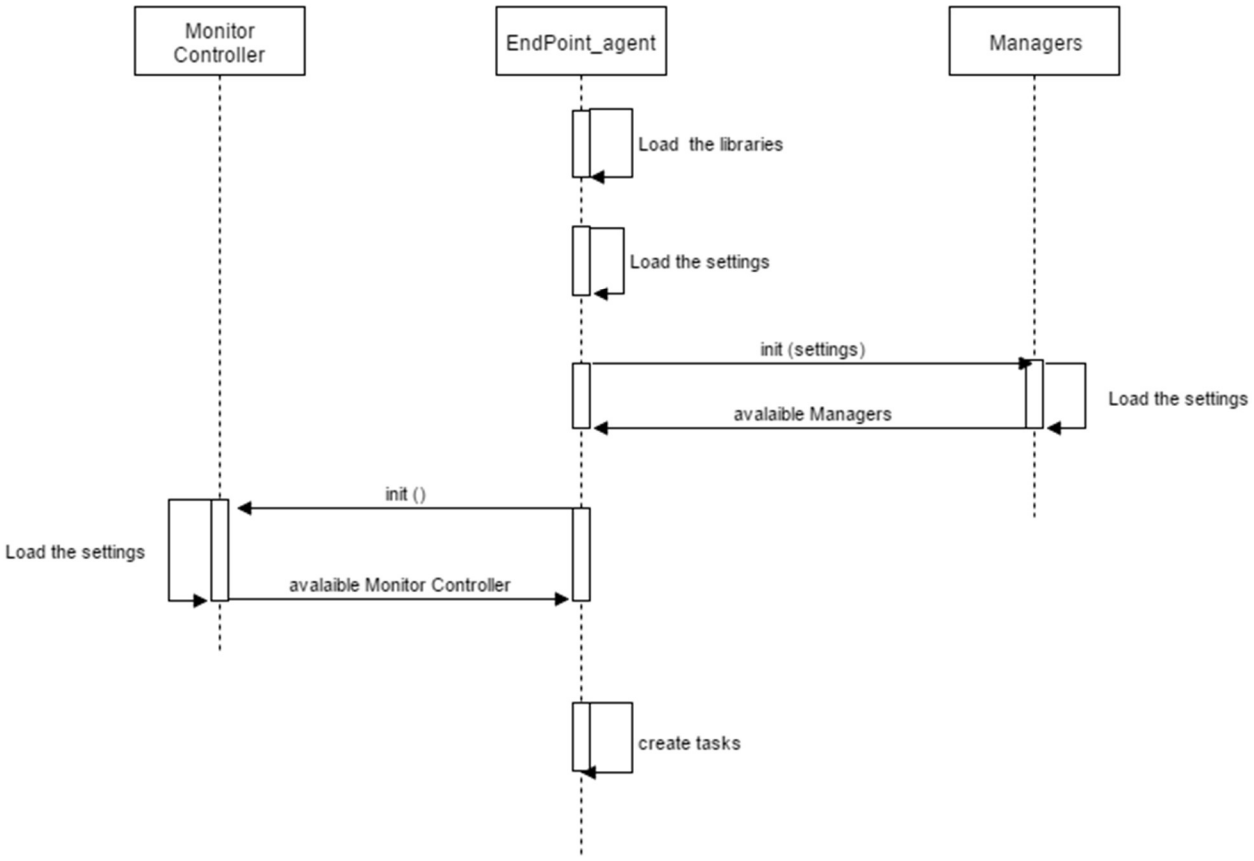


Figura 15 Secuencia de inicio

4.2.1.1 Submódulo de configuración (configuration)

Este submódulo del módulo principal se encarga de seis tareas muy definidas y distinguidas para que el agente EndPoint funcione de forma correcta:

- Carga las librerías (explicadas en la sección 4.1.1) y las deja listas para ser utilizadas.
- Carga el archivo de configuración de logs de aplicación e histórico de eventos (explicados en la sección 4.1.1).
- Carga el fichero con los parámetros necesarios para hacer uso de los servicios de reputación, Kafka y Riak.
- Inicia los logs de aplicación con la configuración obtenida anteriormente.
- Inicia el histórico de eventos de la misma forma que los logs de aplicación.
- Configura los submódulos de riak, kafka, caché y filtrado de parámetros del módulo gestor.

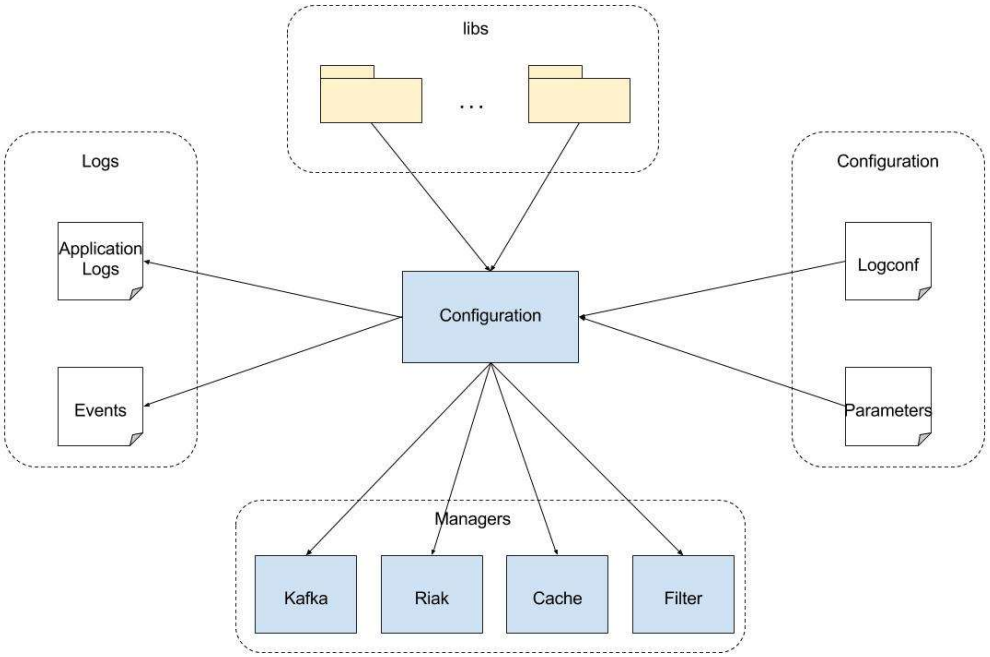


Figura 16 Tareas del submódulo de configuración

Esta son las seis acciones que realiza el submódulo de configuración del módulo principal del agente EndPoint. Una vez realizadas, se inicia el módulo controlador, se inician las tareas y se asignan las tareas al controlador para que se encargue de su realización y gestión.

4.2.2 Módulo de gestores (Managers)

Es un módulo compuesto por submódulos totalmente independientes entre ellos, que implementan una pequeña lógica, las cuales permiten obtener una serie de funcionalidades a las diferentes tareas del agente EndPoint de una forma transparente.

Los submódulos que componen este módulo son los siguientes:

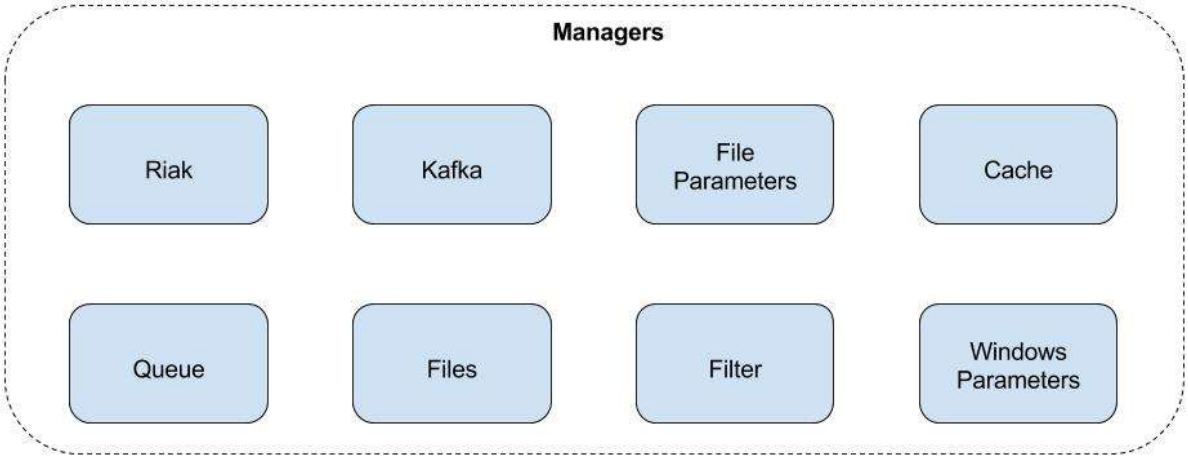


Figura 17 Gestores

4.2.2.1 Gestor de la caché (Cache)

El gestor de la caché se encarga de todas las gestiones relacionadas con la caché local y con el servicio de reputación, siendo el único elemento del agente EndPoint que entra en contacto con dichos servicios.

Este gestor se puede subdividir en dos bloques que ofrecen diferentes funcionalidades:

- **Sistema de reputación**: Uno de los bloques es el que realiza las consultas a la caché remota del sistema de reputación de redBorder Malware. Para ello, crea un worker (sección 4.2.3.1)

  Dicho bloque dispone de una única funcionalidad, la cual se inicia desde el submódulo de configuración del agente EndPoint y estará activa hasta que el redBorder Malware EndPoint se apague. Esta funcionalidad, realiza las siguientes acciones:

  - Carga los parámetros de configuración (versión de la caché remota y el alias del sistema de reputación), construye la URL y envía una petición para obtener todo el contenido de la caché remota (sección 4.1.2), guardando dicho contenido en la caché local.
  - A continuación, entra en bucle, y cada cierto periodo de tiempo, lanza una petición para obtener la incremental de la caché remota y posteriormente su contenido. Con dicho contenido actualiza la caché local.

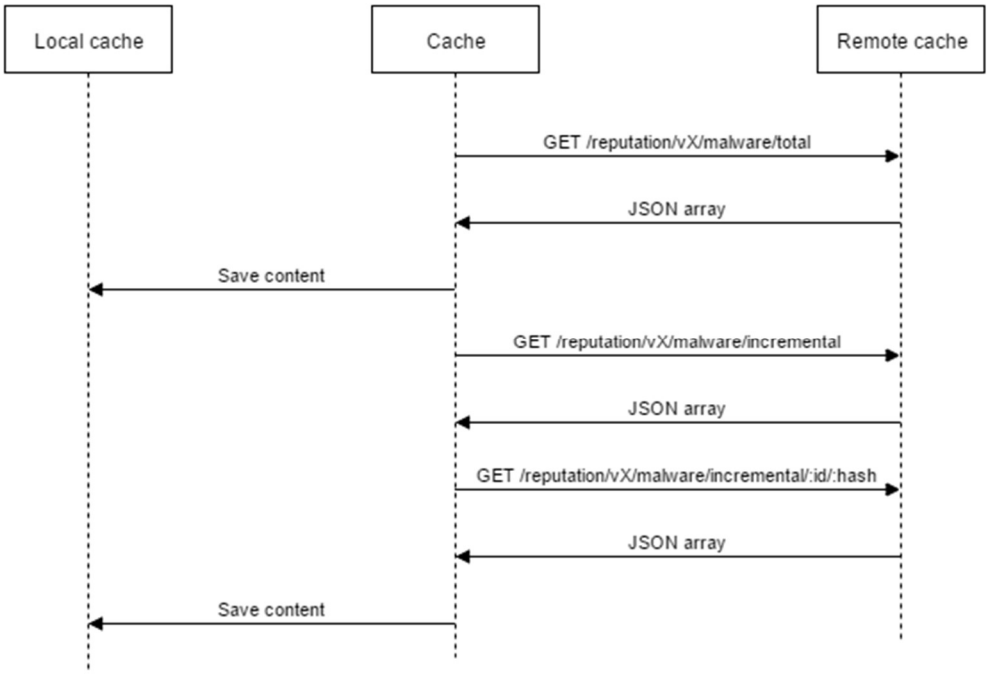


Figura 18 Consulta caché remota

Todas las respuestas de la caché remota relacionadas con el contenido son de tipo JSON, de los cuales, el gestor selecciona el hash y su score y los guarda en la caché local.

- **Caché Local**: El otro bloque del que se compone este gestor, se encarga de proporcionar una serie de funcionalidades al agente EndPoint para que haga consultas a través del gestor a la caché local cuando lo requiera. Esas funcionalidades se explican a continuación:
  - Comprobar si un hash existe en la caché local.
  - Obtener el score de un hash dado y existente en la caché local.

4.2.2.2 Gestor de Kafka (Kafka)

El gestor de Kafka se encarga de gestionar las conexiones y envios de eventos hacia el manager de redBorder Malware, los cuales tienen un broker[6]. Para lograr dicho fin, crea un productor y pone a disposición del agente EndPoint las funcionalidades de conexión y envío de mensajes.

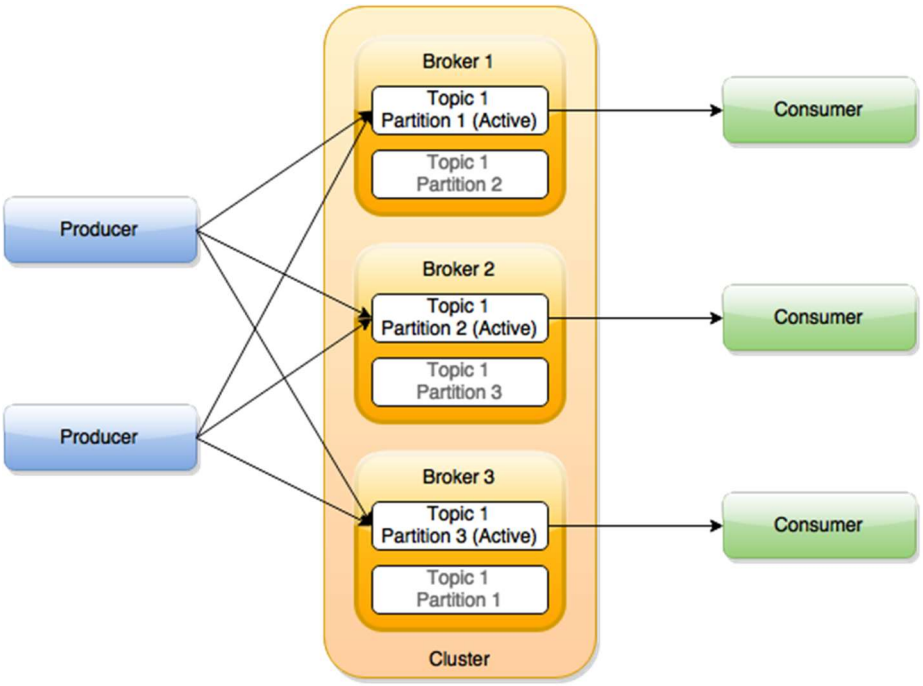


Figura 19 Productor y consumidor de Kafka

Se observa que un productor, como es el caso que aquí se describe, puede enviar eventos a distintos topics[7], en este caso los topics son dos:

- rb_ioc
- rb_endpoint

Adicionalmente, cada topic está compuesto de diferentes particiones donde enviar estos eventos. En este caso, la partición a la cual se envían los eventos viene dada por el identificador unívoco dado a cada EndPoint por GRR. Con esta información, el gestor pone a disposición del agente EndPoint las funcionalidades de conexión y envío de mensajes:

- **Conexión con el clúster de broker**: Cuando el submódulo de configuración inicia este gestor, le pasa el alias de los brokers a los cuales el cliente de Kafka debe conectarse y se realizan las siguientes acciones:
    - Se crea un cliente y se conecta al clúster.
    - Se crea un productor y se deja preparado para su uso.
- **Envío de eventos a los brokers**: Funcionalidad que está disponible para el agente EndPoint cuando este solicite su uso. Para ello, el agente EndPoint decide el topic y evento, y usa gestor para el envío del mensaje al manager. De forma adicional y transparente, esta funcionalidad cuenta con un sistema de gestión de errores, de forma que, si el error no es grave, reintenta envíar el mensaje una cantidad determinada de veces, en cambio, si el error resulta ser grave, se reinicia la conexión con el clúster de

[6] Proceso que se encarga de publicar la información recibida de los productores y que está dividida en topics.
[7] Estructura de un conjunto determinado de datos, cada topic tendrá unos datos y estructura distintos.

brokers.

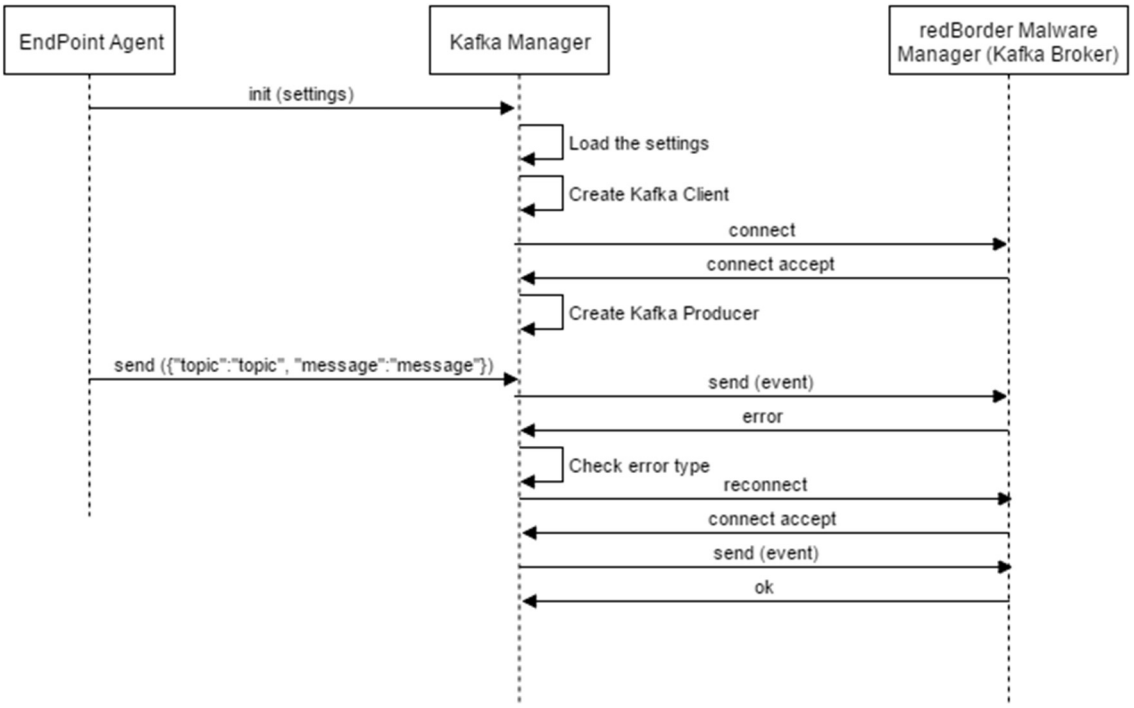


Figura 20 Secuencia del funcionamiento del gestor de Kafka

En la imagen anterior se muestran las funcionalidades que el gestor de Kafka ofrece al agente EndPoint y se detalla el caso en el cual se produce un error grave o de conexión al enviar un mensaje.

4.2.2.3 Gestor de Riak (Riak)

El gestor de Riak se encarga de proporcionar al agente EndPoint el mecanismo para enviar un fichero a Riak, dentro del manager de redBorder Malware. Para proporcionar este mecanismo es necesario configurar el gestor. El submódulo de configuración se encarga de iniciar el gestor y configurarlo enviándole los siguientes parámetros:

- Bucket [8]al que subir las muestras.
- Ruta dentro del bucket donde subir los ficheros.
- Las claves secretas y de acceso configuradas por el manager para Riak.

Una vez configurado, el gestor se queda a la espera de recibir peticiones para subir muestras. Una vez recibida la petición por parte del agente EndPoint, el gestor realiza las siguientes acciones para subir la muestra:

- Hace una copia del fichero recibido a la carpeta temporal de riak. Esta acción se realiza porque la subida del fichero puede tardar unos segundos, por tanto, el fichero estará siendo usado por el proceso de redBorder Malware EndPoint, dejándolo indisponible para otros procesos. De esta forma, el fichero que se sube a Riak es una copia del original, cuyo contenido, que es lo que realmente importa, es invariable repecto al original.
- Sube a Riak el fichero.
- Borra el fichero de la carpeta temporal.

[8] Sistema de almacenamiento de objetos, del cual se pueden añadir o extraer datos.

Una vez realizadas las acciones anteriores, el fichero se encontrará en Riak dentro del manager de redBorder Malware para ser analizado en busca de malware. Tras el análisis, se generará un score que, junto al hash del fichero, se introducirá mediante una incremental en el sistema de reputación y que el gestor de la caché obtendrá en la siguiente consulta a la caché remota.

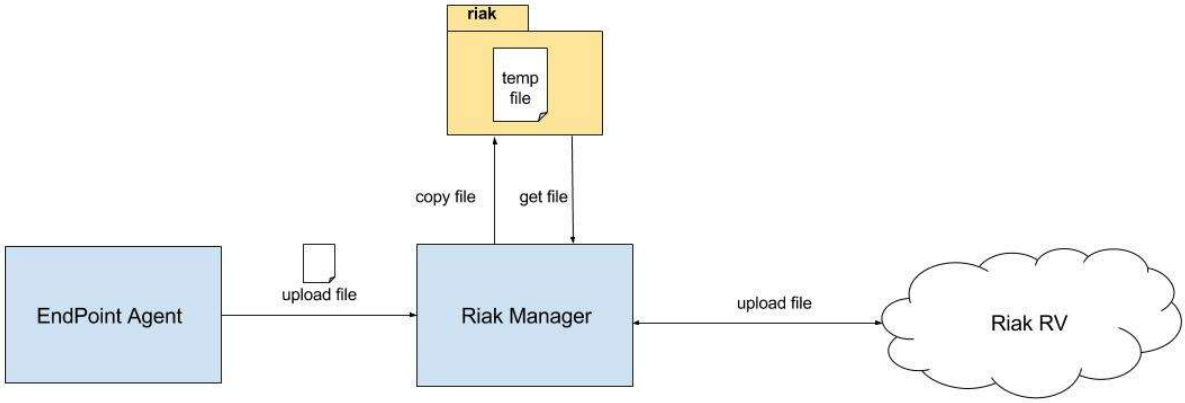


Figura 21 Funcionamiento del gestor de Riak

4.2.2.4 Gestor de filtrado (Filter)

El gestor de filtrado se encarga, mediante una pequeña lógica, de proporcionar funcionalidades al agente EndPoint de filtrado de ficheros. Es decir, se encarga de comprobar características de un fichero, de forma que el agente EndPoint decida si subir a Riak o no un fichero.

Las funcionalidades de filtrado proporcionadas son las siguientes:

- **Filtrado por ruta**: Previamente a este proyecto, se ha hecho un estudio acerca de los diferentes ficheros de logs internos del sistema operativo Windows. Debido a que los mencionados archivos cambian de forma constante, provocarían constantes procesamientos innecesarios que ocuparían recursos en el proceso del agente EndPoint. Por este motivo, existe este filtrado, para cuando se detecten estos ficheros, ignorarlos.
- **Filtrado por tamaño**: Tal como se explico en la sección 4.1.1, en el fichero "parameters.yaml" existe un parámetro llamado "file_size", que indica el tamaño máximo que puede tener un archivo para subirlo a Riak. Este filtrado se encarga, dado un fichero, de extraer su tamaño y compararlo con el máximo para indicar si es mayor, menor o igual y que de esta forma el agente EndPoint lo suba o no a Riak.
- **Filtrado por tipo**: Como se explico en la sección 4.1.1, dentro del directorio "Filter" existe un fichero llamado "Type filter" que contiene una lista de tipos de ficheros que deben ser ignorados, por tanto, no deben subirse a Riak.

```
file_filters:
        - ZIP
        - FLIC
```

Figura 22 Fichero type filter

Por tanto, lo que hace esta funcionalidad, es consultar la lista de filtrado de tipos y dado un tipo, le dice al agente EndPoint si debe ignorar el fichero correspondiente o debe subirlo a Riak.

4.2.2.5 Gestor de ficheros (Files)

Este gestor se encarga de gestionar todas las acciones posibles a realizar con un fichero de una forma totalmente transparente.

Todas las acciones gestionadas son las siguientes:

- **Apertura de ficheros**: Esta acción permite abrir un fichero asignándole una clave y un modo de apertura, los cuales se detallan a continuación:
    - Modo lectura.
    - Modo escritura.
    - Modo adición.
    - Modo lectura binaria.
    - Modo escritura binaria
- **Cierre de un fichero**: Todos los ficheros, una vez abiertos, no son cerrados hasta que se indique desde el agente EndPoint, de lo cual se encarga el gestor a través de esta funcionalidad. Para llevar a cabo esta acción, simplemente es necesario que el agente de carga pida al gestor que cierre un fichero a través de su clave asociada.
- **Escritura de ficheros**: Dada la clave asociada a un fichero abierto y un contenido a escribir en el mismo, el gestor a través de esta funcionalidad, escribe el contenido en el fichero.
- **Obtención del contenido de un fichero**: Esta acción del gestor del fichero permite obtener el contenido de un fichero con solo dar la clave asociada al mismo.
- **Obtener el descriptor de un fichero**: El gestor permite obtener el descriptor de un fichero a través de su clave asociada. Con esta acción, permite trabajar con un fichero de forma independiente al gestor si así se desea.
- **Descripción de un fichero**: Esta funcionalidad permite obtener la siguiente información acerca de un fichero dado.
    - Modo del fichero.
    - Identificador del fichero.
    - Identificador del dispositivo donde reside el fichero.
    - Número de enlaces duros.
    - Identificador del usuario dueño del fichero.
    - Identificador del grupo dueño del fichero.
    - Tamaño en bytes del fichero.
    - Tiempo en segundos del último acceso al fichero.
    - Tiempo en segundos de la última modificación del fichero.
    - Tiempo en segundos desde la creación del fichero.
- **Cierre de todos los ficheros abiertos**: El gestor permite al agente EndPoint cerrar todos los ficheros que se encuentren abiertos. De esta forma, es una funcionalidad tremendamente útil cuando se apaga el EndPoint.

4.2.2.6 Gestor de colas (Queue)

El gestor de colas, se encargará, de forma transparente, de gestionar las cuatro colas de las que consta el agente EndPoint, que son las siguientes:

- **Cola de ficheros**: En esta cola, la tarea apropiada introduce todos los eventos detectados del sistema de ficheros.
- **Cola lenta de ficheros**: Cola donde se introducen todos los eventos que se detecten del sistema de ficheros en el agente EndPoint. La diferencia con la cola anterior, es que los ficheros relacionados con dichos eventos no se suben a Riak, es decir, están filtrados, por tanto, no tienen la misma prioridad, siendo su tratamiento distinto para minimizar el consumo de recursos del sistema.
- **Cola de ficheros inexistentes**: En esta cola, aparecen los eventos del sistema de ficheros relacionados con ficheros que ya no existen en el sistema, es decir, ficheros borrados o renombrados.
- **Cola de Kafka**: Esta cola contiene todos los eventos que han de enviarse a Kafka. De esta forma, el agente EndPoint, a través de la tarea apropiada, solo tiene que usar la cola y el gestor de Kafka para notificar los eventos al manager.

Para gestionar las colas, el gestor, mediante una pequeña lógica interna, ofrece las siguientes funcionalidades:

- **Crear una cola**: Esta acción crea una cola y le asocia una clave, dejando la cola preparada para su uso.
- **Introducir elementos en la cola**: Permite encolar eventos en las diferentes colas a partir de la clave asociada a la misma.
- **Extraer elementos de la cola**: Al igual que pueden introducirse eventos a través de la clave asociada a la cola, pueden extraerse a partir de esta funcionalidad.
- **Comprobar si la cola esta vacía**: Acción que el gestor ofrece para tener un control del estado de la cola y detectar cuando deja de estar vacía para consumir.
- **Obtener el descriptor de una cola**: Como en el gestor de ficheros, existe esta funcionalidad, de forma que, si se desea, se puede trabajar con una cola al margen del gestor.

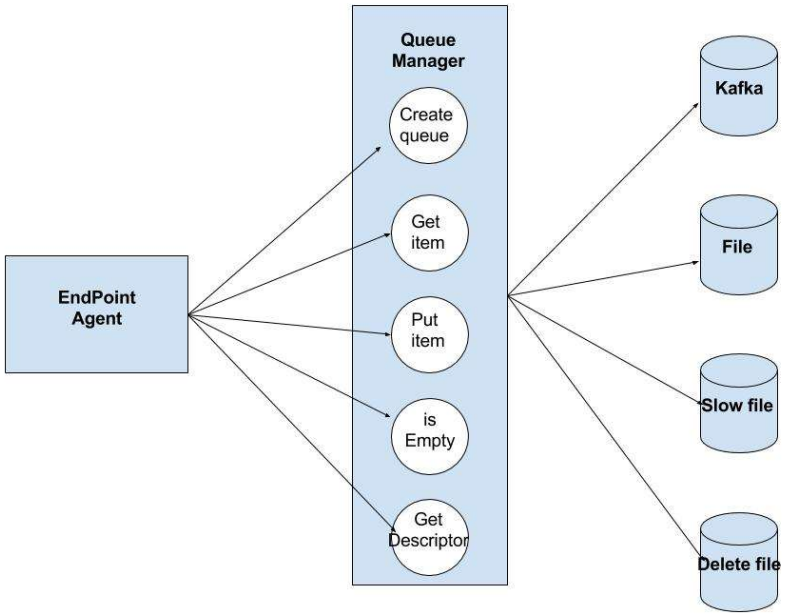


Figura 23 Gestor de colas

4.2.2.7 Gestor de parámetros de ficheros (File Parameters)

Este gestor es de suma importancia, ya que, de forma totalmente transparente, ofrece una serie de funcionalidades para extraer parámetros de ficheros, agilizando así el proceso del agente EndPoint y facilitando la búsqueda de malware en el posterior análisis.

Dichas funcionalidades son las siguientes:

- **Obtención del tamaño de un fichero**: A partir de un fichero dado, se obtiene la descripción del fichero, a partir de la cual, se consigue su tamaño en bytes. De esta forma, el agente EndPoint obtiene un dato relevante para filtrar o no el archivo.
- **Obtención del hash MD5 de un fichero**: Este método de reducción criptográfico se usa para obtener el resumen en 128 bits de una cadena de caracteres.
- **Obtención del hash SHA256 de un fichero**: Al igual que al generar el hash MD5, se hace un resumen de un conjunto de caracteres, pero esta vez de 256 bits.
- **Obtención de la ACL [9](Access Control List)**: Esta operación, permite obtener los permisos en decimal de un fichero, lo cual es muy útil en caso de análisis forense, ya que en el histórico de eventos aparecerá reflejado.
- **Obtención del tipo del fichero**: Esta funcionalidad permite al agente EndPoint conocer el tipo de un fichero dado para poder consultarlo con el gestor de filtrado y así determinar si el fichero es subido o no a Riak.

  Para dicho fin, el gestor hace uso del fichero "Type file" dentro del directorio del entorno "filters".

[9] Lista de Control de Acceso (Access Control List) es una lista con un conjunto de perfiles de usuarios que tienen una serie de permisos o restricciones asociados para usar un determinado fichero del sistema. Se puede considerar el equivalente en Windows a los permisos de un fichero en Linux.

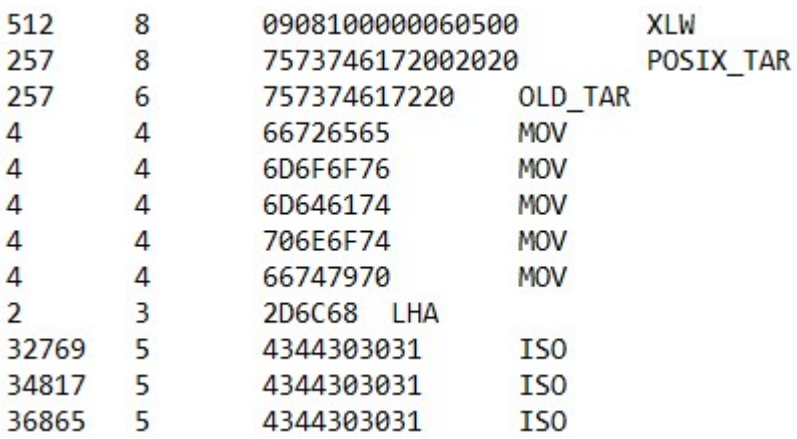


Figura 24 Tipos de los ficheros

Como se observa en la imagen, para averiguar el tipo lo que se hace es parsear el fichero que consta de:

- Byte de comienzo.
- Nnúmero de bytes que componen los caracteres característicos del fichero.
- Los caracteres.
- Tipo correspondiente a los caracteres.

Lo que el gestor hace es extraer los caracteres del fichero a partir del byte de comienzo y el número de bytes y compararlos con los aparecidos en el fichero "Type files".

4.2.2.8 Gestor de los parámetros de Windows (Windows Parameters)

Este gestor es esencial para poder tener todos los datos de un EndPoint en el manager y de esta forma conocer en que equipo de la red se ha detectado un determinado malware o se ha producido cierta incidencia.

Para ello, provee al agente EndPoint de dos funcionalidades:

- Extracción de los siguientes parámetros referentes al equipo y sistema operativo para almacenarlos en memoria.
  - IP del equipo.
  - MAC del equipo.
  - Subred del equipo.
  - Dominio DNS del equipo.
  - Nombre del equipo.
  - Perfil de usuario que está usando el equipo.
- Extracción de la memoria de los anteriores datos.

Al ser estos datos comunes a todos los eventos de ficheros que han de enviarse mediante Kafka al manager de redBorder Malware, cada vez que se detecta un nuevo evento de alteración del sistema de ficheros, se consultan estos datos mediante el gestor y se introducen como parte del evento.

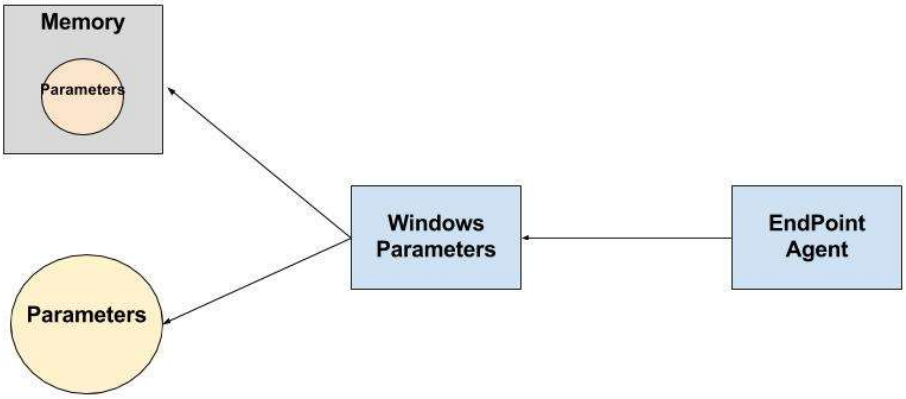


Figura 25 Gestor de parámetros de Windows

4.2.3 Módulo controlador (Monitor Controller)

Este módulo es de una enorme importancia dentro del agente EndPoint. Se encarga de gestionar las tareas del agente EndPoint para cumplir con las funcionalidades para las que ha sido diseñado.

El módulo principal se encarga de configurar y crear las tareas a realizar para, a continuación, pasar el control de las mismas a este módulo. Una vez que tiene control sobre las tareas, las despliega usando un worker, que las prepara y ejecuta.

La ventaja de desplegar workers es que la gestiones de las tareas se simplifican, ya que se encargan de monitorizar el estado de estas. Cuando una tarea es inicializada o falla por cualquier circunstancia es reportado por el worker al controlador de tareas y de éste, al módulo principal.

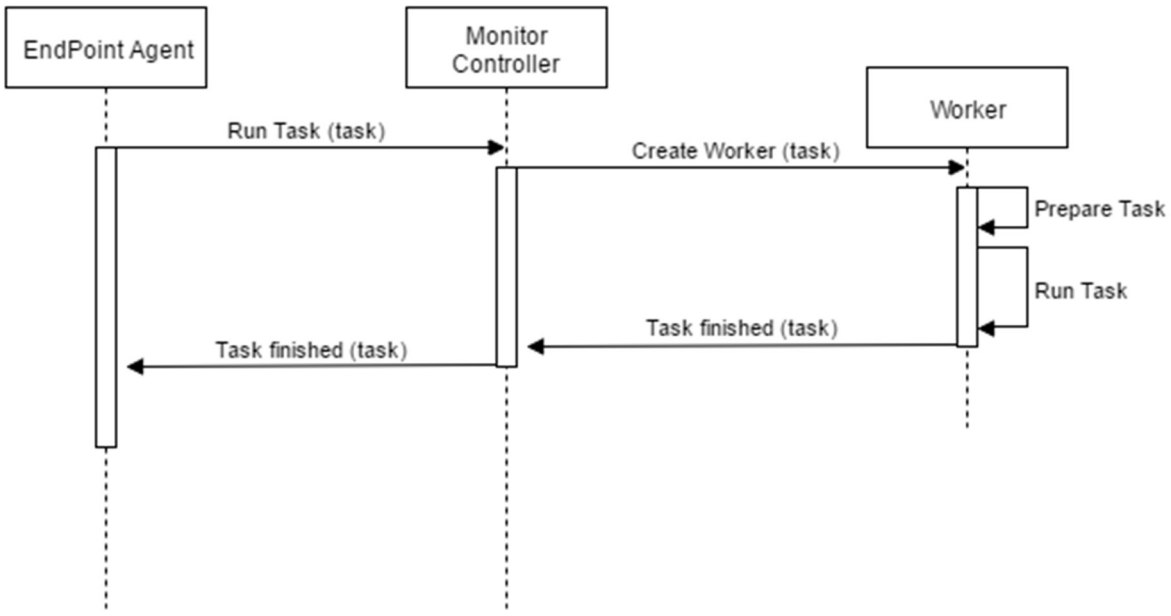


Figura 26 Funcionamiento del módulo controlador

Podemos comprobar en la imagen anterior, que lo primero de lo que se encarga un worker cuando se crea y recibe una tarea por parte del controlador, es preparala/configurarla y posteriormente ejecutarla.

Dicha preparación y configuración de las tareas es fundamental, ya que resulta indispensable para que funcionen de forma correcta.

Una vez la tarea ha sido ejecutada, el worker espera la finalización de la misma, o bien, una orden por parte del módulo principal para finalizarla. Concluida la ejecución de la tarea, independientemente de la forma, el worker se destruye, avisando el controlador al módulo principal de este hecho.

La gestión de los workers es la causa por la cual se han usado, ya que permite al controlador llevar un exhaustivo control de los mismos. Entre las funcionalidades de gestión de workers destacan:

- La ejecución de tareas.
- Obtención del estado de los workers en un instante de tiempo.
- Obtener el resultado de la ejecución de una tarea.
- Parar una tarea en cualquier instante de tiempo.

Gracias a estas funcionalidades disponibles en este módulo, el agente EndPoint posee un gestor al cual enviar las tareas para que, de forma totalmente controlada, se desplieguen, ejecuten y gestione de forma transparente.

4.2.3.1 Workers

Un worker no es más que un proceso. La enorme ventaja que poseen los workers, adicionalmente a la gentión, es que, tras crearlos, asignarles tareas y ejecutarlos, generan una serie de procesos hijos que se ejecutan en segundo plano, permitiendo al proceso padre continuar sus tareas sin tener que esperar a la finalización de estos.

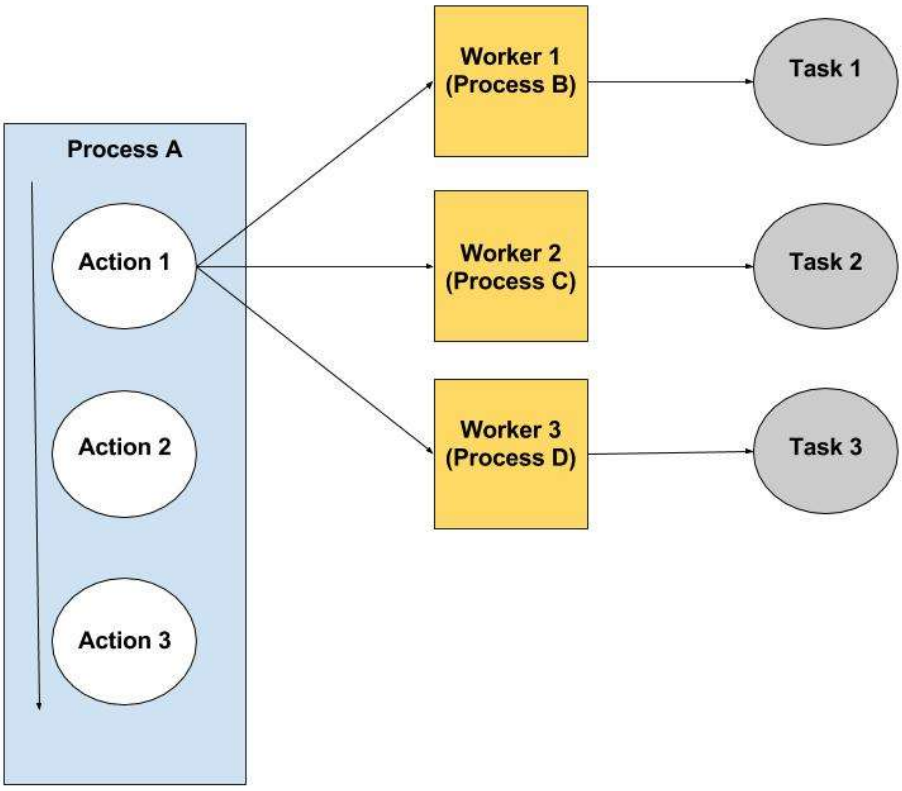


Figura 27 Funcionamiento de los workers

Como se puede observar en la imagen, el proceso A ejecuta la acción 1, que a su vez lanza una serie de workers, los cuales no son mas que procesos hijos que preparan y ejecutan tareas.

Una vez creados los workers y asignadas las tareas, el proceso A continua con las acciones 2 y 3 mientras los procesos B, C y D están activos. Una vez finalizados, los workers lanzan un método interno call back para anunciar al proceso padre que han finalizado.

Adicionalmente, como se ha explicado anteriormente, las facilidades de gestión de los workers, los hacen ideales para realizar cualquier tipo de tarea.

4.2.3.2 Tareas (Tasks)

Las tareas están directamente relacionadas con las principales funcionalidades que ofrece este proyecto. Por tanto, cada una de ellas requiere de una o varias tareas para poder llevarse a cabo.

En la siguiente tabla se aprecian las funcionalidades del proyecto junto a una relación con las tareas necesarias asociadas.

Funcionalidades	Workers	Tareas relacionadas
Detección en tiempo real de modificaciones en el sistema de ficheros	4	• Monitorización de las particiones físicas • Captura de ficheros • Captura lenta de ficheros • Captura de ficheros no existentes
Captura de procesos en tiempo real	2	• Captura de procesos creados • Captura de procesos destruidos
Detección de las conexiones de red en tiempo real	1	• Detección de las conexiones de red
Generación de un histórico de eventos	0	Todas las anteriores tareas se encargan, cada vez que generan un evento, de añadirlo al histórico de eventos

Tabla 2 Relación Funcionalidades – Tareas

Como se observa en la tabla anterior, cada tarea esta gestionada por un worker, el cual está encargado de prepararla, ejecutarla y, a través de los estados, gestionarla. Por tanto, los estados de las tareas son imprescindibles para la gestión de las mismas, y son los siguientes:

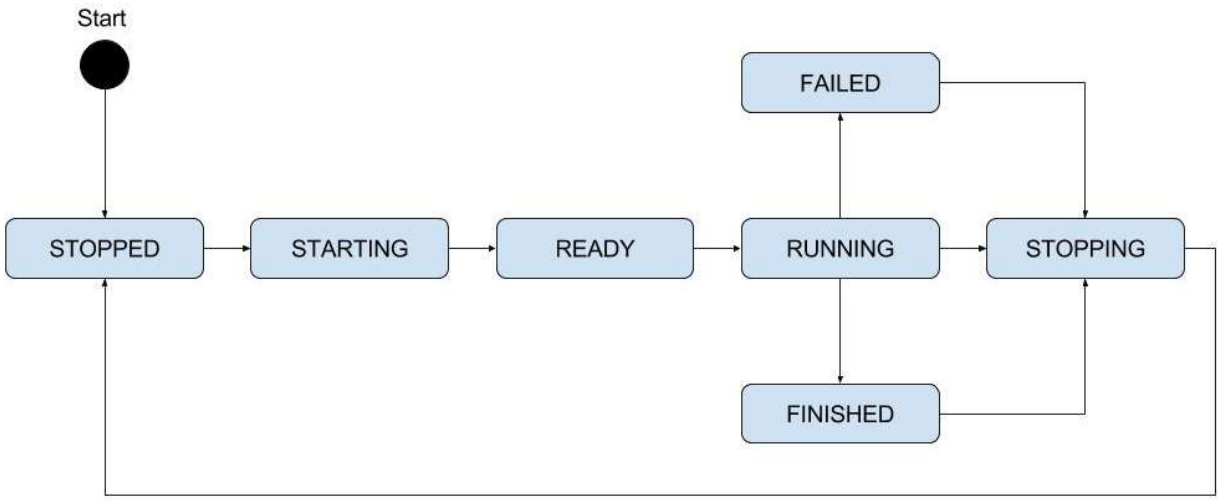


Figura 28 Estado de las tareas

Se observa, que cuando a un worker se le asigna una tarea, está parada, y que es el worker el encargado de prepararla. Para ello, la inicia/configura y la deja lista para su ejecución, notificándole este hecho al agente EndPoint. Posteriormente, se ejecuta, pudiendo ocurrir que falle, o bien que la ejecución sea correcta y finalice, o que mientras se ejecuta, el agente EndPoint ordene al módulo controlador finalizar la tarea. En todos estos casos, la tarea se para y vuelve al estado inicial.

Destacar que, en este proyecto, debido a su carácter y a las funcionalidades que cubre, las tareas no finalizan

hasta que el agente EndPoint se para, dando la orden de finalización de todas las tareas de forma previa.

4.2.3.2.1 Tarea de monitorización de las particiones físicas del disco del equipo

Esta tarea se encarga de detectar los cambios en el sistema de ficheros, por tanto, fundamental para la lucha contra el malware. Para ello, realiza una serie de acciones.

En primer lugar, monitoriza todas las particiones lógicas y físicas existentes en el disco del equipo. En el caso de Windows, son "C:\", "D:\", "E:\", etc. De esta forma, monitoriza todo el sistema de ficheros y además todas las unidades de almacenamiento extraíbles que se inserten en el equipo. La monitorización finalizará cuando la unidad desaparezca, es decir, en caso de un USB, cuando sea extraido del sistema. Esta acción, es fundamental dentro del proyecto, ya que monitoriza uno de los puntos calientes de entrada de malware en un entorno empresarial.

En segundo lugar, por cada partición física detectada, se monitoriza, a través de WMI, su sistema de ficheros hasta que la partición desaparezca del disco. La monitorización implica que a partir de ese instante se empiecen a detectar y capturar diferentes eventos en el sistema de fichero. Los eventos son:

- Fichero creado
- Fichero borrado
- Fichero modificado

Destacar que para considerar que un fichero ha sido modificado, debe cumplirse algunos de los siguientes requisitos:

- Cambio en el contenido
- Cambio de permisos
- Cambio del nombre
- Cambio en los metadatos
- Cambio del tamaño

Estos requisitos no aparecen reflejados cuando se captura un evento a excepción del cambio de nombre, por tanto, sólo se sabe que el fichero ha sido modificado o renombrado. Es decir, se sabe que un fichero ha sido modificado, pero no se sabe si es porque han cambiado los permisos, el contenido, los metadatos o el tamaño.

Por último, una vez detectado y capturado un evento, esta tarea extraen los datos y se introducen en la correspondiente cola. Los datos extraidos son:

- **Ruta del fichero**: ruta completa del fichero afectado.
- **Tipo de evento**: se refiere a si el evento es de un fichero creado, borrado o modificado. En caso de ser el evento de tipo modificado y el fichero es renombrado, se pueden extraer de forma adicional el nombre anterior y el actual. Al tener un nuevo nombre, el sistema también genera un evento de tipo fichero creado.

En función del tipo del evento, se usa la una cola u otra. En caso de que un fichero sea de tipo creado o modificado, el evnto se añade a la cola "cola de ficheros", pero en caso de ser de tipo borrado o renombrado, el evento se introduce en la "cola de ficheros inexistentes". De esta forma, las correspondientes tareas que leen de cada cola tratan de una forma distinta cada evento.

Mencionar, que la tarea, antes de encolar el evento, comprueba a través del gestor de filtrado si la ruta del fichero debe ser ignorada o no. En caso que haya que filtrar, la muestra no se añade a ninguna cola, por tanto, se ignora.

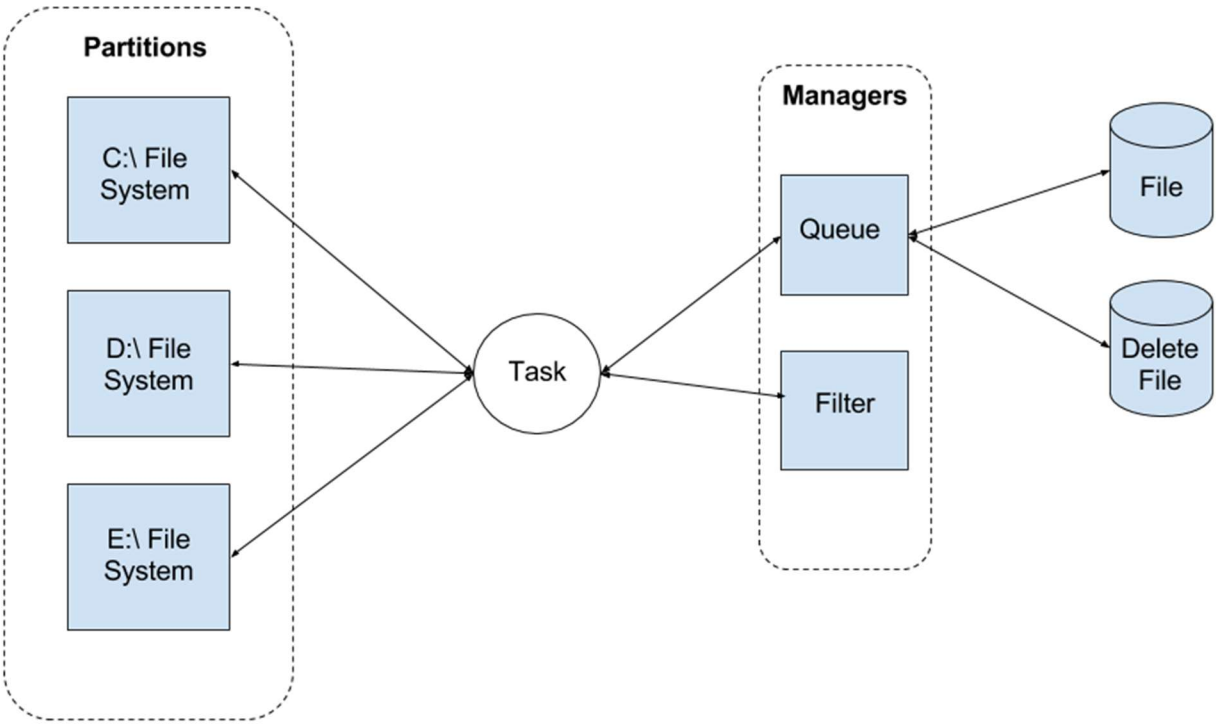


Figura 29 Tarea de monitorización de particiones lógicas

En la imagen se observa, como de cada partición lógica, se monitoriza el sistema de ficheros, detectando así cada cambio producido en el mismo. Por cada cambio producido, la tarea extrae los datos del evento y en función del tipo del evento y de si de la ruta esta filtrada o no, a través del gestor de cola, lo añade a una cola u otra.

4.2.3.2.2 Tarea de captura de ficheros

Tarea encargada de leer de forma continua de la cola de ficheros, comprobar si un fichero está filtrado por tipo o tamaño, extraer todos los datos necesarios de un fichero, subir los archivos no filtrados a Riak a través del gestor, enviar a Kafka a través del gestor el evento, y añadir el evento al histórico de eventos.

Para empezar, la tarea obtiene de la cola de ficheros un evento, extrae la ruta y tipo del evento y, a través del gestor de parámetros de ficheros, obtiene el tipo del fichero y su tamaño. A continuación, comprueba si está filtrado por tipo o por tamaño usando el gestor de filtrado.

En caso de estar filtrado el fichero, se vuelve a introducir el evento extraido de la cola de ficheros en la cola lenta de ficheros, incluyendo el tipo del fichero. De esta forma, se priorizan los ficheros que deban tratarse y subirse a Riak para que no sufran retardo y se puedan analizar los antes posible.

A continuación, a los ficheros no filtrados se le extraen, a través del gestor de parámetros de ficheros, los siguientes parámetros para construir un nuevo evento que enviar al manager a través de Kafka:

- Hash SHA256
- Hash MD5
- Permisos
- Extensión
- Nombre

A estos parámetros, hay que añadir el tamaño y tipo del fichero, que con anterioridad se obtuvieron.

Una vez se tienen los parámetros, se comprueba usando el gestor de la caché, si el hash SHA256 es conocido o desconocido en la caché local. En caso de ser conocido, no hay que subir el fichero a Riak, por tanto, se obtiene el score asociado al hash, y se añade al nuevo evento construido anteriormente. Si el fichero resulta ser desconocido en la caché local, se añade una nueva entrada en la misma, introduciendo el SHA256 correspondiente al fichero y el score a menos uno (-1) que indica desconocido. A continuación, usando el gestor de Riak, se envía al manager el fichero para su análisis, y se completa el evento construido con anterioridad con el score.

Destacar que la nueva entrada añadida a la caché local, se actualizará en el siguiente incremento que el gestor de la caché descargue del sistema de reputación, debido a que el manager analizará el fichero y le asignará un score, actualizando así la caché remota.

Por último, haciendo uso del gestor de parámetros de Windows, se obtienen los parámetros, se completa el evento, y se encola para su envío al manager mediante Kafka.

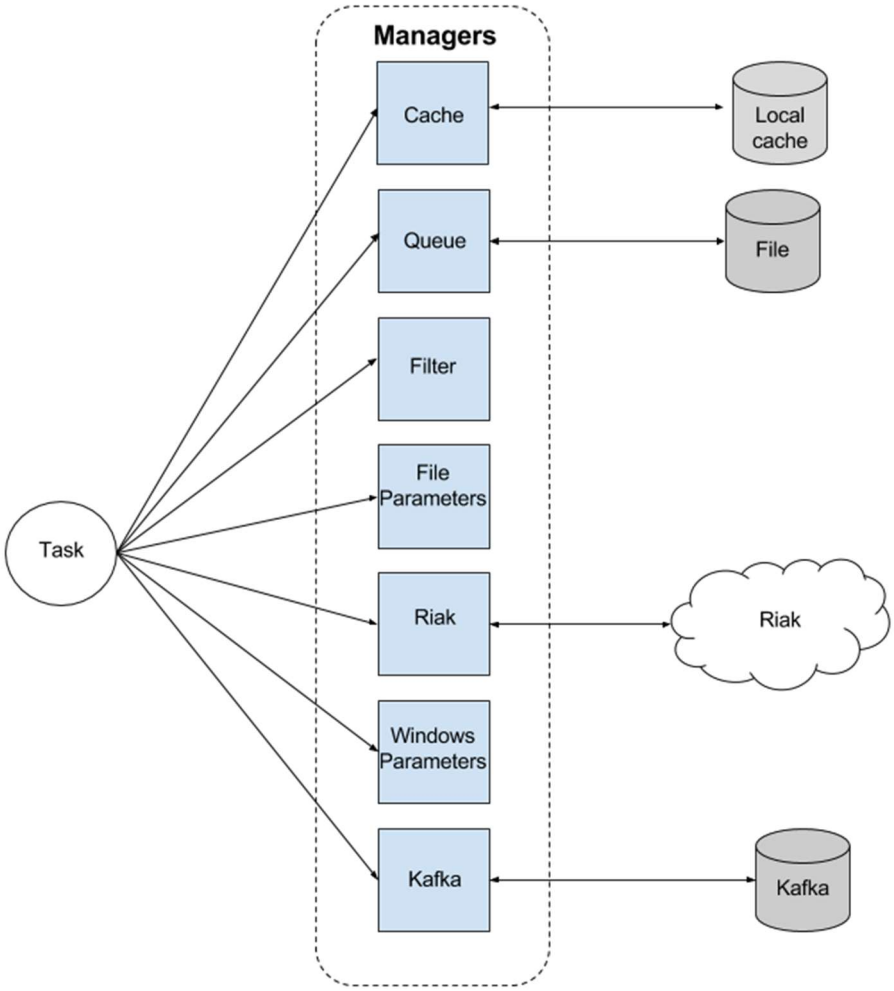


Figura 30 Tarea de captura de ficheros

4.2.3.2.3 Tarea de captura lenta de ficheros

Esta tarea se encarga de leer de la cola lenta de ficheros los eventos para extraer todos los parámetros de los mismos, crear un nuevo evento con dichos parámetros e introducirlo en la cola de Kafka. De manera adicional, añade al histórico de eventos una entrada para enriquecerlo de cara a un posible análisis forense posterior.

Se trata de una tarea parecida a la anterior, pero mucho más simple, ya que los eventos que extre de la cola son de ficheros filtrados, que no deben subirse a Riak. El objetivo de esta tarea, a parte de informar al manager y enriquecer el histórico de eventos, es reducir el consumo de recursos al máximo al extraer los datos de estos ficheros que han sido filtrados, ya que los recursos deben destinarse a los ficheros que deben ser analizados.

Destacar que los eventos que se extraen de la cola, pueden referirse a eventos relacionados con ficheros de tipo ISO, AVI, MOV, etc. Por tanto, ficheros de un gran tamaño, los cuales pueden hacer que el gestor de parámetros de ficheros tarde un tiempo amplio en obtener ciertos parámetros, como el SHA256 o MD5, ya que se debe cargar antes todo el contenido del correspondiente fichero. Por este motivo, esta tarea no es prioritaria, ni importa el tiempo que tarde en realizarse, lo único que importa es que termine, se informe al manager del evento sucedido, y como ya se ha explicado, que ocupe los mínimos recursos posibles.

La tarea se encarga de leer la cola lenta de ficheros hasta que el agente de EndPoint envía la orden al worker que la controla de finalizar la misma.

De cada elemento que se lee de la cola, se extrae la ruta del fichero, el tipo del evento y el tipo del fichero. Una vez obtenidos, se recurre al gestor de los parámetros de ficheros para obtener los siguientes parámetros:

- Hash SHA256
- Hash MD5
- Permisos
- Tamaño

Una vez obtenidos, se crea un nuevo evento para enviarlo al manager usando Kafka. Adicionalmente, se añaden los siguientes parámetros al evento:

- Extensión del fichero
- Nombre del fichero
- Ruta del fichero
- Tipo del evento extraido de la cola

Por último, se recurre al gestor de los parámetros de Windows para conseguir los parámetros referentes al sistema operativo y usuario que lo usa.

Tras el último paso, se envía al manager el evento haciendo uso del gestor de Kafka y se añade al histórico de eventos.

4.2.3.2.4 Tarea de captura de ficheros inexistentes

Esta tarea se encarga de leer de forma continua de la "cola de ficheros inexistentes" y extraer los elementos para construir un nuevo evento que enviar al manager mediante el uso del gestor de Kafka.

Se debe tener en cuenta, que en esta cola sólo se introducen eventos relacionados con archivos que ya no existen en el sistema de ficheros, es decir, ficheros borrados o ficheros renombrados. Por tanto, no se puede trabajar con esos ficheros, solo se puede anunciar lo ocurrido al manager y añadirlo al histórico de eventos.

Se generan dos tipos de eventos en función del tipo de evento extraido de la cola:

- **Ficheros borrados**: Se genera un nuevo evento en el que se informa que un fichero ha sido borrado.
- **Ficheros renombrados**: Se crea un nuevo evento informando que un fichero llamado de una forma, ha pasado a llamarse de otra.

4.2.3.2.5 Tarea de captura de procesos creados

Tarea creada a partir de la necesidad de detectar en tiempo real la ejecución de un programa posiblemente infectado con malware, y a través del manager de redBorder Malware, actuar en consecuencia.

Para llevar a cabo esta necesidad, esta tarea monitoriza la creación de procesos. Todo ello gracias a WMI, que permite capturar los procesos y obtener una serie de parámetros de los mismos, imprescindibles para el posterior análisis mediante IOC.

Por cada proceso creado y capturado, esta tarea, extrae la siguiente información con el fin de construir un evento que enviar al manager:

- **Nombre**: Nombre del proceso
- **PID**: Identificador numérico del proceso que lo identifica de forma unívoca.
- **El ejecutable**: Fichero que se ha ejecutado provocando la creación del proceso.
- **La priorida**: Prioridad numérica (1-13) que un tiene para ejecutarse dentro del sistema.
- **El identificado del perfil de usuario**: Identificador numérico del perfil de usuario que ha ejecutado el proceso.
- **El ejecutable padre**: Este parámetro hace referencia al fichero que ha provocado el proceso padre del proceso capturado, en caso de su existencia.

Una vez obtenidos todos los parámetros, se construye un nuevo evento, se introducen los parámetros y, por un lado, se añade una nueva entrada en el histórico de eventos y, por otro, se encola el evento para enviarse al manager.

El manager de redBorder Malware, recoge el evento y lo ejecuta contra IOC para detectar si el proceso ejecutado entraña algún peligro para la seguridad del equipo y de la red en la que se encuentra. En caso de encontrar algún peligro, mediante GRR, actúa de inmediato y paraliza la amenaza.

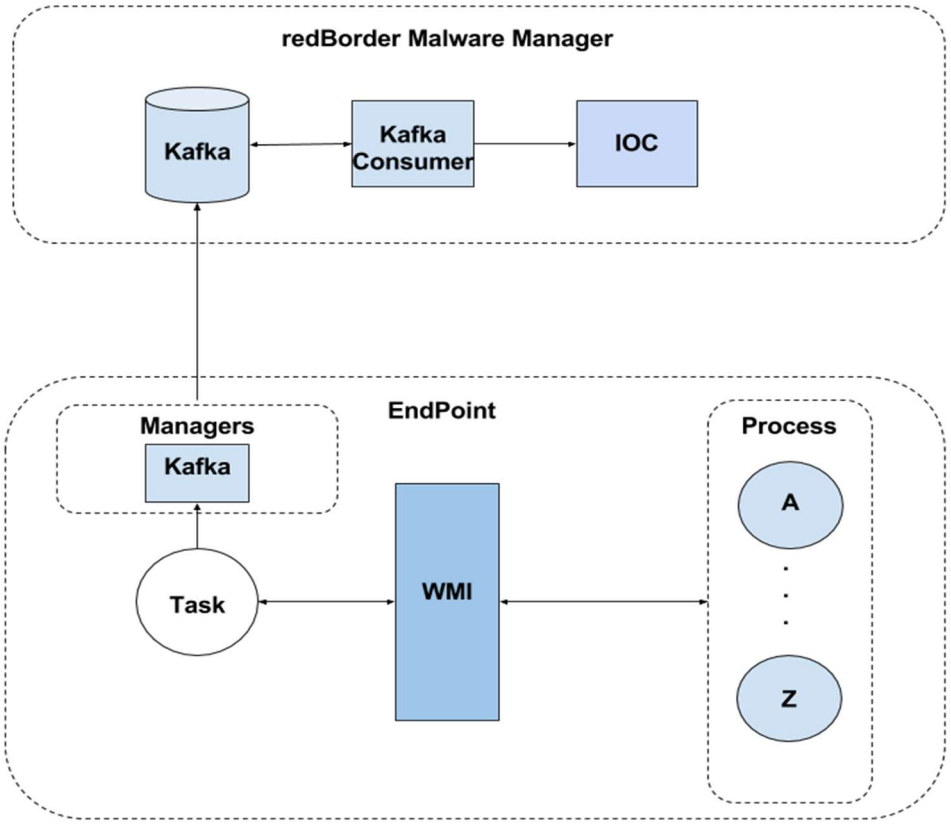


Figura 31 Tarea de captura de procesos creados

4.2.3.2.6 Tarea de captura de procesos destruidos

Al igual que la tarea de captura de procesos creados, tan necesaria para detectar en tiempo real posibles ficheros infectados de malware, es necesario tener una tarea que se encargue de detectar, en tiempo real, la destrucción de un proceso. Este necesario control de los procesos tiene dos motivos principales:

- El primer motivo, es que cuando en el manager se reciba la alerta y se compare contra IOC, se debe tener un control en tiempo real, de la ejecución de un archivo infectado y del equipo implicado para poder actuar en consecuecia.
- El segundo motivo tiene que ver con el análisis forense. Si se tiene un exhaustivo control acerca de la ejecución de un malware, se puede determinar desde los ficheros que ha podido ejecutar de forma adicional ese archivo, hasta los daños causados en el periodo de tiempo que ha estado activo.

Por estos motivos, existe esta tarea, que se encarga de enviar al manager, a través del gestor Kafka, eventos relacionados con la destrucción de un proceso, para que una vez comparada contra IOC, gestione las alertas de la forma oportuna, y para añadir una nueva entrada en el histórico de eventos. Para ello, se monitoriza en tiempo real la destrucción de procesos y se obtienen datos acerca del mismo, todo ello gracias a WMI.

Los procesos que se destruyen guardan sus datos en memoria durante un tiempo determinado antes de ser eliminados por completo. Estos datos, al no existir el proceso, son escasos y son los siguientes:

- Nombre
- PID

Con estos datos, se construye un nuevo evento, se introducen en la cola de Kafka para enviarlo al manager, y se añade una nueva entrada al histórico de eventos.

Los datos extraidos en esta tarea pueden parecer escasos, pero una vez llegado el evento al manager, se correlacionan los datos con el del proceso creado y se extraen los demás datos.

4.2.3.2.7 Tarea de detección de las conexiones de red

La detección en tiempo real de conexiones de red es de vital importancia en la lucha contra ataques o amenazas avanzadas, ya que un temprano conocimiento de las conexiones, puede evitar un gran perjuicio tanto en un equipo como en una red.

Hoy día, muchos EndPoint son tomados de forma remota por botnets [10] que establecen el control a partir de un fichero instalado en el equipo, el cual duerme hasta que recibe la orden de actuar. También existen organizaciones de cibercriminales que se dedican a extraer información de empresas mediante ataques.

Todo lo anterior, se puede evitar si se detectan a tiempo las conexiones, ya que permite actuar de forma inmediata y evitar cualquier daño que pueda llegar a producirse. Por este motivo, el agente EndPoint implementa esta tarea, que monitoriza, a partir de los procesos creados (encargados de generar conexiones), las conexiones que se establecen en un EndPoint.

Una vez extraidas las conexiones, se establece la ruta completa de direcciones IP y puertos hasta el destino, se crea un nuevo evento, se encola para enviarse a Kafka y se añade una nueva entrada al fichero de histórico de eventos.

[10] Conjunto de servidores (clúster) que de forma remota controlan un equipo mediante la instalación en el mismo de un malware, convirtiendo ese equipo en un zombie.

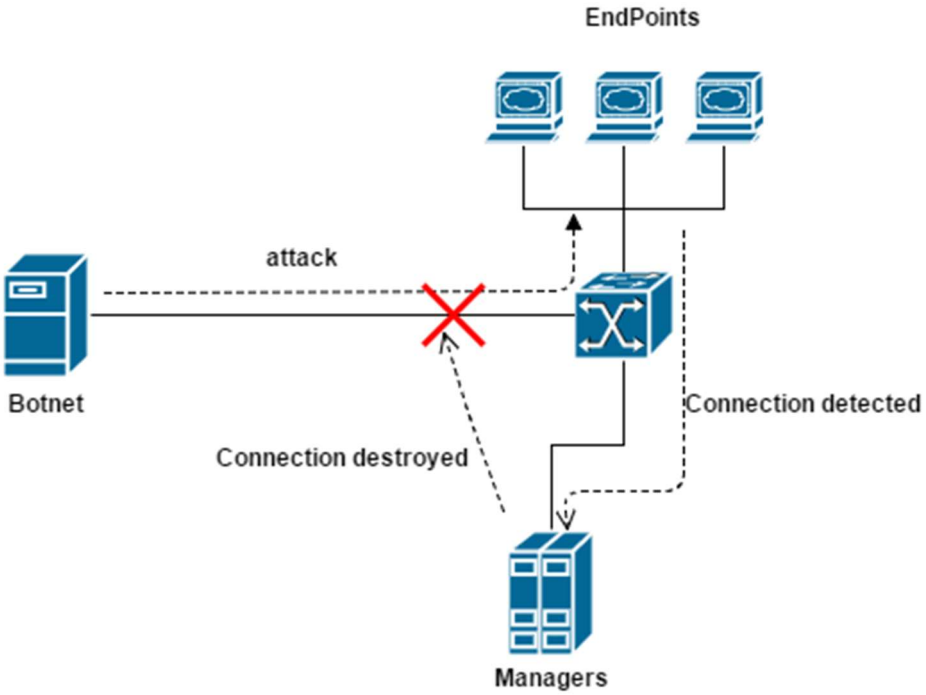


Figura 32 Tarea de captura de conexiones de red

Tal como se observa en la imagen anterior, el manager, una vez recibido el evento y ejecutado contra los IOC, si se detecta una amenaza, ejecuta mediante GRR la acción pertinente para evitar cualquier daño o robo de información que los EndPoints puedan sufrir.

4.2.3.2.8 Tarea de envío de eventos

En todas las tareas anteriores se necesita enviar eventos al manager mediante Kafka. Para evitar envíos masivos de eventos simultaneos al manager y agilizar todo el proceso de envío, se ha creado esta tarea.

Es una tarea simple, ya que se encarga de extraer eventos, de forma constante, de la cola de Kafka mediante el gestor de colas y mediante el uso del gestor, enviarlos al manager de redBorder Malware usando el topic y partición adecuada.

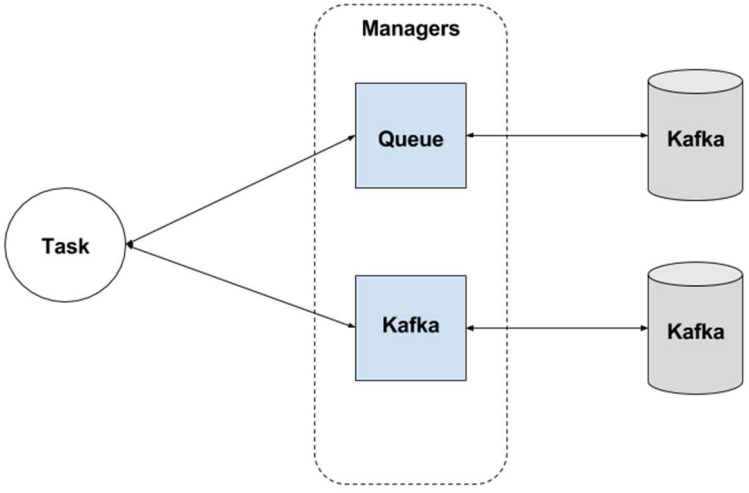


Figura 33 Tarea de envío de eventos

4.3 Eventos

En este proyecto, tal como se ha mencionado en puntos anteriores, genera de forma continua eventos con información referente a la detección de posibles ficheros infectados, o bien, a procesos o conexiones de red.

Estos eventos, son siempre enviados al manager de redBorder Malware a través de Apache Kafka haciendo uso de la tarea de envío de eventos. Tal como se explico en la sección 4.2.2.2, Kafka está dividida en topics, que no son más que diferentes estructuras de datos, en este caso JSON. Además, también se explicó que en este proyecto se usan dos topics:

- rb_ioc
- rb_endpoint

Por tanto, se tienen dos estructuras de datos distintas, la primera, usada en todas las tareas, y la segunda usada sólo en las tareas relacionadas con la captura de ficheros.

4.3.1 rb_ioc

Este topics lo usa el agente EndPoint en cualquier tarea, ya sean las de captura de ficheros, detección de procesos o detección de conexiones web.

En caso de ser nuevo el fichero el evento es del siguiente tipo:


```
{
        "cache_score": -1,
        "action": "updated",
        "namespace_uuid": "C.91c40668ddea3975",
        "filename": "setup.exe",
        "type": "file_capture",
        "file_type": "MSEXE",
        "acl": 32805760,
        "timestamp": 1465424338,
        "file_extension": ".exe",
        "md5": "4fe45b73cb23e4e77fcf6463f93c85fb",
        "filepath": "C:\\Program Files (x86)\\Google\\Chrome\\Application\\51.0.2704.84\\Installer\\setup.exe",
        "endpoint_uuid": "C.91c40668ddea3975",
        "sha256": "d42a7b102ccbbb69a1074a8d65fe3c41e3d7389503c10f10238b48a6667e6d2f"
}
```


Figura 34 Evento de detección de un fichero desconocido

Se observa como en el evento se encuentran una serie de parámetros que se explican a continuación:

- **cache_score**: Score, que el hash SHA256 asociado al fichero, tiene en la caché local.
- **action**: Tipo de evento asociado al fichero detectado. En este caso el fichero ha sido modificado.
- **namespace_uuid**: Identificador del EndPoint emisor del evento. Este parámetro se usa para la parte de Big Data en el manager de redBorder Malware.
- **filename**: Nombre del fichero capturado.
- **type**: Tipo del evento. En este caso, fichero capturado.
- **file_type**: Tipo del fichero.
- **acl**: Permisos del fichero en decimal.
- **timestamp**: Marca de tiempo del instante de detección del fichero.

- **file_extension**: Extensión del fichero.
- **md5**: Hash MD5 del fichero, es decir, resumen de 128 bits del contenido del fichero.
- **filepath**: Ruta del fichero detectado.
- **endpoint_uuid**: Identificador unívoco del EndPoint. Este parámetro se usa para la parte web de redBorder Malware.
- **sha256**: Hash SHA256 del fichero, es decir, resumen de 256 bits del contenido del fichero.

En caso de ser un fichero ya conocido por el sistema de reputación, lo único que cambia es el score del fichero. Ese hecho, se debe a que el fichero ya ha sido analizado por el manager y mediante es sistema de reputación, el agente EndPoint ha actualizado la caché local.


```
{
        "cache_score": 0,
        "action": "updated",
        "namespace_uuid": "C.91c40668ddea3975",
        "filename": "setup.exe",
        "type": "file_capture",
        "file_type": "MSEXE",
        "acl": 32805760,
        "timestamp": 1465424338,
        "file_extension": ".exe",
        "md5": "4fe45b73cb23e4e77fcf6463f93c85fb",
        "filepath": "C:\\Program Files (x86)\\Google\\Chrome\\Application\\51.0.2704.84\\Installer\\setup.exe",
        "endpoint_uuid": "C.91c40668ddea3975",
        "sha256": "d42a7b102ccbbb69a1074a8d65fe3c41e3d7389503c10f10238b48a6667e6d2f"
}
```


Figura 35 Evento de detección de un fichero conocido

En la tarea de captura de ficheros inexistentes, es decir, ficheros borrados o renombrados, se observó que se generan diferentes eventos para cada caso. En caso de ser un fichero borrado el evento es del siguiente tipo:


```
{
        "filename": "C:\\Windows\\SoftwareDistribution\\DataStore\\Logs\\tmp.edb",
        "action": "deleted",
        "endpoint_uuid": "C.91c40668ddea3975",
        "namespace_uuid": "C.91c40668ddea3975",
        "type": "file_information"
}
```


Figura 36 Evento de detección de un fichero borrado

El otro caso de ficheros inexistentes que puede ocurrir, es que un fichero se renombre, siendo el evento del siguiente tipo:

```
{
        "type": "file_information",
        "filename": "C:\\Windows\\System32\\wbem\\Performance\\WmiApRpl.h",
        "namespace_uuid": "C.91c40668ddea3975",
        "action": "file C:\\Windows\\System32\\wbem\\Performance\\WmiApRpl_new.h renamed to C:\\Windows\\System32\\wbem\\Performance\\WmiApRpl.h",
        "endpoint_uuid": "C.91c40668ddea3975"
}
```


Figura 37 Evento de detección de un fichero renombrado

Por otro lado, existen los eventos que se envían a este topic desde las tareas de los procesos, que son tres, los relacionados con los procesos creados, los destruidos y las conexiones web.

El primero de ellos es un evento del siguiente tipo:


```
{
        "timestamp": 1471855654,
        "sessionId": 2,
        "namespace_uuid": "C.91c40668ddea3975",
        "action": "creation",
        "parent_application": "C:\\Windows\\Explorer.EXE",
        "priority": 8,
        "type": "process",
        "application": "C:\\Windows\\system32\\rundll32.exe",
        "name": "rundll32.exe",
        "pid": 3472,
        "endpoint_uuid": "C.91c40668ddea3975"
}
```


Figura 38 Evento de detección de un proceso creado

En este tipo de eventos, existen otros parámetros distintos a los eventos de los ficheros, los cuales se explican a continuación:

- **timestamp**: Marca de tiempo del instante de detección del proceso.
- **sessionId**: Identificador numérico del perfil de usuario que ejecuta el proceso.
- **namespace_uuid**: Identificador del EndPoint emisor del evento. Este parámetro se usa para la parte de Big Data en el manager de redBorder Malware.
- **endpoint_uuid**: Identificador unívoco del EndPoint. Este parámetro se usa para la parte web de redBorder Malware.
- **type**: Tipo del evento, de proceso en este caso.
- **action**: Acción del tipo del evento. Como se observa, en este caso es de creación de un proceso.
- **application**: Ruta del fichero que se ha ejecutado, provocando la creación del proceso.
- **parent_application**: Ruta del fichero que ha ejecutado al archivo referenciado en el parámetro "application", provocando la creación de un proceso hijo, por tanto, este fichero es el que ha provocado el proceso padre.
- **name**: Nombre del fichero que se ha ejecutado, provocando la creación del proceso.
- **pid**: Identificador numérico y unívoco del proceso.
- **priority**: prioridad (1-13) en la ejecución del proceso.

El segundo tipo de eventos relacionados con procesos, son aquellos que hacen referencia a los procesos destruidos, generando un evento como el siguiente:


```
{
        "namespace_uuid": "C.91c40668ddea3975",
        "type": "process",
        "timestamp": 1471855657,
        "name": "rundll32.exe",
        "pid": 3472,
        "action": "deletion",
        "endpoint_uuid": "C.91c40668ddea3975"
}
```


Figura 39 Evento de detección de un proceso destruido

El último de los eventos de los procesos, son aquellos que tienen que ver con una conexión de red. Como se explicó en la sección 4.2.3.2.7, las conexiones son detectadas a partir de los procesos creados, ya que cuando se incia una conexión, lo hace a través de un proceso, generando en el agente EndPoint el siguiente tipo de evento:


```
{
        "connections": "91.126.247.83:443",
        "type": "connection",
        "namespace_uuid": "C.91c40668ddea3975",
        "application": "chrome.exe",
        "timestamp": 1471501731,
        "pid": 3584,
        "endpoint_uuid": "C.91c40668ddea3975"
}
```


Figura 40 Evento de detección de una conexión de red

Observando la imagen anterior, existe un nuevo parámetro llamado “connections”, que muestra el servidor al cual se ha conectado un EndPoint y por que puerto lo ha hecho. De igual forma, si se observa el fichero responsable de la conexión, se dictamina que la conexión se ha producido a través de un proceso de Google Chrome.

Todos los eventos mostrados con anterioridad, son enviados al manager mediante el gestor de Kafka haciendo uso del topic “rb_ioc”. El motivo de enviarlo a este topic, no es otro que comparar todos estos eventos contra IOC para detectar cualquier malware, ataque o incidencia que pueda estar ocurriendo en un EndPoint.

Los parámetros de los eventos, están adaptados para los IOC, de forma que, una vez recibido el evento, la comparación sea inmediata, sin necesidad de adaptación.

4.3.1 rb_endpoint

Es el otro topic que se usa en el agente EndPoint para enviar eventos al manager, concretamente, se usa en las tareas relacionadas con la detección y captura de ficheros. Estas tareas generan dos tipos de eventos, uno de ficheros nuevos detectados y otro de ficheros conocidos o vistos con anterioridad, ya estén filtrados por tipo o tamaño o no lo estén.

Los eventos relacionados con ficheros desconocidos por el sistema de reputación son los siguientes:


```
{
        "endpoint_uuid": "C.91c40668ddea3975",
        "timestamp": 1471503536,
        "action": "forward",
        "file_name": "setup.exe",
        "file_uri": "C:\\Program Files (x86)\\Google\\Chrome\\Application\\51.0.2704.84\\Installer\\setup.exe",
        "hash": "d42a7b102ccbbb69a1074a8d65fe3c41e3d7389503c10f10238b48a6667e6d2f"
        "file_size": 11534336
        "hash_probe_score": -1
        "sensor_name": "redBorder-PC"
        "client_mac": "52-16-93-0D-4B-5H"
        "domain_name": "redBorder.lan"
        "endpoint_agent": "10.0.106.71"
        "dst": "10.0.106.71"
        "sensor_ip": "10.0.106.71"
        "src_net_name":  "255.255.255.0"
        "client_id": "redBorder-PC\\EndPoint01"
}
```


Figura 41 Evento de detección de un fichero desconocido en rb_endpoint

Los parámetros que aparecen en este tipo de eventos son ligeramente distintos a los explicados para el topic "rb_ioc", y se explican a continuación:

- **endpoint_uuid**: Identificador unívoco del EndPoint. Este parámetro se usa para la parte web de redBorder Malware.
- **namespace_uuid**: Identificador del EndPoint emisor del evento. Este parámetro se usa para la parte de Big Data en el manager de redBorder Malware.
- **timestamp**: Marca de tiempo del instante de detección del fichero.
- **hash**: Hash SHA256 del fichero, es decir, resumen de 256 bits del contenido del fichero.
- **action**: Tipo de acción ejercida por el agente EndPoint al detectar el fichero. En este caso es "forward", ya que el evento es reenviado directamente al manager para que lo analice.
- **file_name**: Nombre del fichero detectado.
- **file_uri**: Ruta del fichero detectado.
- **file_size**: Tamaño del fichero en bytes.
- **hash_probe_score**: Score que posee el fichero en el sistema de reputación. En este caso, al ser un fichero desconocido por el mismo, el valor es -1.
- **sensor_name**: Nombre del EndPoint.
- **client_mac**: MAC del EndPoint que envía el evento.
- **domain_name**: Dominio de red donde se encuentra el EndPoint.
- **endpoint_agent**, **dst**, **sensor_ip**: Son valores que se usan en diferentes tareas del manager de redBorder Malware, por tanto, cada tarea necesita el valor con un nombre distinto, pero todos se refieren a la IP del EndPoint.
- **src_net_name:** IP de la subred del EndPoint**.**
- **client_id:** Perfil de usuario donde se ha detectado el fichero.

Los eventos de este topic son siempre iguales, lo único que puede variar es el valor del score, ya que se puede volver a detectar el mismo fichero, pero que esta vez, el fichero ya se haya analizado en el manager de redBorder Malware y se haya actualizado el sistema de reputación, a partir del cual, el agente EndPoint se haya descargado el incremento correspondiente, actualizando así la caché local.

En ese caso, el evento variaría tal como se muestra en el siguiente ejemplo:


```
{
        "endpoint_uuid": "C.91c40668ddea3975",
        "timestamp": 1471503536,
        "action": "forward",
        "file_name": "setup.exe",
        "file_uri": "C:\\Program Files (x86)\\Google\\Chrome\\Application\\51.0.2704.84\\Installer\\setup.exe",
        "hash": "d42a7b102ccbbb69a1074a8d65fe3c41e3d7389503c10f10238b48a6667e6d2f"
        "file_size": 11534336
        "hash_probe_score": 26
        "sensor_name": "redBorder-PC"
        "client_mac": "52-16-93-0D-4B-5H"
        "domain_name": "redBorder.lan"
        "endpoint_agent": "10.0.106.71"
        "dst": "10.0.106.71"
        "sensor_ip": "10.0.106.71"
        "src_net_name":  "255.255.255.0"
        "client_id": "redBorder-PC\\EndPoint01"
}
```


Figura 42 Evento de detección de un fichero conocido en rb_endpoint

5 INTEGRACIÓN EN REDBORDER MALWARE.

La integración de redBorder Malware EndPoint se ha hecho de una forma bastante sencilla para poder integrarlo como un elemento más dentro de la aplicación de malware de redBorder.

Se ha implementado un instalador de Microsoft Windows (MSI) que instale todas las necesidades junto a este proyecto, y el cliente de GRR. De esta manera, con el servidor de GRR en el manager, se tiene un mecanismo de comunicación con el EndPoint para poder actuar en caso de que se detecte cualquier anomalía o incidencia.

La arquitectura final queda de la siguiente forma:

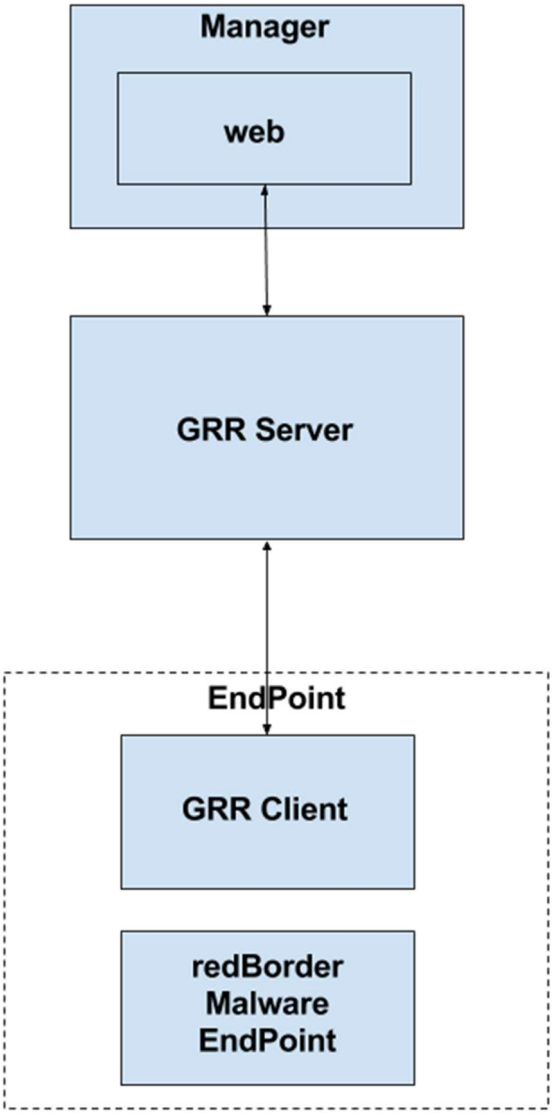


Figura 43 Arquitectura de integración de redBorder Malware EndPoint

Una vez se tiene el instalador MSI, se introduce en la web de redBorder Malware para poder ser instalado en diferentes EndPoints.

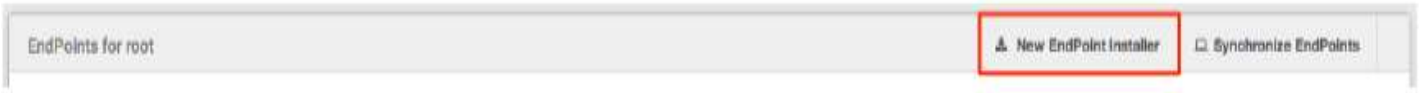


Figura 44 Descarga de redBorder Malware EndPoint desde la web

Para descargarlo, se hace desde la web, que una vez pulsado el botón mostrado en la imagen anterior, pide la arquitectura sobre la cual va a ser instalado.

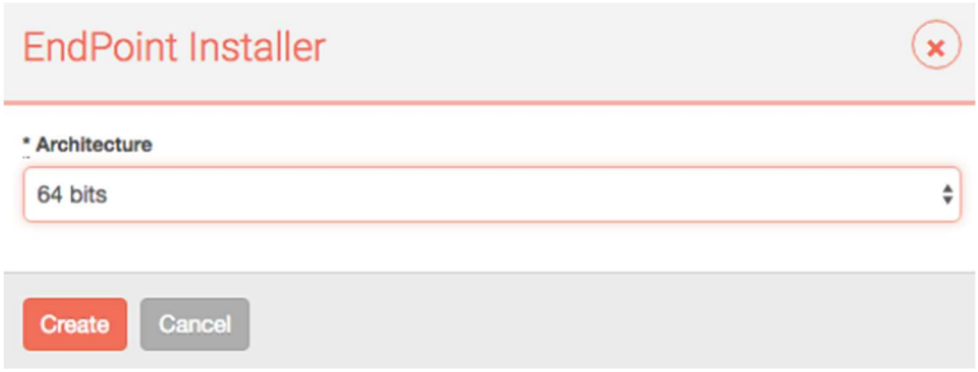


Figura 45 Elección de la arquitectura de redBorder Malware EndPoint

Una vez descargado en un EndPoint, se ejecuta el instalador, que instalará el cliente de GRR y redBorder Malware EndPoint. Una vez instalado, el cliente GRR lanza un call back al servidor para decirle que ha sido instalado, dando los datos del EndPoint. Esos datos, son cogidos por parte del manager de redBorder para reconocer como un elemento más de la red al EndPoint donde se ha instalado.

Una vez realizado el paso anterior, el manager extrae del servidor el identificador que el servidor GRR le ha dado al cliente y mediante el propio servidor, se le comunica a redBorder Malware EndPoint cual es su identificador, añadiéndolo al fichero "parameters.yaml" que se explico en la sección 4.2. Posteriormente, inicia el servicio "EndPoint service", que es el servicio generado por este proyecto.

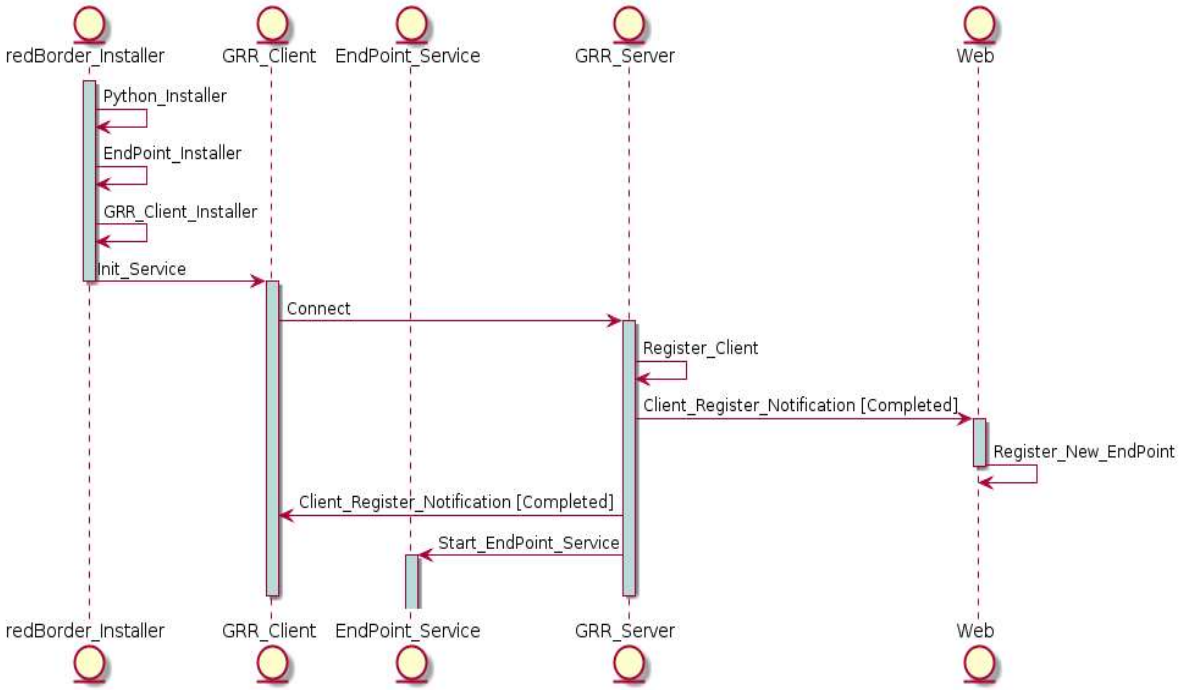


Figura 46 Diagrama de instalación

Con el servicio arrancado, se tiene el EndPoint registrado en el manager y funcionando como se muestra en la siguiente imagen, por tanto, la integración con redBorder Malware se ha completado.

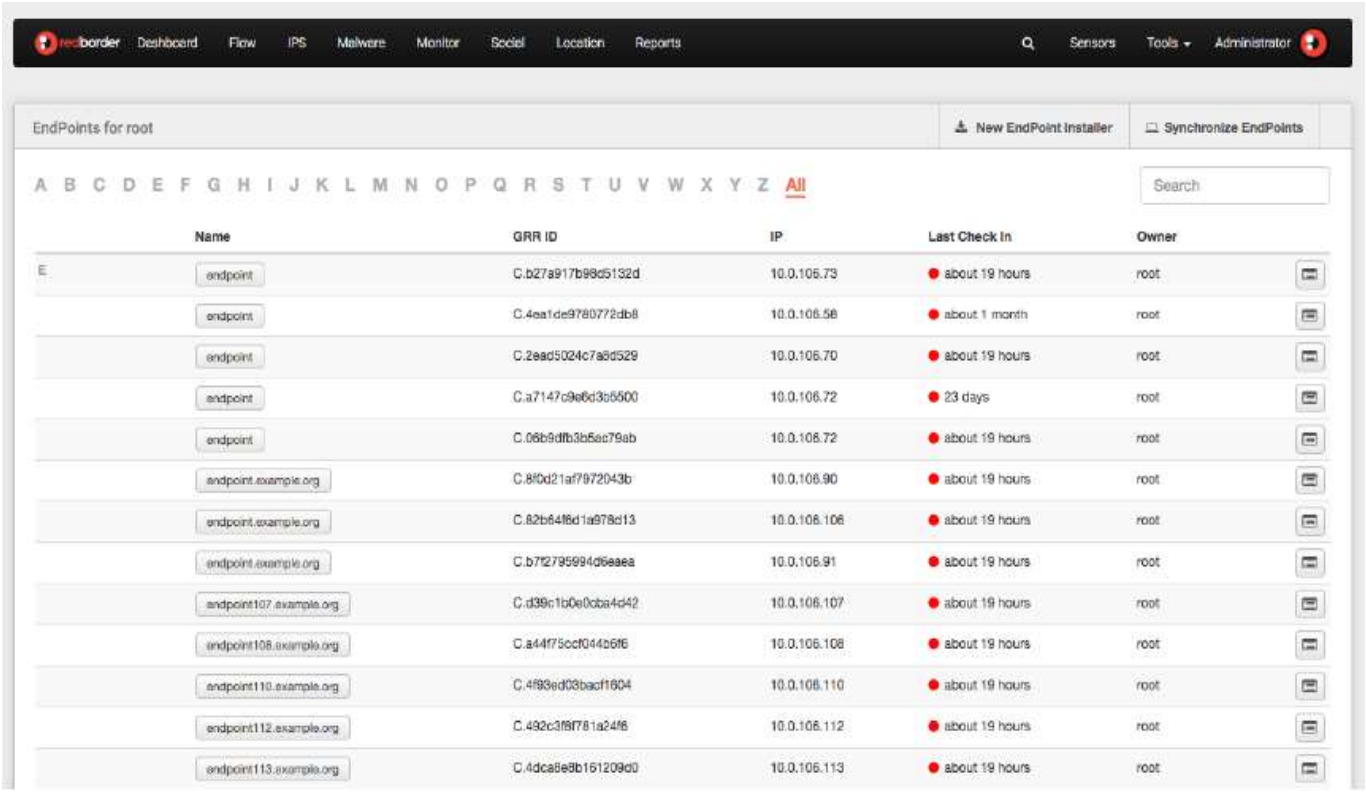


Figura 47 Lista de EndPoints registrados

6 PRUEBAS DE CONCEPTO.

6.1 Pruebas de concepto y funcionalidad

En este apartado se van a realizar diferentes pruebas con el fin de demostrar que el proyecto funciona de forma correcta y cumple cada una de las funcionalidades para las cuales ha sido diseñado.

6.1.1 Instalación y registro

La primera prueba va a consistir en descargarse vía web redBorder Malware EndPoint, instalarlo en un EndPoint y verificar que el registro en la web de redBorder Malware se ha realizado de forma correcta. De forma adicional, se va a comprobar como los servicios, tanto de GRR como del EndPoint, están activos y como el software se inicia de forma correcta.

Para comenzar la prueba se abre la web del manager de redBorder Malware, cuya dirección es https://<IP>/endpoints y se descarga el software.

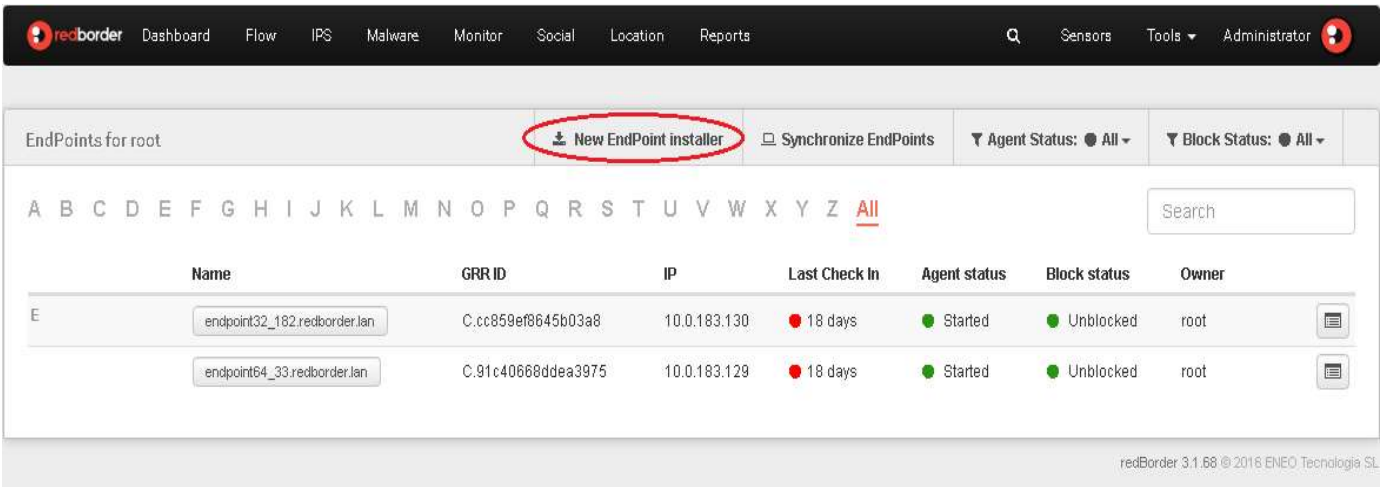


Figura 48 Descarga de redBorder Malware EndPoint vía web

Una vez descargado, se abre en el EndPoint el directorio, y se ejecuta el instalador, que instala el cliente GRR y el software, ambos como servicios de Windows. Una vez instalado, el cliente de GRR lanza un callback al servidor para registrarse, a lo que este último responde con un identificador. Los datos, tanto los de registro como los del identificador, son extraidos por el manager para hacer el registro del EndPoint en la web y para que, vía GRR, se lance un flow [11] al cliente para introducir el identificador en el fichero del entorno de configuración e iniciar el servicio EndPoint.

En la siguiente imagen se aprecia el instalador que se debe ejecutar:

[11] Flujo de comunicación servidor – cliente de GRR, por el cual el servidor hace una petición al cliente.

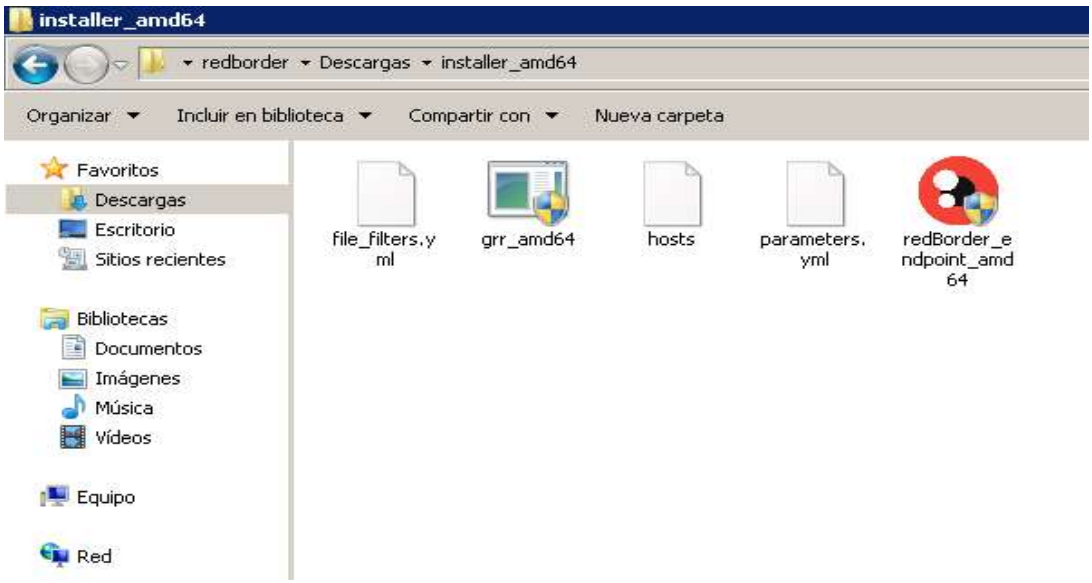


Figura 49 Instalador

Una vez instalado, se puede apreciar como crea tanto el servicio de GRR (GRR Monitor) como el servicio de EndPoint (EndPoint Service):

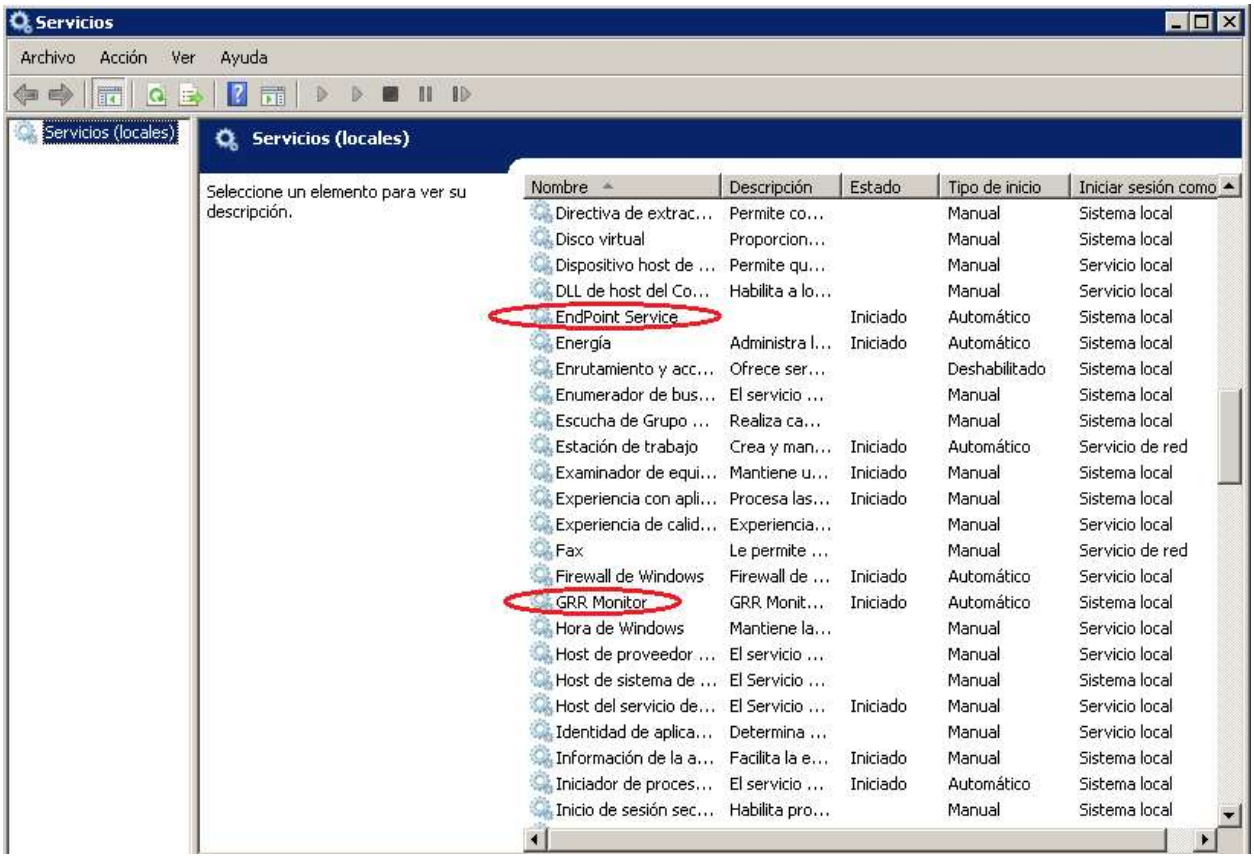


Figura 50 Servicios instalados

El siguiente paso, como se explico con anterioridad, es demostrar que el EndPoint se registra de forma correcta en la web de redBorder Malware. Para ello, el servicio “GRR Monitor” manda un callback al servidor para registrarse, asignándole este un identificador. Tanto el callback como el identificador son extraidos por parte del manager para registrarlo en la web, momento en el cual ya aparece como un nuevo EndPoint, como puede apreciarse a continuación:

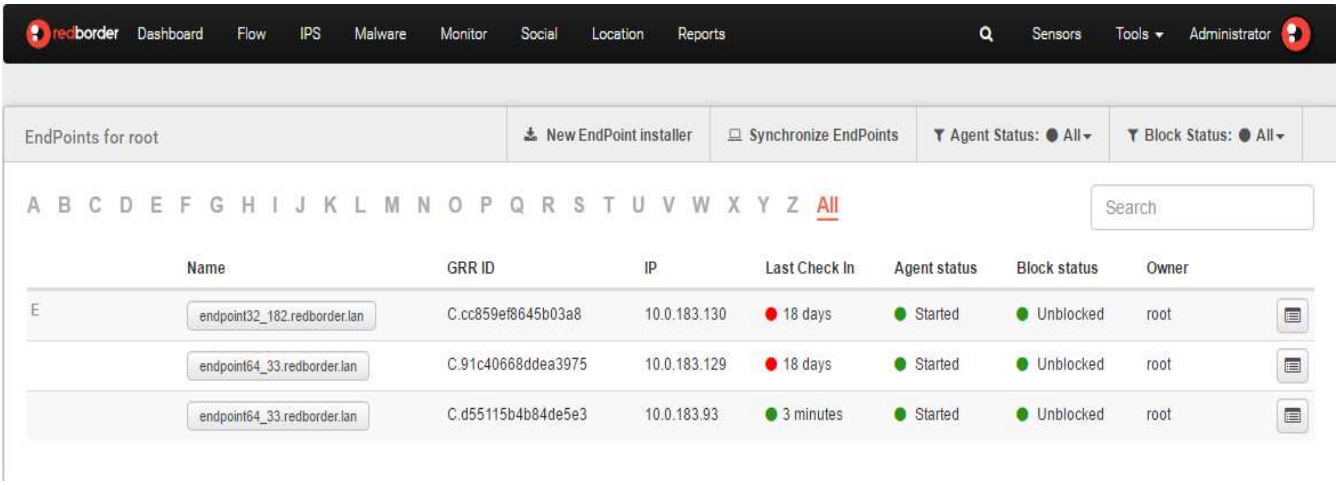


Figura 51 Registro web de un EndPoint

Por último, el manager usa la comunicación entre servidor-cliente de GRR para lanzar un flow a través de esta tecnología para introducir el identificador en el fichero “parameters.yaml” , dentro del directorio “config” del entorno de redBorder Malware EndPoint. A continuación, se inicia el servicio “EndPoint service”.

```
---
kafka_server: kafka.redborder.cluster:9092
s3_bucket: malware
s3_key: "/mdata/input/"
access_key: <ACCESS_KEY>
secret_key: <SECRET_KEY>
reputation_server: rb-reputation.redborder.cluster
file_size: 134217728
version_cache: v1
domain_id: 1
score: 1
client_id: C.d55115b4b84de5e3
```

Figura 52 Parámetro client_id

Para comprobar el correcto inicio del servicio, se pueden consultar los servicios y comprobar que está en estado “Iniciado”, como se demuestra en la imagen mostrada anteriormente de los servicios. En caso de algún tipo de error, el servicio estaría parado.

Realizadas las pruebas, se observa que todo ha transcurrido de forma correcta, por tanto, la instalación y registro de redBorder Malware EndPoint, ha sido un éxito.

6.1.2 Detección y captura de ficheros

La realización de esta prueba, va a estar orientada a la demostración de la detección y captura de ficheros, la subida de los mismos a Riak, el envío del evento mediante Kafka y su posterior aparición en la web de redBorder Malware.

El primer paso a dar es crear un fichero, al cual se da un nombre cualquier, en este caso, "FILE.exe".

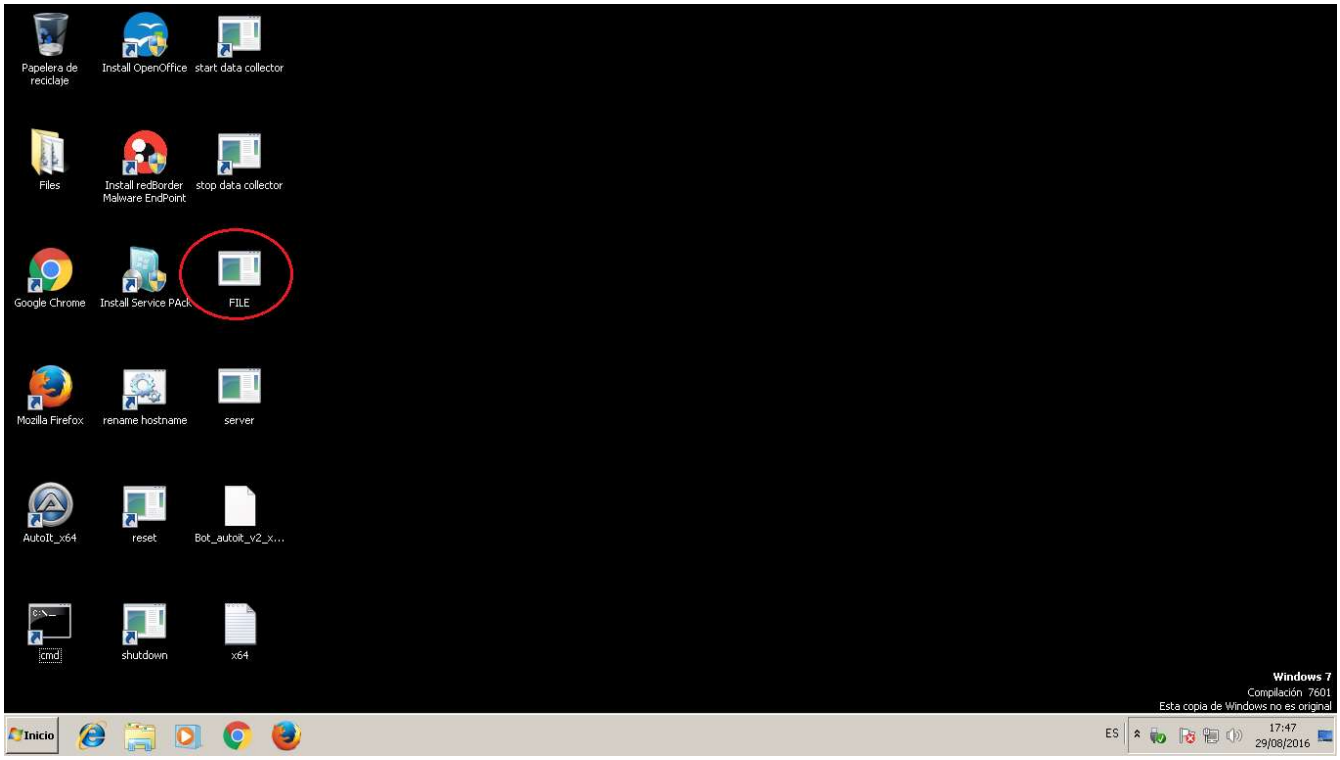


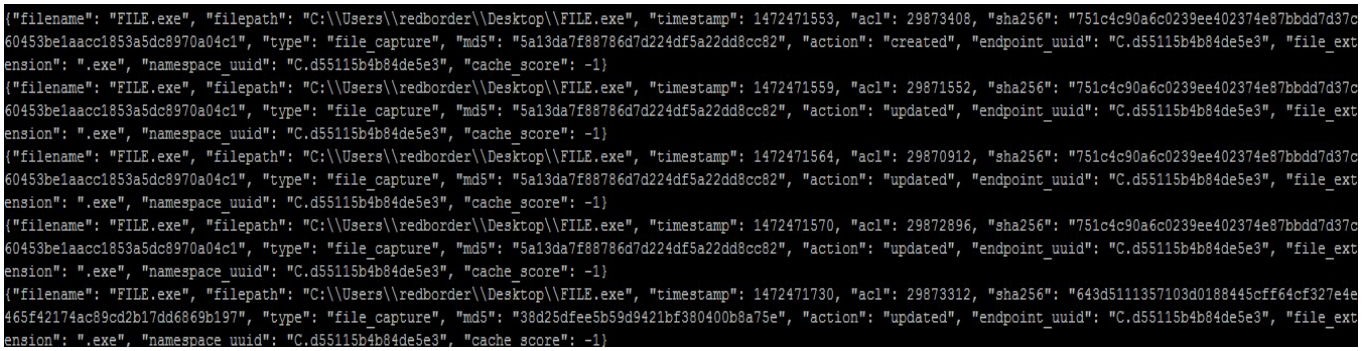
Figura 53 Creación de un fichero

A continuación, para comprobar si se ha detectado el fichero, se abre desde el manager de redBorder Malware una consola y se inicia el consumidor de Kafka en el topic "rb_endpoint":


```
{"sensor_ip": "10.0.183.93", "timestamp": 1472471553, "client_id": "ENDPOINT64_33\\redborder", "file_uri": "C:\\Users\\redborder\\Desktop\\FILE.exe", "endpoint_agent":
"10.0.183.93", "src_net_name": "255.255.255.0", "file_size": 32, "hash_probe_score": -1, "action": "forward", "file_name": "FILE.exe", "hash": "751c4c90a6c0239ee402374e
87bbdd7d37c60453be1aacc1853a5dc8970a04c1", "endpoint_uuid": "C.d55115b4b84de5e3", "sensor_name": "endpoint64_33", "domain_name": "redborder.lan", "dst": "10.0.183.93",
"client_mac": "00:50:56:82:81:c2"}
{"sensor_ip": "10.0.183.93", "timestamp": 1472471559, "client_id": "ENDPOINT64_33\\redborder", "file_uri": "C:\\Users\\redborder\\Desktop\\FILE.exe", "endpoint_agent":
"10.0.183.93", "src_net_name": "255.255.255.0", "file_size": 32, "hash_probe_score": -1, "action": "forward", "file_name": "FILE.exe", "hash": "751c4c90a6c0239ee402374e
87bbdd7d37c60453be1aacc1853a5dc8970a04c1", "endpoint_uuid": "C.d55115b4b84de5e3", "sensor_name": "endpoint64_33", "domain_name": "redborder.lan", "dst": "10.0.183.93",
"client_mac": "00:50:56:82:81:c2"}
{"sensor_ip": "10.0.183.93", "timestamp": 1472471564, "client_id": "ENDPOINT64_33\\redborder", "file_uri": "C:\\Users\\redborder\\Desktop\\FILE.exe", "endpoint_agent":
"10.0.183.93", "src_net_name": "255.255.255.0", "file_size": 32, "hash_probe_score": -1, "action": "forward", "file_name": "FILE.exe", "hash": "751c4c90a6c0239ee402374e
87bbdd7d37c60453be1aacc1853a5dc8970a04c1", "endpoint_uuid": "C.d55115b4b84de5e3", "sensor_name": "endpoint64_33", "domain_name": "redborder.lan", "dst": "10.0.183.93",
"client_mac": "00:50:56:82:81:c2"}
{"sensor_ip": "10.0.183.93", "timestamp": 1472471570, "client_id": "ENDPOINT64_33\\redborder", "file_uri": "C:\\Users\\redborder\\Desktop\\FILE.exe", "endpoint_agent":
"10.0.183.93", "src_net_name": "255.255.255.0", "file_size": 32, "hash_probe_score": -1, "action": "forward", "file_name": "FILE.exe", "hash": "751c4c90a6c0239ee402374e
87bbdd7d37c60453be1aacc1853a5dc8970a04c1", "endpoint_uuid": "C.d55115b4b84de5e3", "sensor_name": "endpoint64_33", "domain_name": "redborder.lan", "dst": "10.0.183.93",
"client_mac": "00:50:56:82:81:c2"}
```


Figura 54 Fichero nuevo en rb_endpoint

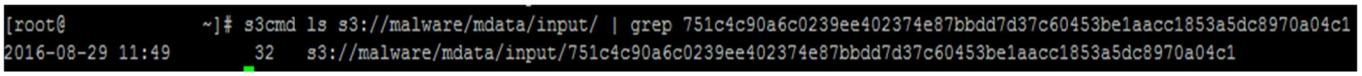
Se observa que los eventos llegan de forma correcta, por tanto, se comprueba el topic “rb_ioc” para asegurar que llegan los eventos y que, en caso de amenazas, sean analizados por los IOC:



{"filename": "FILE.exe", "filepath": "C:\\Users\\redborder\\Desktop\\FILE.exe", "timestamp": 1472471553, "acl": 29873408, "sha256": "751c4c90a6c0239ee402374e87bbdd7d37c
60453be1aacc1853a5dc8970a04c1", "type": "file_capture", "md5": "5a13da7f88786d7d224df5a22dd8cc82", "action": "created", "endpoint_uuid": "C.d55115b4b84de5e3", "file_ext
ension": ".exe", "namespace_uuid": "C.d55115b4b84de5e3", "cache_score": -1}
{"filename": "FILE.exe", "filepath": "C:\\Users\\redborder\\Desktop\\FILE.exe", "timestamp": 1472471559, "acl": 29871552, "sha256": "751c4c90a6c0239ee402374e87bbdd7d37c
60453be1aacc1853a5dc8970a04c1", "type": "file_capture", "md5": "5a13da7f88786d7d224df5a22dd8cc82", "action": "updated", "endpoint_uuid": "C.d55115b4b84de5e3", "file_ext
ension": ".exe", "namespace_uuid": "C.d55115b4b84de5e3", "cache_score": -1}
{"filename": "FILE.exe", "filepath": "C:\\Users\\redborder\\Desktop\\FILE.exe", "timestamp": 1472471564, "acl": 29870912, "sha256": "751c4c90a6c0239ee402374e87bbdd7d37c
60453be1aacc1853a5dc8970a04c1", "type": "file_capture", "md5": "5a13da7f88786d7d224df5a22dd8cc82", "action": "updated", "endpoint_uuid": "C.d55115b4b84de5e3", "file_ext
ension": ".exe", "namespace_uuid": "C.d55115b4b84de5e3", "cache_score": -1}
{"filename": "FILE.exe", "filepath": "C:\\Users\\redborder\\Desktop\\FILE.exe", "timestamp": 1472471570, "acl": 29872896, "sha256": "751c4c90a6c0239ee402374e87bbdd7d37c
60453be1aacc1853a5dc8970a04c1", "type": "file_capture", "md5": "5a13da7f88786d7d224df5a22dd8cc82", "action": "updated", "endpoint_uuid": "C.d55115b4b84de5e3", "file_ext
ension": ".exe", "namespace_uuid": "C.d55115b4b84de5e3", "cache_score": -1}
{"filename": "FILE.exe", "filepath": "C:\\Users\\redborder\\Desktop\\FILE.exe", "timestamp": 1472471730, "acl": 29873312, "sha256": "643d5111357103d0188445cff64cf327e4e
465f42174ac89cd2b17dd6869b197", "type": "file_capture", "md5": "38d25dfee5b59d9421bf380400b8a75e", "action": "updated", "endpoint_uuid": "C.d55115b4b84de5e3", "file_ext
ension": ".exe", "namespace_uuid": "C.d55115b4b84de5e3", "cache_score": -1}

Figura 55 Fichero nuevo en rb_ioc

Al igual que antes, los eventos llegan sin problemas al manager, por tanto, sólo quedan dos pasos pendientes. El primero de ellos es, a partir del hash del fichero, cerciorarse que el fichero se encuentra en Riak. Por tanto, desde la misma consola donde se han comprobado los eventos, se comprueba si el fichero se encuentra en Riak, hecho que podemos observar en la siguiente imagen:



[root@ ~]# s3cmd ls s3://malware/mdata/input/ | grep 751c4c90a6c0239ee402374e87bbdd7d37c60453be1aacc1853a5dc8970a04c1
2016-08-29 11:49 32 s3://malware/mdata/input/751c4c90a6c0239ee402374e87bbdd7d37c60453be1aacc1853a5dc8970a04c1

Figura 56 Fichero nuevo en Riak

El segundo paso es demostrar vía web que los ficheros están siendo analizados. Para ello, es necesario observar si existe algún fichero lanzado desde el equipo “adrian-HP-Notebook.redborder.lan” en la vista Top de Malware de la web.

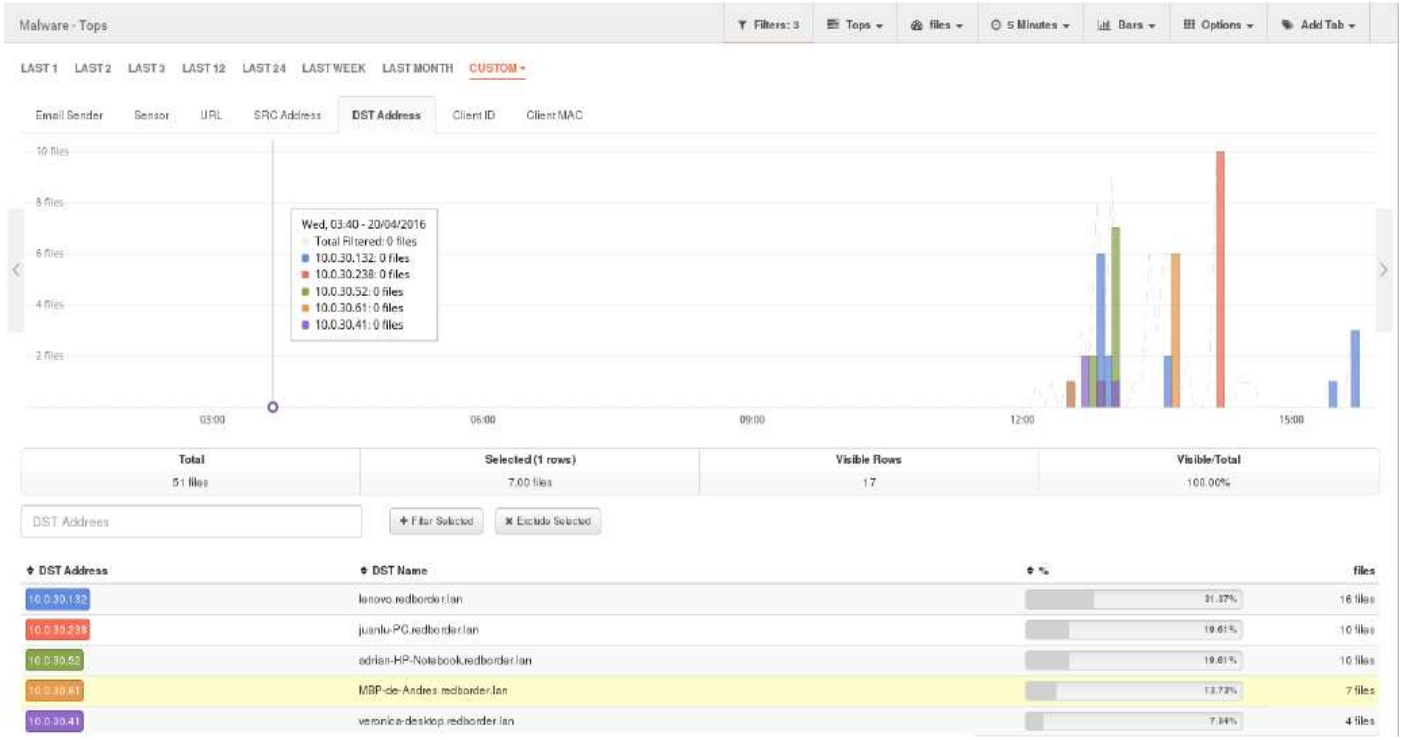


Figura 57 Fichero nuevo en la web

El equipo anteriormente mencionado, ha lanzado varias muestras, entre las que se encuentra el fichero FILE.exe, por tanto, esta prueba ha sido realizada con éxito.

6.1.3 Detección de procesos

Esta prueba es bastante más sencilla que la anterior, ya que la comprobación acerca de la detección de un proceso sólo se comprueba a través del consumidor de Kafka en el topic "rb_ioc". Para ello, primero se abre un proceso cualquiera, como puede ser "notepad":

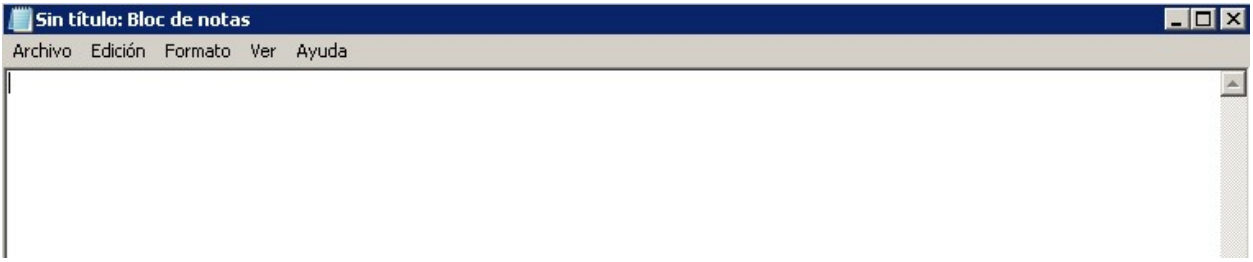


Figura 58 Creación del proceso notepad

Acto seguido, se cierra el proceso y se comprueba, mediante la consola del manager, si en el consumidor existen eventos de Kafka relacionados con dicho proceso. Para ello, se miran los eventos llegados al topic "rb_ioc". Como puede comprobarse a continuación, la prueba resulta satisfactoria:

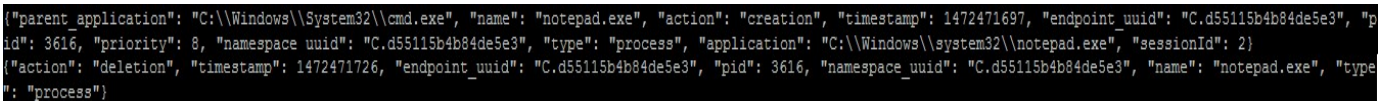


Figura 59 Evento de creación del proceso notepad en Kafka

6.1.4 Detección de conexiones web

En esta prueba se va a realizar una conexión a una página web y se va a comprobar mediante un consumidor de Kafka que el evento llega al manager de forma correcta.

Por tanto, lo primero que se debe hacer es abrir una conexión web, desde Google Chrome, por ejemplo:

Figura 60 Creación de una conexión web

Posteriormente, desde el consumidor Kafka del manager de redBorder Malware, se comprueba el topic de "rb_ioc" para asegurar si ha llegado o no el evento referente a la conexión anterior:

{"connections": "193.110.128.110:80; 80.252.91.53:80; 46.4.104.106:80; 31.186.225.24:80; 91.126.247.7:80; 195.81.202.113:80; 46.105.114.234:80; 23.62.145.125:80; 23.38
44.123:80; 91.126.247.9:80; 52.31.123.4:443; 54.210.87.134:80; 216.58.214.162:80; 151.101.61.108:80; 216.58.201.134:443; 93.184.220.187:80; 37.252.162.201:80; 209.197.
.19:443; 216.58.210.134:443; 173.222.98.253:443; 23.34.176.143:80; 216.58.214.162:443; 195.81.202.118:80; 52.8.198.99:80; 62.73.184.148:80; 216.58.210.161:80; 54.175.1
2.174:80; 54.76.227.214:443; 216.58.201.130:443; 104.98.148.20:443; 216.58.210.162:443; 216.58.201.130:80; 185.33.222.187:80; 54.243.100.45:80; 54.173.156.189:80; 151.
01.60.175:80; 66.235.148.133:80; 104.16.53.4:80; 87.98.228.78:80; 193.0.160.206:80; 54.217.251.51:80; 216.58.210.162:80; 52.58.124.101:80; 178.250.2.100:80; 204.11.109
66:80; 77.238.185.35:443; 193.110.128.109:80; 173.231.180.110:80; 54.230.62.81:80; 54.208.226.181:80; 31.186.225.23:80; 62.73.184.138:80; 208.146.36.47:80; 91.126.247.
1:443; 81.20.72.187:80; 198.54.12.127:80; 46.51.171.153:80; 54.93.160.134:443; 31.186.225.24:443; 31.13.93.36:443; 185.33.220.27:80; 159.253.128.188:80; 216.58.210.161
443; 94.46.159.28:80; 23.201.173.188:443; 64.95.32.47:80; 216.58.214.170:443; 185.29.133.58:80; 207.46.194.10:80; 52.28.65.105:80; 50.16.234.101:80; 52.58.23.63:443; 5
.246.132.55:80; 91.126.247.98:443; 37.157.2.24:80; 54.171.221.167:80; 46.228.164.11:80; 152.163.51.2:80; 52.3.189.203:80; 173.192.220.64:80; 46.137.127.140:80; 91.126.
47.102:443; 70.42.146.149:80; 104.16.53.4:443; 37.252.163.93:80; 178.250.2.67:443; 208.146.36.28:80; 52.85.47.166:443; 91.126.247.108:443; 37.252.162.201:443; ", "time
tamp": 1472473013, "endpoint uuid": "C.d55115b4b84de5e3", "pid": 3952, "namespace uuid": "C.d55115b4b84de5e3", "type": "connection", "application": "chrome.exe"}

Figura 61 Evento de creación del proceso de conexión web

Se puede observar como la detección y el envío del evento a Kafka ha sido un éxito, ya que se muestra que el proceso es "chrome.exe" y todos los saltos que ha dado hasta llegar al servidor web destino.

6.1.5 Detección mediante IOC de amenazas

Para llevar a cabo esta prueba, sólo es necesario saber que proceso o fichero, puede activar una alerta por IOC. En este caso, se ha diseñado una alerta, a modo de ejemplo, que salta al detectar la creación de un fichero llamado "server.exe" en un EndPoint. Por tanto, el primer paso es crear un fichero con ese nombre, como se hace a continuación:

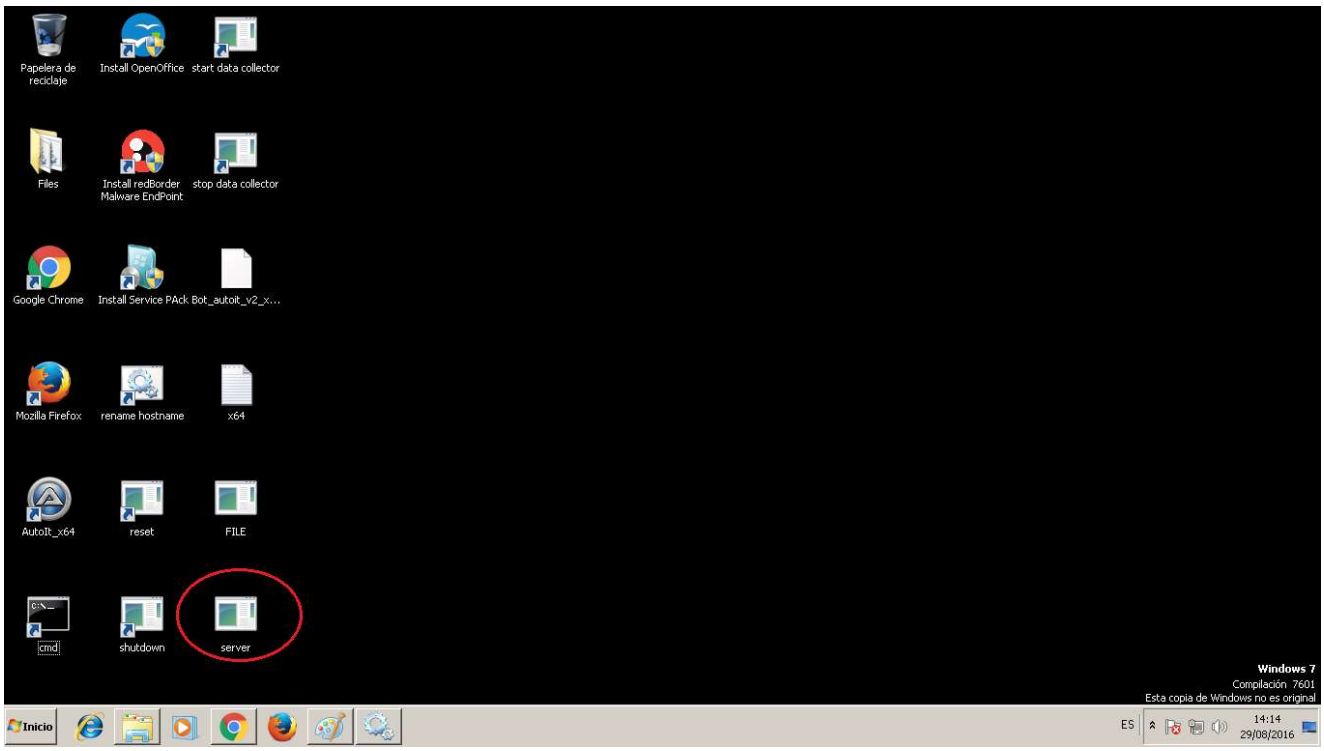


Figura 62 Creación de una alerta de IOC

Acto seguido, se comprueba en la vista de IOC de la web de redBorder Malware si se ha detectado alguna alerta de IOC. Se observa a continuación, que la alerta salta debido a que la comparación del evento contra los IOC detecta el fichero “server.exe”.

Figura 63 Alerta de IOC

6.1.6 Creación del histórico de eventos

La prueba es sencilla, ya que desde la web de redBorder Malware, se puede descargar el histórico de eventos sin ningún tipo de problemas.

Para descargarse el fichero, es necesario ir a la dirección https://IPmanager/endpoints y pinchar sobre alguno de ellos. Una vez dado el paso anterior, tal como se muestra en la siguiente imagen, se puede:

- Parar/Reanudar el agente EndPoint
- Bloquear las conexiones de entrada y salida del EndPoint
- Descargar el histórico de eventos.

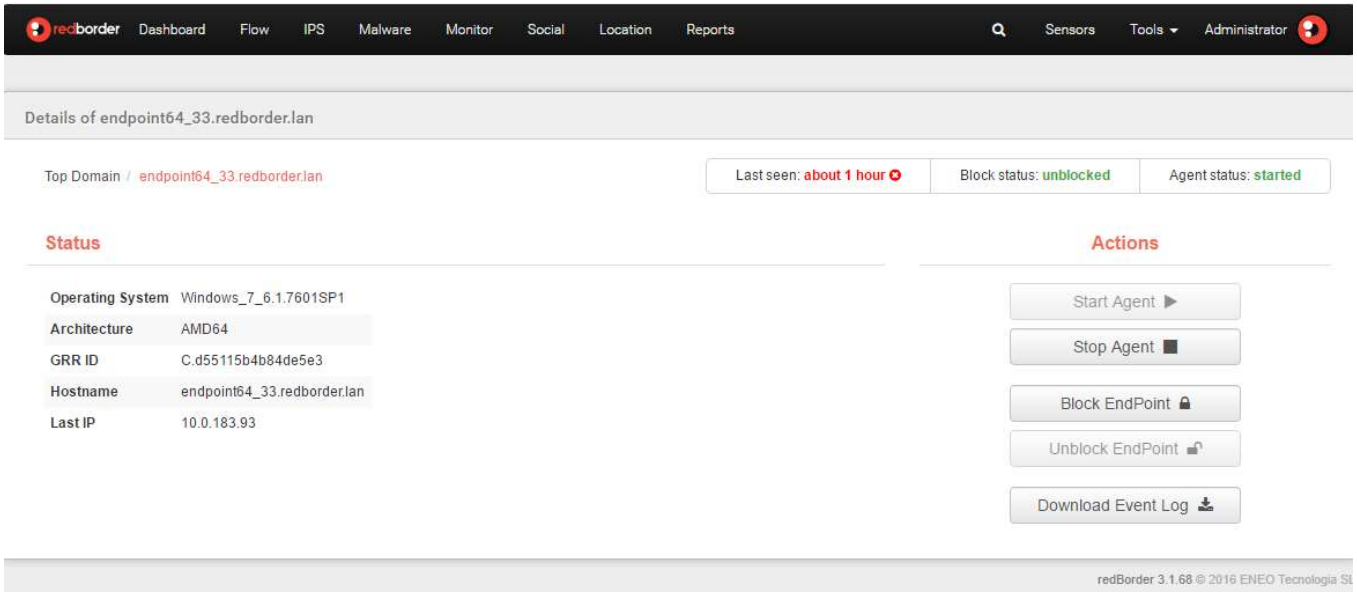


Figura 64 Descarga del histórico de eventos

Una vez descargado, solo es necesario abrirlo con un editor de texto y comprobar el contenido del mismo.

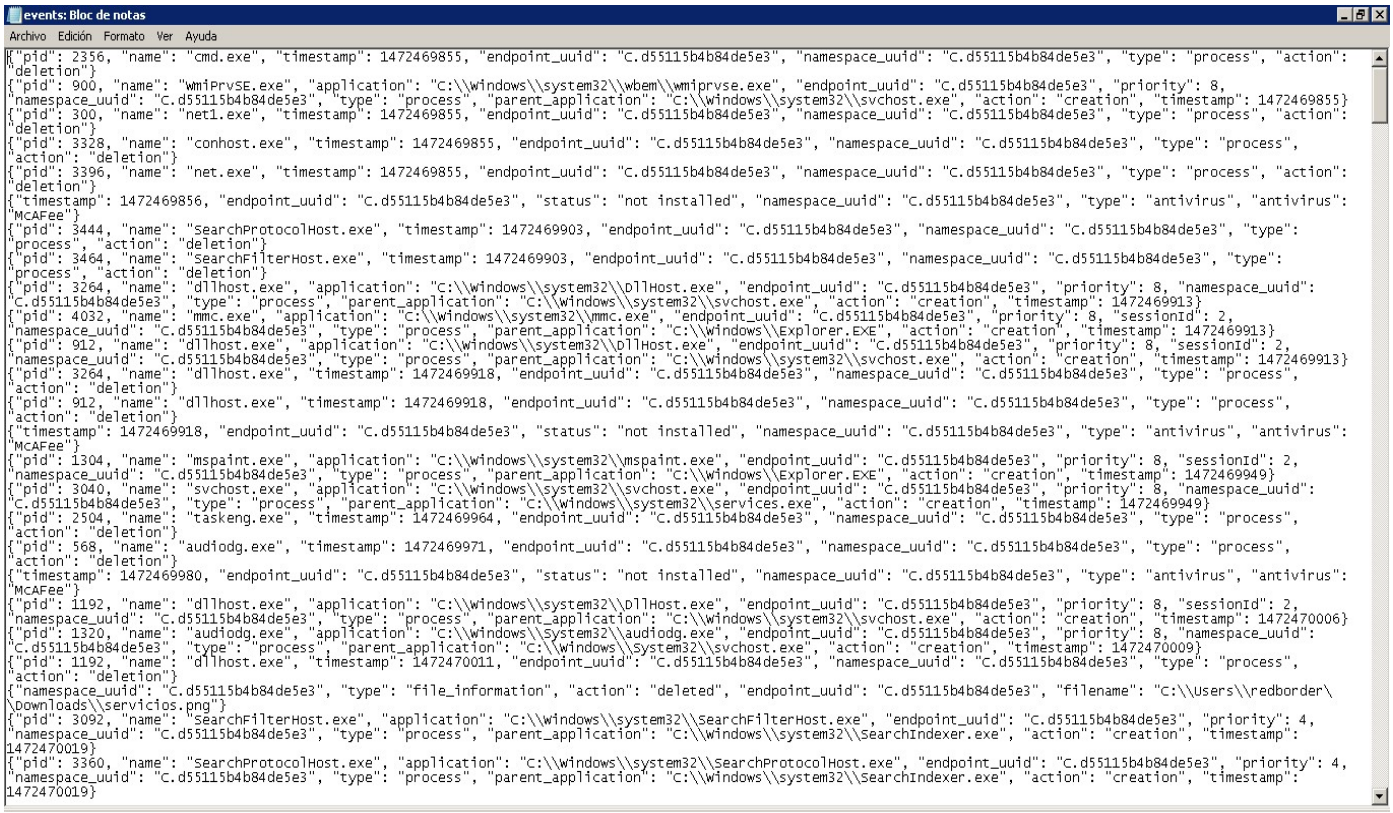


Figura 65 Contenido del histórico de eventos

6.2 Pruebas de carga y rendimiento

6.2.1 Instalación y registro de redBorder Malware EndPoint en el sistema

El objetivo de esta prueba es medir el rendimiento de un EndPoint para comprobar la actividad del equipo durante la instalación y ejecución de los componentes de redBorder Malware EndPoint. Para realizar esta tarea, se va a usar el motor de rendimiento de Microsoft Windows, llamado Perfmon.

El desarrollo de esta prueba ha consistido en instalar el software en diferentes EndPoints de 32 bits. Como resultado se han obtenidos tiempos que van desde los 2 minutos hasta los 15 minutos, en función de la congestión que sufra el servidor de GRR. Destacar que el tiempo mencionado hace referencia a la unión del tiempo de instalación en el equipo de los servicios con tiempo de registro de un EndPoint en la web de redBorder Malware. Por último, mencionar que todas las pruebas han sido realizadas por el equipo de desarrollo, entre los que se encuentra el autor de este proyecto.

A continuación, se ofrecen gráficas de la monitorización de uno de los EndPoints usados durante la prueba:

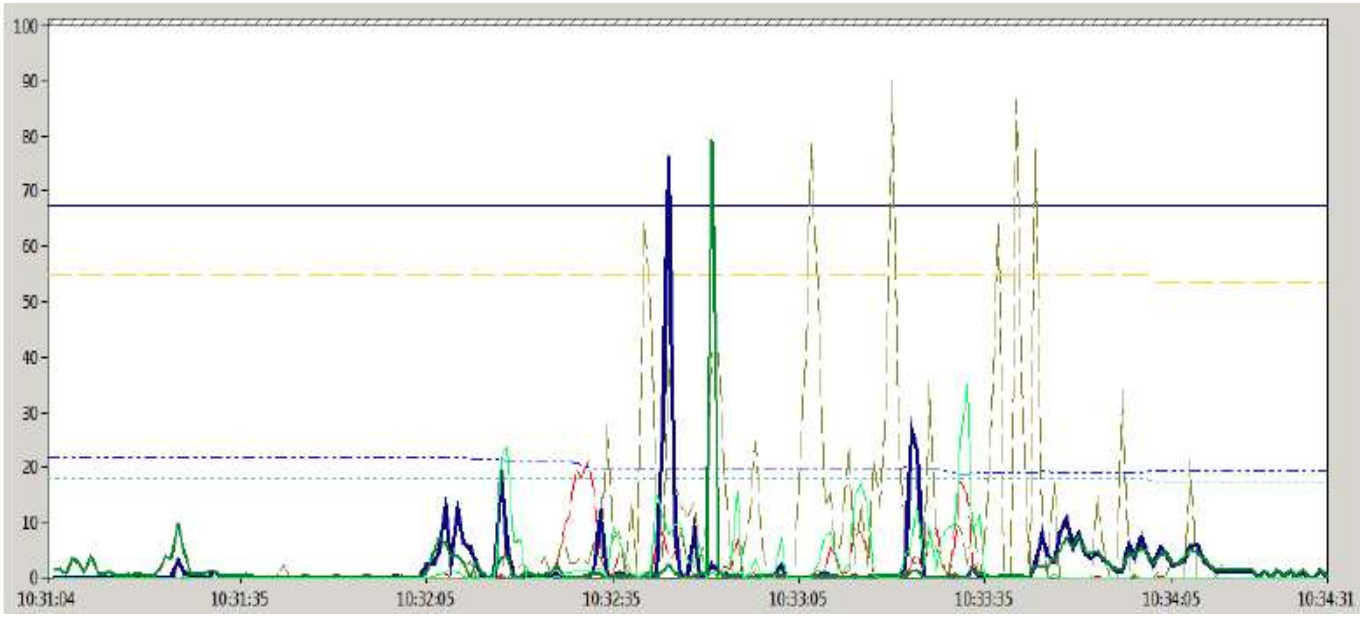


Figura 66 Actividad de un EndPoint durante la instalación del software

Como se observa, existen diferentes gráficos superpuesto en la imagen anterior, de los cuales, se explican los más relevantes a continuación:

- **Disco**: En la siguiente imagen, se puede apreciar el porcentaje de espacio en disco junto los MB ocupados por la instalación. La línea superior representa el porcentaje de espacio en disco disponible, disminuyendo en un 1,32% de un disco duro de 30 GB.

  El total de 432 MB son los usados en la instalación. Hay que tener en cuenta que durante el proceso de instalación se instala Python 3.4, dependencias de las librerías de bajo nivel para el agente EndPoint, el cliente GRR y el propio agente.

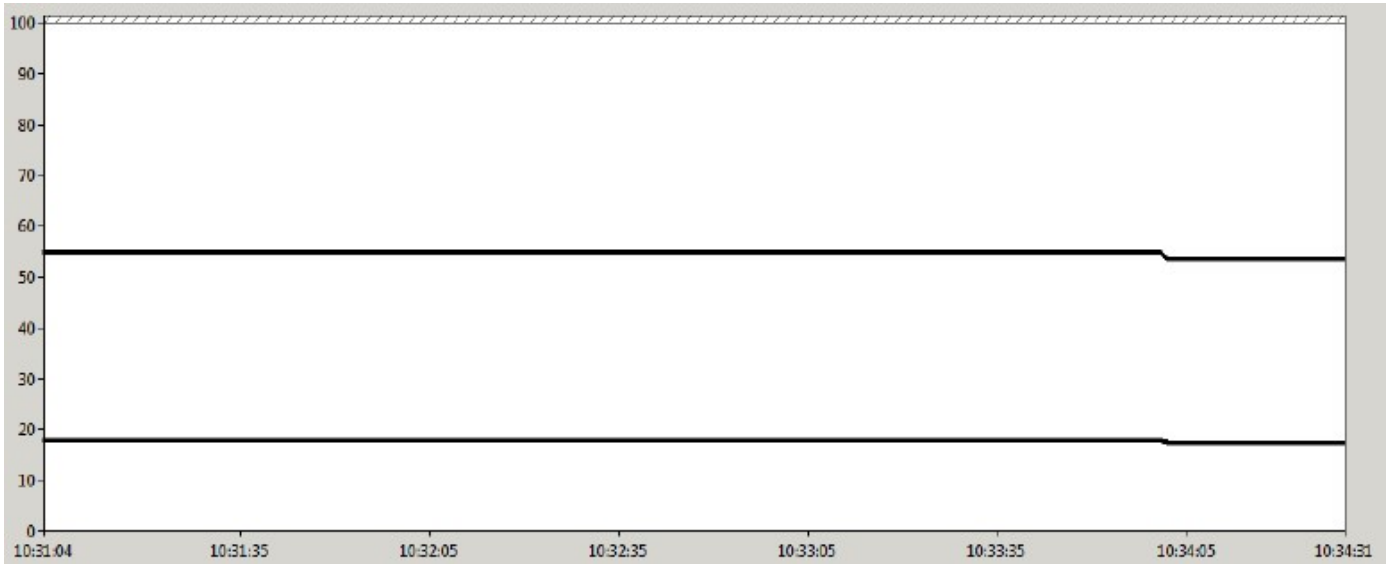


Figura 67 Actividad de Disco de un EndPoint durante la instalación del software

- **Memoria**: En esta gráfica se muestra la cantidad de memoria RAM consumida durante el proceso de instalación, mostrándose en la línea superior los Megabytes disponibles en memoria.

  Al iniciar la instalación, la memoria RAM inicial era de 2 175MB. Las variaciones se deben a los procesos levantados al instalar los componentes del endPoint. Finalmente, la memoria consumida es de 245MB. En la línea de la inferior de la gráfica, se representa la memoria en caché para el sistema, aumentando esta hasta los 171MB.

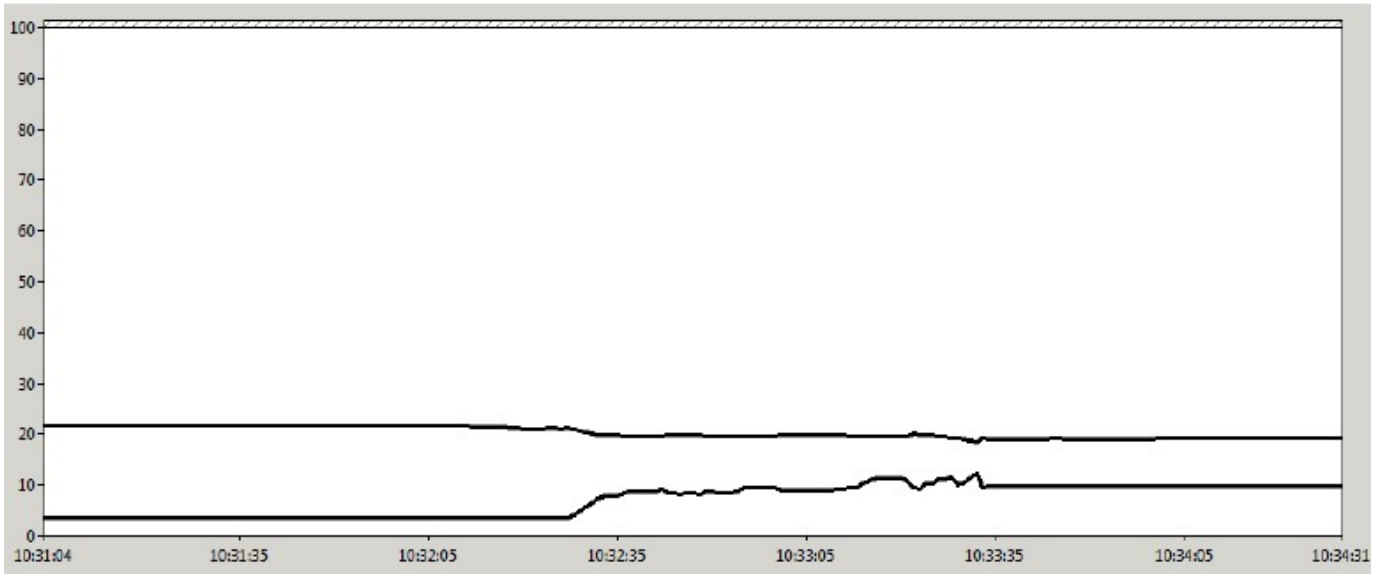


Figura 68 Actividad de la memoria RAM de un EndPoint durante la instalación del software

- **Tráfico de red**: La gráfica mostrada a continuación, muestra, por un lado, los bytes enviados por un EndPoint (azul), y por otro, los bytes recibidos (verde).

  Se pueden observar dos picos, ambos debidos al proceso GRR Monitor, provocado por el cliente GRR. El primero, se debe al registro del cliente en el servidor GRR, y el segundo, es provocado por el envío de información referente al EndPoint al mismo servidor.

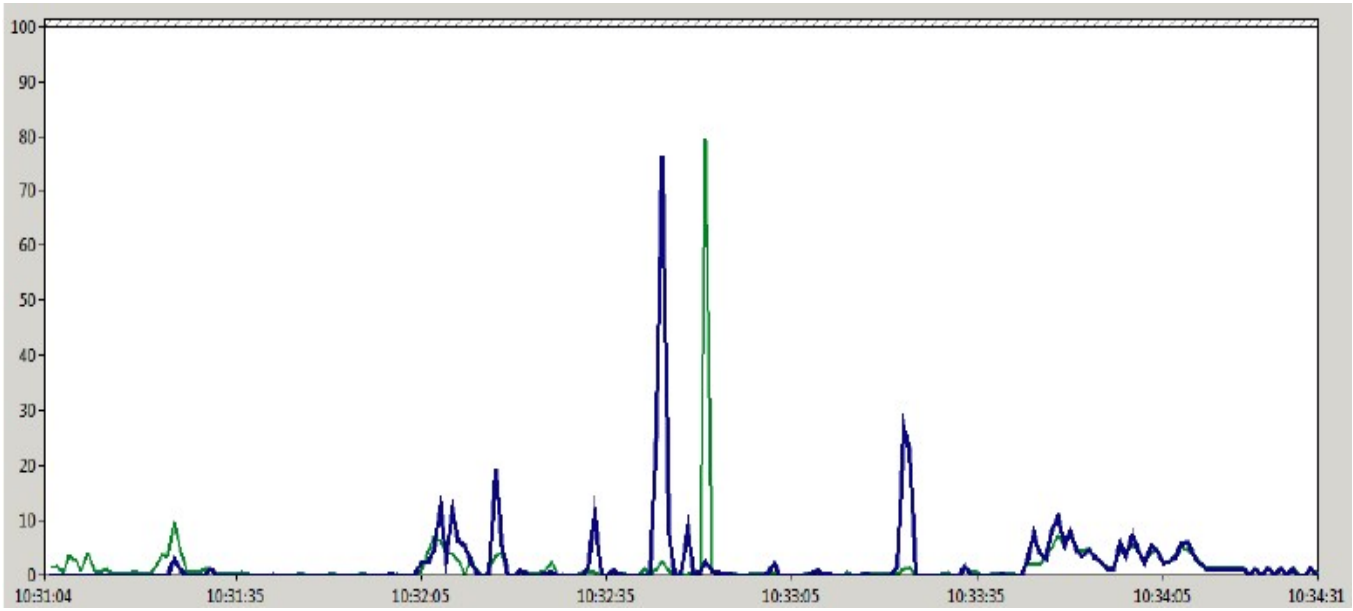


Figura 69 Actividad del tráfico de red de un EndPoint durante la instalación del software

- **Tiempo de actividad**: La siguiente gráfica, representa el tiempo de actividad del sistema (línea superior) y las operaciones de lectura y escritura (verde y roja respectivamente) del sistema.

  Se aprecia como la actividad es prácticamente constante, no llegando al 1% las variaciones. La operación de escritura (en rojo) tiene dos picos, uno al inciarse la instalación y otro al instalar los agentes de GRR y EndPoint. Las operaciones de lectura (en verde**)** son múltiples, pero entre todas ellas se puede observar un pico notable a la derecha de la gráfica que corresponde con la recolección de información del sistema para el registro en el servidor GRR. Entre esta información se encuentran: datos de registros, información de usuarios, conexiones de red, directorios de ficheros, etc.

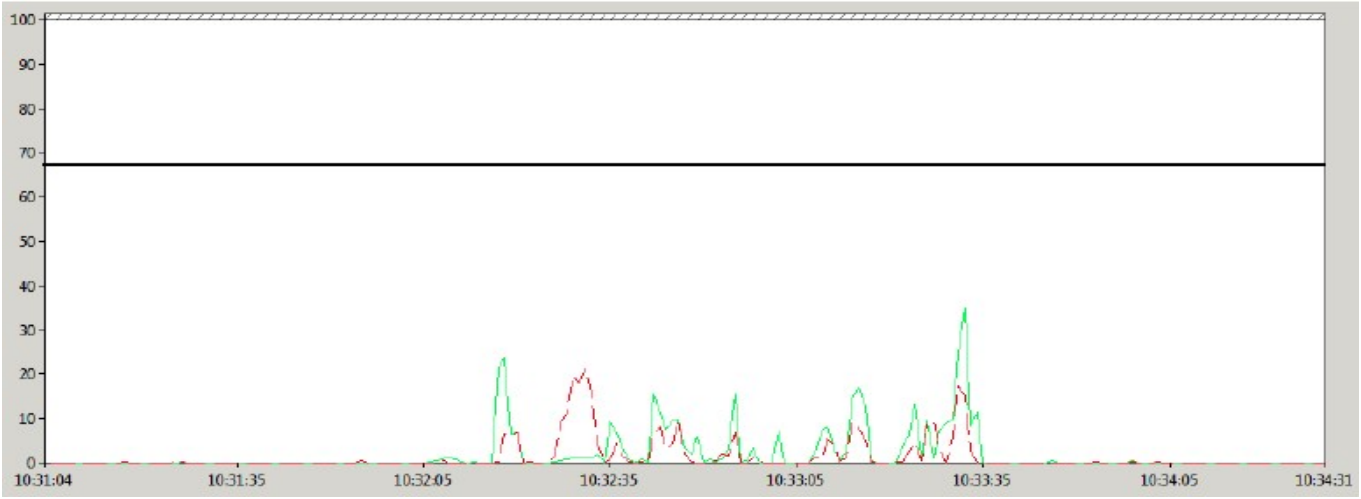


Figura 70 Actividad del sistema durante la instalación del software

6.2.2 Procesamiento de lotes de ficheros

La siguiente prueba pretende probar la capacidad de procesar lotes de ficheros en un EndPoint**.** Para ello se va a descargar y descomprimir un conjunto de ficheros que serán procesados por redBorder Malware EndPoint.

Se describen a continuación las características del EndPoint de prueba:

- Máquina virtual con Windows 7 de 64 bits de arquitectura.
- Gb de RAM.
- CPU con 2 cores.

El procedimiento a seguir es el siguiente:

- Se dispone de varios ficheros comprimidos con lotes de 5, 50, 100, 500 y 10000 archivos de diversos tipos y tamaños.
- El servicio de endPoint estará apagado al inicio de cada prueba para limpiar la caché y las colas de proceso, de esa forma priorizamos los archivos descomprimidos.

En la siguiente tabla, se da información referente a los ficheros, sus tipos y tamaños.

LOTE	TAMAÑO	TIPOS
5	992 KB	JPEG, MP3, TXT, HTML
50	20,4 MB	JPEG, MP3, TXT, HTML, AVI, DOCX
100	145 MB	JPEG, MP3, TXT, HTML, AVI, DOCX, GIF
500	438 MB	JPEG, MP3, TXT, HTML, AVI, DOCX, GIF, ZIP, XLS, BIN, PNG, MOV, MKV, PDF, PPT
1000	637 MB	JPEG, MP3, TXT, HTML, AVI, DOCX, GIF, ZIP, XLS, BIN, PNG, MOV, MKV, PDF, PPT

Tabla 3 Información de los ficheros de pruebas

Una vez se han procesado todos los lotes de ficheros por parte de redBorder Malware EndPoint, se obtienen los siguientes resultados:

LOTES	5	50	100	500	1000	UNIDADES
CPU	8	7,67	7,6	23,6	6,342	%
RAM	0,82	0,81	0,9	1,5	0,4	%
HDD	0,5	2,14	2,33	55,36	2,240	%
RED	8996,61	35766,5	15547,487	9789,543	15154,19	Bytes/seg
TIEMPO	138	1086	2371	19010	35665	Seg

Tabla 4 Resultados del análisis de los lotes de ficheros

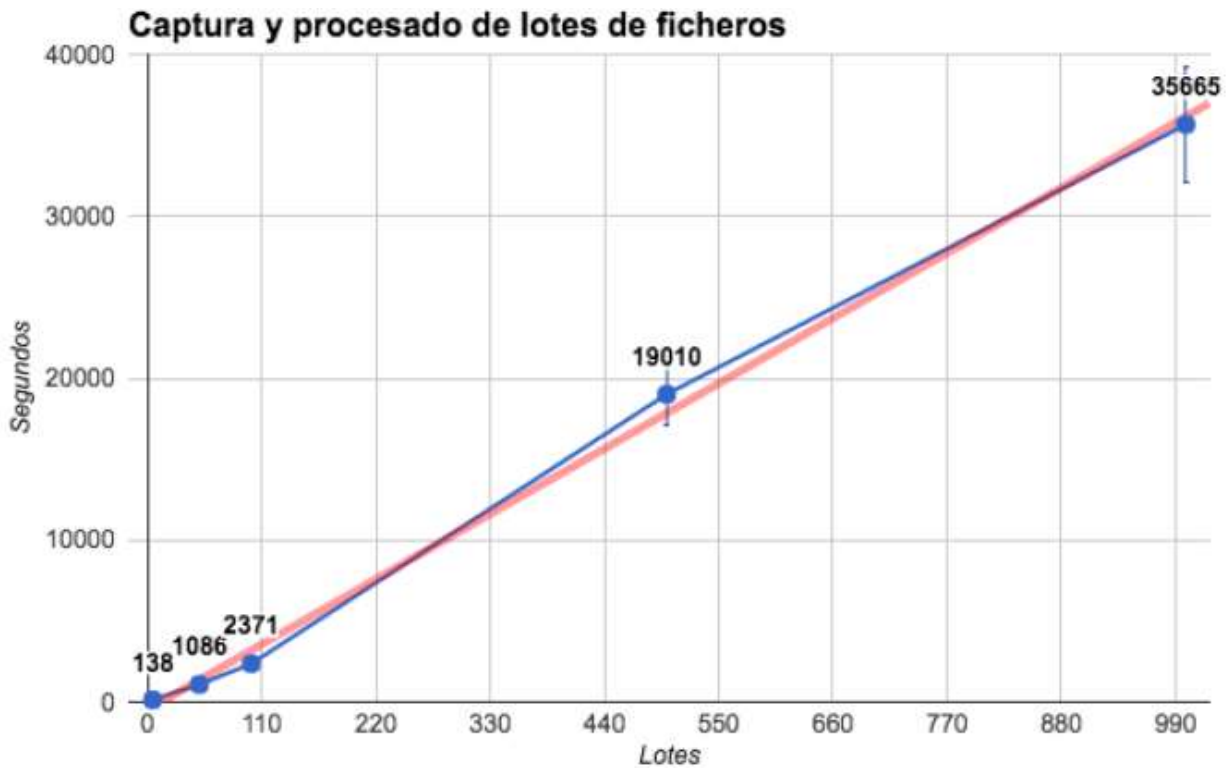


Figura 71 Gráfica de procesado de lotes de ficheros

La gráfica muestra el tiempo requerido para el procesado de cada uno de los lotes. Destacar cierta tendencia lineal en el comportamiento del servicio de EndPoint. Observando los recursos utilizados en los distintos procesamientos de lotes, se puede comprobar que el consumo de RAM se mantiene en unos valores en torno al 0,5% del incremento sobre el utilizado.

Destaca de forma notable el lote de 500 archivos, donde se aprecia un consumo de CPU y disco duro mayores; estos valores son penalizaciones debido a que el equipo está virtualizado y comparte los recursos del anfitrión con otras máquinas virtuales.

También puede impactar el uso de red para los lotes más grandes, dado que su uso es menor que por ejemplo el lote de 50 archivos. Estos valores más "relajados" están calculados como media en el espacio de tiempo que dura el procesamiento de los lotes de archivos.

6.2.3 Uso de un EndPoint

Esta prueba consiste en simular el uso por parte de un usuario de un EndPoint. Para automatizar el comportamiento se ha usado un bot, combinando acciones de edición y renombrado de de ficheros, descarga de malware y acceso a enlaces multimedia.

El objetivo de la prueba es tener el conocimiento necesario acerca del uso de CPU y memoria RAM:

- **CPU**: La actividad ha sido constante, como el patrón de funcionamiento del bot durante más de 12 horas. Se aprecian picos de procesamiento debido a la acumulación de procesos en los accesos a contenidos multimedia.

  Durante la prueba, el promedio de uso de CPU ha sido del 34,51%.

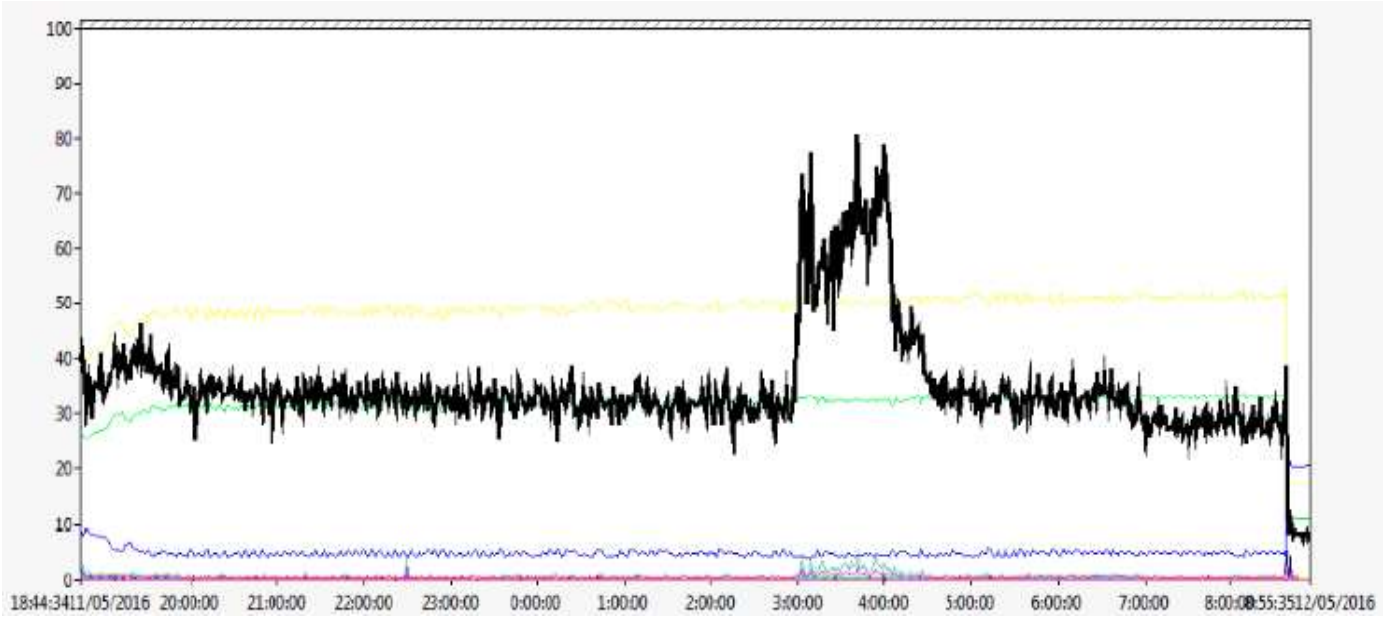


Figura 72 Gráfica de uso de CPU

- **Memoria RAM**: La actividad durante periodos de actividad intensa ha sido del 48,8%, pero una vez finalizada la actividad del bot, se observa un descenso hasta llegar al promedio del 30%.

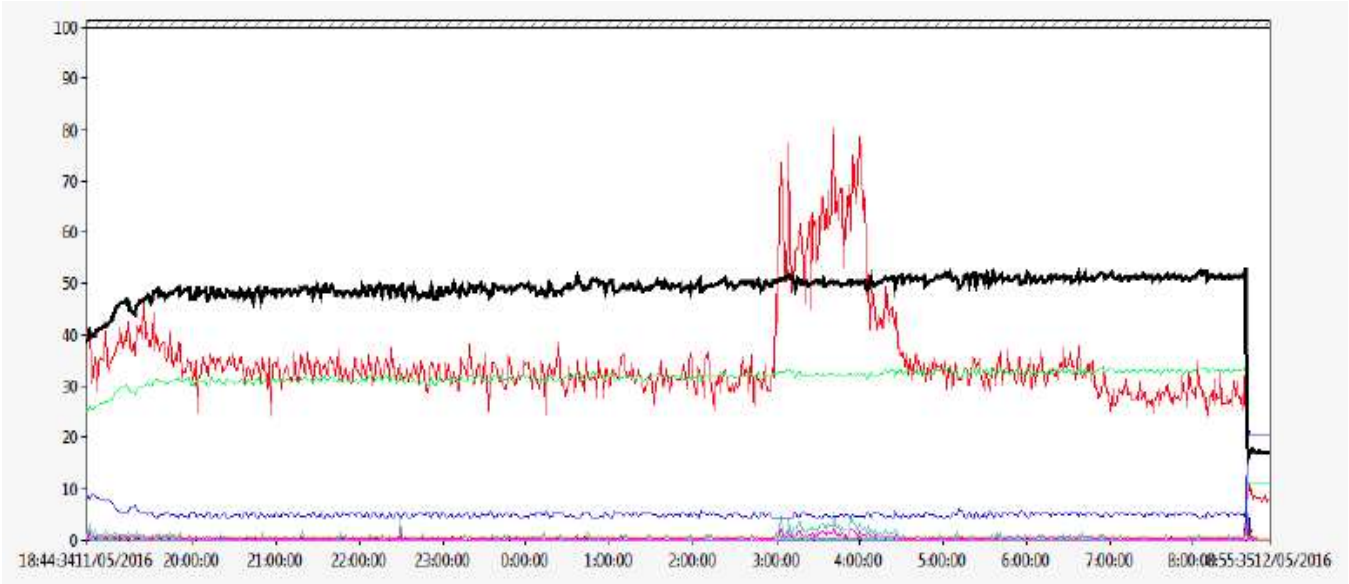


Figura 73 Gráfica de uso de memoria RAM

7 PRESUPUESTO.

En esta sección estiramos el coste del proyecto, basándonos en las horas dedicadas y el equipamento necesario para instalar todo el sistema:

Periodo de Trabajo	Número de Horas	Precio/Hora	Total
Agosto 2015 – Mayo 2016	**1035**	**30€/hora**	**31.020€**
Equipamiento	**Cantidad**	**Precio/Hora**	**Total**
Anfitrión para VMware ESXi (Entorno de virtualización de servidores).	**1**	**5.000€**	**5.000€**

Tabla 5 Presupuestos

En el anfitrión se virtualizan servidores que soportan los distintos servicios involucrados en el proyecto, estos servidores tienen las siguientes características:

- Memoria: 15 Gigas de RAM.
- CPUs: modelo “Intel(R) Xeon(R) CPU E5-1650 0 @ 3.20GHz”.
- Sistema redBorder, basado en CentOs.

Para llevar acabo este proyecto se ha calculado un presupuesto total de **36.020 €.**

8 CONCLUSIONES.

8.1 Conclusiones

La herramienta desarrollada en este proyecto, es estable, funcional y está incluida en la aplicación de malware de redBorder dentro del portfolio de productos.

Este presente documento, recoge el resultado de un largo y complejo proceso de investigación, aprendizaje, superación personal y principalmente trabajo. El trabajado se ha desarrollado con tecnologías código abierto para el desarrollo de este proyecto como son Python o GRR.

De forma adicional, al haber trabajo con el equipo de Eneo Tecnología SL, he adquirido enormes capacidades de trabajo en equipo y de comunicación.

Por último, atendiendo al resultado del proyecto y de sus pertinentes pruebas, se han conseguido cubrir todos los hitos propuestos. Se ha conseguido un sistema de seguridad para EndPoints que detecte en tiempo real eventos del sistema y los notifique a la plataforma de análisis de redBorder Malware, detectando así ataques, malware o amenazas avanzadas.

8.2 Futuras mejoras

El siguiente paso de este proyecto tratará sobre la adición al agente EndPoint un nuevo módulo de IOC, de modo que, cualquier evento del sistema sea analizado en el propio EndPoint para dilucidar si existe algún riesgo para el equipo.

De esta manera, el agente EndPoint quedaría exento de la necesidad de un análisis remoto contra IOC y además podría pasar a actuar de manera inmediate ante cualquier riesgo sin necesidad de esperar la respuesta del servidor GRR.

9 REFERENCIAS.

[1] Mike Halsey, Mayo 2011. Windows 7 Power User Guide [En línea] [Documento PDF]

Disponible: https://blogs.technet.microsoft.com/uktechnet/2011/05/10/download-the-windows-7-power-users-guide-ebook-free/

[2] Python.org. Documentación [En línea] [Web]

Disponible: https://docs.python.org/3/

[3] Apache Software Foundation – Apache Kafka [En línea] [Web]

Disponible: http://kafka.apache.org

[4] ENEO Tecnología – redBorder Malware [En línea] [Web]

Disponible: https://redborder.com/

[5] Basho – Riak CS Documentation [En línea] [Web]

Disponible: http://docs.basho.com/riakcs/latest/

[6] Basho – Riak Documentation [En línea] [Web]

Disponible: http://docs.basho.com/riak/latest/

[7] Microsoft – WMI API Documentation [En línea] [Web]

Disponible: https://msdn.microsoft.com/en-us/library/windows/desktop/ff818516(v=vs.85).aspx

[8] Malware Analyst's Cookbook. Autores: Michael Hale Ligh, Steven Adair, Blake Hartstein and Matthew Richard [En línea] [Documento PDF]

Disponible: https://docs.google.com/file/d/0B7d_gqEI7itKTm44RC1Hczh3LUU/edit?pli=1

[9] Github – GRR Documentation [En línea] [Web]

Disponible: https://github.com/google/grr-doc

[10] Raul Gonzalez Duque, Octubre 2014. Python para todos [En línea] [Documento PDF]

Disponible: https://launchpadlibrarian.net/18980633/Python%20para%20todos.pdf

[11] Mark Kedrich, Marzo 2007. EndPoint Security

Anexo A: Código del modulo principal

endpoint_agent.py

```
1.  import os
2.  import sys
3.
4.  # Config lib path for modules
5.  sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), 'lib')))
6.
7.  import platform
8.  import logging.config
9.  import pythoncom
10. import re
11. import win32serviceutil
12. import win32service
13. import win32event
14. import servicemanager
15. import socket
16. from config.Config import Config
17. from controllers.MonitorController import MonitorController
18. from managers.CacheManager import CacheManager
19. from managers.WindowsValueManager import WindowsValueManager
20. from managers.FileManager import FileManager
21. from controllers.tasks.general.KafkaSenderTask import KafkaSenderTask
22. from controllers.tasks.general.FileCaptureEventTask import FileCaptureEventTask
23. from controllers.tasks.general.FileCaptureSlowTask import FileCaptureSlowTask
24. from controllers.tasks.general.FileCaptureTask import FileCaptureTask
25. from controllers.tasks.windows.CreationProcessTask import CreationProcessTask
26. from controllers.tasks.windows.WebProcessTask import WebProcessTask
27. from controllers.tasks.windows.DeletionProcessTask import DeletionProcessTask
28. from controllers.tasks.windows.AntivirusAnalysisTask import AntivirusAnalysisTask
29. from controllers.tasks.windows.MonitorDiskPartition import MonitorDiskPartition
30. from managers.QueueManager import QueueManager
31. from managers.FilterManager import FilterManager
32. from managers.FileParameters import FileParameters
33. from managers.S3Manager import S3Manager
34.
35. S3_FOLDER_NAME = "s3"
36. LOGS_FOLDER_NAME = "logs"
37. BINNACLE_FOLDER_NAME = "events"
38. CONFIG_FOLDER_NAME = "config"
39. DATABASES_FOLDER_NAME = "db"
40.
41. class EndpointService(win32serviceutil.ServiceFramework):
42.
43.     _svc_name_ = "EndPoint"
44.     _svc_display_name_ = "EndPoint Service"
45.     configuration = None
46.     rootPath = None
47.
48.     def __init__(self, args):
49.         win32serviceutil.ServiceFramework.__init__(self, args)
50.         self.hWaitStop = win32event.CreateEvent(None, 0, 0, None)
51.         socket.setdefaulttimeout(60)
52.
```

```
53.         self.rootPath = os.path.abspath(os.path.dirname(__file__)).strip('endpoint_agen
    t')
54.
55.         if len(args) == 2:
56.             client_id = args[1][1:]
57.             with open(os.path.join(self.rootPath, 'config', 'parameters.yml'), 'a') as
    configFile:
58.                 configFile.write('\nclient_id: {0}'.format(client_id))
59.                 configFile.close()
60.
61.         # init logging system
62.         self.initLog()
63.
64.         # check if environment is completed
65.         self.initEnvironment()
66.
67.         # main thread configuration
68.         self.configuration = Config(path=os.path.join(self.rootPath, 'config', 'paramet
    ers.yml'))
69.
70.         # put in configuration a root path
71.         self.configuration.getConfigProperties()['rootpath'] = self.rootPath
72.
73.         # init file parameters
74.         self.initFileParametersConfig()
75.
76.         # init s3 configuration
77.         self.S3Config(self.configuration)
78.
79.         # init filter list
80.         self.initFilterLists(self.configuration)
81.
82.         # init cache manager
83.         CacheManager.initCache(self.configuration.getConfigProperties())
84.
85.
86.
87.     def SvcStop(self):
88.         self.ReportServiceStatus(win32service.SERVICE_STOP_PENDING)
89.         win32event.SetEvent(self.hWaitStop)
90.
91.         self.shutdown()
92.
93.
94.     def SvcDoRun(self):
95.         servicemanager.LogMsg(servicemanager.EVENTLOG_INFORMATION_TYPE,
96.                               servicemanager.PYS_SERVICE_STARTED,
97.                               (self._svc_name_,''))
98.
99.         self.run()
100.
101.
102.        def run(self):
103.
104.             # create queues
105.             QueueManager.createQueue('slow')
106.             QueueManager.createQueue('file_event')
107.             QueueManager.createQueue('kafka')
108.             QueueManager.createQueue('file_capture')
109.
110.             CacheManager.start()
111.
112.             # create monitor controller
113.             self.controller = MonitorController()
114.
115.             # running task kafka sender
```

```
116.                kafkaTask = KafkaSenderTask(taskName='KafkaSender', config=self.configur
    ation.getConfigProperties())
117.                self.controller.runTask(kafkaTask)
118.
119.                # running task antivirus analysis
120.                antivirusAnalysis = AntivirusAnalysisTask(taskName='AntivirusAnalysis',
    config=self.configuration.getConfigProperties())
121.                self.controller.runTask(antivirusAnalysis)
122.
123.                # running task creation process
124.                creationProcess = CreationProcessTask(taskName='creationProcess')
125.                self.controller.runTask(creationProcess)
126.
127.                # running task deletion process
128.                deletionProcess = DeletionProcessTask(taskName='deletionProcess')
129.                self.controller.runTask(deletionProcess)
130.
131.                # running task web process
132.                webProcess = WebProcessTask(taskName='webProcess')
133.                self.controller.runTask(webProcess)
134.
135.                # init windows variables
136.                self.initWindowsValues()
137.
138.                # runnning task file capture slow
139.                slowTask = FileCaptureSlowTask(taskName='slowFileCapture')
140.                self.controller.runTask(slowTask)
141.
142.                # running task file capture
143.                normalTask = FileCaptureTask(taskName='normalFileCapture')
144.                self.controller.runTask(normalTask)
145.
146.                # running task file capture event
147.                eventTask = FileCaptureEventTask(taskName='fileEventCapture')
148.                self.controller.runTask(eventTask)
149.
150.                # runnning task monitor process
151.                monitorProcess = MonitorDiskPartition(taskName='MonitorDiskProcess')
152.                self.controller.runTask(monitorProcess)
153.
154.                self.controller.getWorkersResults()
155.
156.            def initEnvironment(self):
157.
158.                if not os.path.exists(os.path.join(self.rootPath, 'endpoint_agent')):
159.                    os.mkdir(os.path.join(self.rootPath, 'endpoint_agent'))
160.
161.                # create path for s3
162.                path_s3 = os.path.join(self.rootPath, 'endpoint_agent', S3_FOLDER_NAME)
163.
164.                # check if exist path for s3
165.                if not os.path.exists(path_s3):
166.                    os.mkdir(path_s3)
167.                    logging.getLogger('application').info("Created new folder : {0}".for
    mat(path_s3))
168.
169.                # create path for logs
170.                path_logs = os.path.join(self.rootPath, LOGS_FOLDER_NAME)
171.
172.                # check if exist path for logs
173.                if not os.path.exists(path_logs):
174.                    os.mkdir(path_logs)
175.                    logging.getLogger('application').info("Created new folder : {0}".for
    mat(path_logs))
176.
177.                # create path for binnacle
```

```
178.                 path_binnacle = os.path.join(self.rootPath, BINNACLE_FOLDER_NAME)
179.
180.                 # check if exist path for binnacle
181.                 if not os.path.exists(path_binnacle):
182.                     os.mkdir(path_binnacle)
183.                     logging.getLogger('application').info("Created new folder : {0}".for
    mat(path_binnacle))
184.
185.                 # create path for databases
186.                 path_cache = os.path.join(self.rootPath, 'endpoint_agent', DATABASES_FOL
    DER_NAME)
187.
188.                 # check if exist path for databases
189.                 if not os.path.exists(path_cache):
190.                     os.mkdir(path_cache)
191.                     logging.getLogger('application').info("Created new folder : {0}".for
    mat(path_cache))
192.
193.             # Method to init a logs and binnacle
194.             def initLog(self):
195.                 application = os.path.join(self.rootPath, LOGS_FOLDER_NAME, 'logs.log')

196.                 binnacle = os.path.join(self.rootPath, BINNACLE_FOLDER_NAME, 'events.jso
    n')
197.                 dictionary = dict()
198.                 dictionary['application'] = repr(application)
199.                 dictionary['binnacle'] = repr(binnacle)
200.                 logging.config.fileConfig(os.path.join(self.rootPath, CONFIG_FOLDER_NAME
    , 'logconf.conf'), defaults=dictionary)
201.
202.             # Method to init a windows values
203.             def initWindowsValues(self):
204.                 WindowsValueManager.set_values()
205.
206.             # Method to init a filters lists
207.             def initFilterLists(self, config):
208.                 path = os.path.join(self.rootPath, 'endpoint_agent')
209.                 FilterManager.load_config(config, path)
210.
211.             # Method to configuration a S3.
212.             def S3Config(self, config):
213.                 path = os.path.join(self.rootPath, 'endpoint_agent')
214.                 S3Manager.load_config(config, path)
215.
216.             # Method to init a File Parameters configuration
217.             def initFileParametersConfig(self):
218.                 path = os.path.join(self.rootPath, 'endpoint_agent')
219.                 FileParameters.config(path)
220.
221.             # Method to finish all workers, close all files and a cache
222.             def shutdown(self):
223.                 self.controller.stopWorkers()
224.                 CacheManager.shutdown()
225.                 FileManager.closeAllFiles()
226.
227.
228.
229.         if __name__ == '__main__':
230.             win32serviceutil.HandleCommandLine(EndpointService)
```

Anexo B: Código del módulo de gestores

managers/CacheManager.py

```
1.  from enum import Enum
2.  import requests
3.  import logging
4.  from concurrent.futures import ThreadPoolExecutor
5.  import time
6.  import sqlite3
7.  import os
8.  import threading
9.
10. # Status for cache manager
11. class Status(Enum):
12.
13.     # Cache manager is stopped
14.     STOPPED = 1
15.
16.     # Cache manager is preparing
17.     PREPARING = 2
18.
19.     # Cache manager is ready
20.     READY = 3
21.
22.     # Cache manager is running
23.     RUNNING = 4
24.
25.     # Cache manager is stopping
26.     STOPPING = 5
27.
28.
29. class CacheManager():
30.
31.     URL_REPUTATION_TOTAL = "https://{0}/reputation/{1}/malware/total{2}"
32.     URL_REPUTATION_INCREMENTAL = "https://{0}/reputation/{1}/malware/incremental"
33.     URL_REPUTATION_INCREMENTAL_LIST = "https://{0}/reputation/{1}/malware/incremental/{
    2}{3}"
34.
35.     semaphore = None
36.
37.     DATABASE_NAME = 'cache.db'
38.
39.     # path where cache manager saves information
40.     path_cache = None
41.
42.     # server address
43.     url_server = None
44.
45.     # cache version
46.     version = None
47.
48.     # Cache manager status (STOPPED by default)
49.     status = Status.STOPPED
50.
```

```
51.     # connection to sqlite
52.     dbConnection = None
53.
54.     # lastest incremental list query
55.     last_incremental_query = 'r.i'
56.
57.     # init local cache with config file
58.     @staticmethod
59.     def initCache(config):
60.         if CacheManager.status is Status.STOPPED:
61.
62.             CacheManager.status = Status.PREPARING
63.
64.             logging.getLogger('application').info("Init cache client")
65.             # load configuration
66.             CacheManager.url_server = config['reputation_server']
67.             CacheManager.version = config['version_cache']
68.             databasePath = os.path.join(config['rootpath'], 'endpoint_agent', 'db', Cac
    heManager.DATABASE_NAME)
69.             CacheManager.dbConnection = sqlite3.connect(databasePath, check_same_thread
    =False)
70.             CacheManager.semaphore = threading.Semaphore()
71.
72.             # define table in BBDD
73.             if CacheManager.dbConnection:
74.                 logging.getLogger('application').debug("Connected to local cache databa
    se: {0}".format(CacheManager.DATABASE_NAME))
75.                 CacheManager.dbConnection.cursor().execute('CREATE TABLE IF NOT EXISTS
    cache (hash text PRIMARY KEY, score integer)')
76.                 CacheManager.dbConnection.commit()
77.
78.             CacheManager.status = Status.READY
79.
80.
81.     @staticmethod
82.     def start():
83.
84.         if CacheManager.status is Status.READY:
85.
86.             CacheManager.status = Status.RUNNING
87.
88.             # Init cache task
89.             executor = ThreadPoolExecutor(max_workers=1)
90.             executor.submit(CacheManager.doPetitions)
91.
92.             logging.getLogger('application').info("Started cache client!")
93.
94.
95.     @staticmethod
96.     def doPetitions():
97.         if (CacheManager.status is Status.RUNNING):
98.
99.             logging.getLogger('application').debug("Doing total petition to remote cach
    e")
100.                 # first time get total with filter hash
101.                 CacheManager.getTotal("hash")
102.
103.                 logging.getLogger('application').debug("Doing incremental petition t
    o remote cache")
104.                 # first time get incremental
105.                 CacheManager.getIncremental("hash")
106.
107.                 # every X seconds get incremental with filter hash
108.                 while CacheManager.status is Status.RUNNING and not time.sleep(900):

109.                     CacheManager.getIncremental("hash")
```

```
110.                         logging.getLogger('application').debug("Do incremental petition
    to remote cache")
111.
112.
113.
114.             @staticmethod
115.             def getTotal(filter):
116.                 # build url
117.                 url = CacheManager.URL_REPUTATION_TOTAL.format(CacheManager.url_server,
    CacheManager.version, "/%s" % filter if filter else "")
118.
119.                 try:
120.
121.                     CacheManager.semaphore.acquire()
122.
123.                     if (CacheManager.status is Status.RUNNING):
124.
125.                         # get remote response
126.                         response = requests.get(url, verify=False)
127.
128.                         logging.getLogger('application').debug("Response in total petiti
    on : {0}".format(response.status_code))
129.
130.                         if response.status_code is 200:
131.
132.                             # if status reponse is correct put in BBDD all data
133.                             data = response.json()['data']
134.
135.                             for entry in data:
136.                                 if 'hash' in entry and 'score' in entry:
137.                                     hashValue = entry['hash']
138.                                     scoreValue = entry['score']
139.                                     CacheManager.dbConnection.cursor().execute('INSERT O
    R REPLACE INTO cache VALUES (\'{0}\',{1})'.format(hashValue, scoreValue))
140.                                     logging.getLogger('application').debug("Updated data
    base : hash {0} with score {1}.".format(repr(hashValue), scoreValue))
141.
142.                             CacheManager.dbConnection.commit()
143.                             logging.getLogger('application').info("Updated local cache d
    atabase with total list")
144.                         else:
145.                             logging.getLogger('application').warning("Unable to get tota
    l list from : {1}".format(CacheManager.url_server))
146.
147.                 except Exception as e:
148.                     logging.getLogger('application').error('Error to get total list from
     remote cache')
149.                     logging.getLogger('application').error(e)
150.                 finally:
151.                     CacheManager.semaphore.release()
152.
153.
154.
155.             @staticmethod
156.             def getIncremental(filter):
157.
158.                 try:
159.                     CacheManager.semaphore.acquire()
160.
161.                     if (CacheManager.status is Status.RUNNING):
162.
163.                         # get incremental cache
164.                         url = CacheManager.URL_REPUTATION_INCREMENTAL.format(CacheManage
    r.url_server, CacheManager.version)
165.
166.                         # get remote response
167.                         response = requests.get(url, verify=False)
```

```
168.
169.                         logging.getLogger('application').debug("Response in last increme
    ntal petition : {0}".format(response.status_code))
170.
171.                         # if status reponse is correct checked latest updates
172.                         if response.status_code is 200:
173.
174.                             lastRelease = response.json()['last_release']
175.
176.                             if not lastRelease == CacheManager.last_incremental_query an
    d not CacheManager.last_incremental_query == '0.0':
177.
178.                                 CacheManager.last_incremental_query = lastRelease
179.
180.                                 url = CacheManager.URL_REPUTATION_INCREMENTAL_LIST.forma
    t(CacheManager.url_server, CacheManager.version, lastRelease, "/%s" % filter if filter
    else "")
181.
182.                                 response = requests.get(url, verify=False)
183.
184.                                 logging.getLogger('application').debug("Response in incr
    emental list petition : {0}".format(response.status_code))
185.
186.                                 if response.status_code is 200:
187.
188.                                     #If remote cache changed, updated endpoint local cac
    he
189.                                     data = response.json()['data']
190.
191.                                     for entry in data:
192.                                         if 'hash' in entry and 'score' in entry:
193.                                             hashValue = entry['hash']
194.                                             scoreValue = entry['score']
195.
196.                                             CacheManager.dbConnection.cursor().execute('
    INSERT OR REPLACE INTO cache VALUES (\'{0}\',{1})'.format(hashValue, scoreValue))
197.                                             logging.getLogger('application').debug("Upda
    ted database incremental list {0} : hash {1} with score {2}".format(repr(lastRelease),
    repr(hashValue), str(scoreValue)))
198.
199.                                     CacheManager.dbConnection.commit()
200.
201.                                     logging.getLogger('application').info("Updated local
     cache database with incremental list : {0}".format(repr(lastRelease)))
202.                             else:
203.                                 logging.getLogger('application').info("Not incremental l
    ist available!")
204.
205.                 except Exception as e:
206.                     logging.getLogger('application').error('Error to get incremental lis
    t from remote cache')
207.                     logging.getLogger('application').error(e)
208.                 finally:
209.                     CacheManager.semaphore.release()
210.
211.
212.             @staticmethod
213.             def get_score(hashvalue):
214.
215.                     # get score from hash value
216.                     try:
217.                         CacheManager.semaphore.acquire()
218.
219.                         if (CacheManager.status is Status.RUNNING):
220.
221.                             cursor = CacheManager.dbConnection.cursor()
```

```
222.
223.                             cursor.execute('SELECT score FROM cache WHERE hash=?', (hash
    value,))
224.
225.                             result = cursor.fetchone()
226.
227.                             if result:
228.                                 logging.getLogger('application').debug("Result for hash
    {0} : {1}".format(hashvalue, result[0]))
229.                                 return result[0]
230.                             else:
231.                                 logging.getLogger('application').debug("Results not foun
    d for hash %s" % hashvalue)
232.                         else:
233.                             return 1000
234.
235.                     except Exception as e:
236.                         logging.getLogger('application').error("Error to query hash {0}"
    .format(hashvalue))
237.                         logging.getLogger('application').error(e)
238.                         return 1000
239.                     finally:
240.                         CacheManager.semaphore.release()
241.
242.
243.             @staticmethod
244.             def checkIfExists(hashvalue):
245.
246.                 try:
247.                     CacheManager.semaphore.acquire()
248.
249.                     if(CacheManager.status is Status.RUNNING):
250.                         #checked if hash value exists in local cache
251.                         cursor = CacheManager.dbConnection.cursor()
252.
253.                         cursor.execute("SELECT EXISTS(SELECT 1 FROM cache WHERE hash=\'{
    0}\' LIMIT 1)".format(hashvalue))
254.
255.                         result = cursor.fetchone()[0]
256.
257.                         if bool(result):
258.                             logging.getLogger('application').debug("Value {0} exists in
    database".format(hashvalue))
259.                         else:
260.                             logging.getLogger('application').debug("Value {0} doesn't ex
    ists in database".format(hashvalue))
261.                             CacheManager.dbConnection.cursor().execute('INSERT OR REPLAC
    E INTO cache VALUES (\'{0}\',{1})'.format(hashvalue, -1))
262.                             CacheManager.dbConnection.commit()
263.                             logging.getLogger('application').info("Added new entry {0} i
    n local cache".format(hashvalue))
264.
265.                         return bool(result)
266.
267.                     return None
268.
269.                 except Exception as e:
270.                     logging.getLogger('application').error("Error to add new entry for h
    ash {0}".format(hashvalue))
271.                     logging.getLogger('application').error(e)
272.                 finally:
273.                     CacheManager.semaphore.release()
274.
275.
276.
277.             @staticmethod
278.             def shutdown():
```

```
279.                 if CacheManager.status is Status.RUNNING:
280.
281.                     CacheManager.status = Status.STOPPING
282.
283.                     # Wait a minute, if not close connection
284.                     try:
285.                         CacheManager.semaphore.acquire()
286.                         CacheManager.dbConnection.close()
287.                         CacheManager.status = Status.STOPPED
288.                         logging.getLogger('application').info('Local Cache Manager stopp
    ed')
289.                     except Exception as e:
290.                         logging.getLogger('application').error("Error to close database"
    )
291.                         logging.getLogger('application').error(e)
292.                     finally:
293.                         CacheManager.semaphore.release()
```

managers/KafkaManager.py

```
1.  from kafka import KafkaClient
2.  from kafka import KeyedProducer
3.  from kafka import HashedPartitioner
4.  from kafka.common import (RequestTimedOutError, MessageSizeTooLargeError)
5.
6.  import time
7.  import logging
8.
9.
10. class KafkaManager():
11.
12.
13.     # constructor
14.     def __init__(self, kafkaConfig):
15.         self.kafkaConfig = kafkaConfig
16.         self.init()
17.         self.running = True
18.         logging.getLogger('application').info("Init new kafka manager")
19.
20.     # init kafka producer
21.     def init(self):
22.         running = False
23.         try:
24.             self.kafkaClient = KafkaClient(self.kafkaConfig['kafka_server'])
25.             logging.getLogger('application').info("Connect with broker: {0}".format(sel
    f.kafkaConfig['kafka_server']))
26.
27.             logging.getLogger('application').info("Init new producer")
28.             self.kafkaProducer = KeyedProducer(self.kafkaClient, partitioner=HashedPart
    itioner, async=False)
29.
30.             running = True
31.
32.         except Exception as e:
33.             logging.getLogger('application').error("Kafka server connect failed.")
34.             logging.getLogger('application').error(e)
35.
36.
37.     def reconnect(self, topic, key, msg):
38.         trying = 1
39.         running = False
40.         while trying <= 5 and not running:
41.             try:
42.                 self.kafkaClient = KafkaClient(self.kafkaConfig['kafka_server'])
43.                 logging.getLogger('application').info("Reconnect with broker: {0}".form
    at(self.kafkaConfig['kafka_server']))
44.
45.                 logging.getLogger('application').info("Updated producer")
46.                 self.kafkaProducer = KeyedProducer(self.kafkaClient, partitioner=Hashed
    Partitioner, async=False)
47.
48.                 self.kafkaProducer.send_messages(topic, bytes(key, 'utf-
    8'), bytes(msg, 'utf-8'))
49.
50.                 running = True
51.
52.             except Exception as e:
53.                 logging.getLogger('application').error("Kafka server reconnect failed.
    {0} of 5 retries".format(trying))
54.                 logging.getLogger('application').error(e)
55.                 time.sleep(5)
56.                 trying += 1
57.                 continue
58.
```

```
59.     def kafkaRequestTimedOutError(self, topic, key, msg):
60.         trying = 1
61.         running = False
62.         while trying <= 5 and not running:
63.             try:
64.                 self.kafkaProducer.send_messages(topic, bytes(key, 'utf-
    8'), bytes(msg, 'utf-8'))
65.                 running = True
66.             except RequestTimedOutError:
67.                 logging.getLogger('application').error("Kafka request time out error. {
    0} of 5 retries".format(trying))
68.                 time.sleep(5)
69.                 trying += 1
70.
71.     # send message with or without key to topic
72.     def sendMessage(self, msg, topic, key=None):
73.         logging.getLogger('application').debug("Sending message to topic: %s", repr(top
    ic))
74.         try:
75.             self.kafkaProducer.send_messages(topic, bytes(key, 'utf-
    8'), bytes(msg, 'utf-8'))
76.         except RequestTimedOutError:
77.             self.kafkaRequestTimedOutError(topic, key, msg)
78.             return
79.         except MessageSizeTooLargeError:
80.             logging.getLogger('application').error("Message size is too large to send k
    afka server {0}".format(msg))
81.             return
82.         except Exception as e:
83.             logging.getLogger('application').error(e)
84.             self.reconnect(topic, key, msg)
```

managers/RiakManager.py

```
1.  import time
2.  import os
3.  import shutil
4.  import logging
5.  import boto3
6.  from botocore.utils import fix_s3_host
7.
8.  class S3Manager():
9.
10.     configuration = None
11.     path = None
12.
13.     # Method to load configuration for S3
14.     @staticmethod
15.     def load_config(config, path):
16.         S3Manager.configuration = config.getConfigProperties()
17.         S3Manager.path = path
18.
19.
20.     # Method to get file from S3
21.     @staticmethod
22.     def get_s3_file(sha256):
23.         s3_path = os.path.join(S3Manager.path, 's3', sha256)
24.         return s3_path
25.
26.
27.     # Method to create temporal file.
28.     @staticmethod
29.     def create_temp_file(old_file, new_file):
30.         flag = False
31.         count = 0
32.         while flag == False:
33.             try:
34.                 if count < 5:
35.                     shutil.copyfile(old_file, new_file)
36.                     flag = True
37.                     logging.getLogger('application').info('Created temporal file : {0}'
    .format(new_file))
38.                     return "ok"
39.                 else:
40.                     flag = True
41.                     logging.getLogger('application').error('Error to create temporal fi
    le : {0}'.format(new_file))
42.                     return "error"
43.             except:
44.                 time.sleep(15)
45.                 count += 1
46.
47.
48.     # Method to upload files to S3
49.     @staticmethod
50.     def upload_s3(filename, sha256):
51.         s3_file_upload = S3Manager.get_s3_file(sha256)
52.         aws_access = S3Manager.configuration['access_key']
53.         aws_secret = S3Manager.configuration['secret_key']
54.         bucket_name = S3Manager.configuration['s3_bucket']
55.         bucket_key = S3Manager.configuration['s3_key']
56.
57.         copy_status = S3Manager.create_temp_file(filename, s3_file_upload)
58.
59.         if copy_status == "ok":
60.             count = 0
61.             flag = True
62.             while flag == True:
```

```
                try:
                    # connect to hosts.
                    s3resource = boto3.resource(service_name="s3",
                                                verify=False,
                                                endpoint_url="https://s3.redborder.cluster",
                                                aws_access_key_id=aws_access,
                                                aws_secret_access_key=aws_secret)

                    # Avoid redirect
                    s3resource.meta.client.meta.events.unregister('before-sign.s3', fix_s3_host)

                    # Get bucket
                    bucket = s3resource.Bucket(bucket_name)
                    data = open(s3_file_upload, 'rb')

                    # upload file to s3
                    bucket.put_object(Key=bucket_key + sha256, Body=data)
                    # close file
                    data.close()
                    time.sleep(1)
                    flag = False
                    if os.path.exists(s3_file_upload):
                        os.remove(s3_file_upload)
                    logging.getLogger('application').info('Uploaded file {0}'.format(repr(filename)))
                    return "upload"
                except Exception as e:
                    logging.getLogger('application').error('Failed to upload file {0} to S3'.format(repr(filename)))
                    logging.getLogger('application').error(str(e))
                    #Max intents.
                    if count < 2:
                        count += 1
                        time.sleep(2)
                    else:
                        return "error_upload"
        else:
            return "error_copy"
```

managers/QueueManager.py

```
1.  from queue import Queue
2.  from queue import Full
3.
4.  import logging
5.
6.  # Static class for queue management
7.  class QueueManager():
8.
9.      # created queues
10.     queues = dict()
11.
12.
13.     # method to create a new queue with key
14.     @staticmethod
15.     def createQueue(key):
16.         if not key in QueueManager.queues:
17.             QueueManager.queues[key] = Queue()
18.             logging.getLogger('application').info("Created new queue with key: \'%s\'",
     key)
19.
20.
21.     # method to put data in queue with key
22.     @staticmethod
23.     def putOnQueue(key, data):
24.         if key in QueueManager.queues:
25.             QueueManager.queues[key].put(data)
26.             logging.getLogger('application').debug("Added new data to queue with key: \
    '%s\'", key)
27.             logging.getLogger('application').debug("Data value : %s", data)
28.         else:
29.             logging.error("key \'%s\' not found!", key)
30.             raise QueueManagerException(key)
31.
32.
33.     # method to know if a queue with key is empty
34.     @staticmethod
35.     def isEmpty(key):
36.         if key in QueueManager.queues:
37.             logging.getLogger('application').debug("Checking size of queue with key: \'
    %s\'", key)
38.             return QueueManager.queues[key].empty()
39.         else:
40.             logging.getLogger('application').error("key \'%s\' not found!", key)
41.             raise QueueManagerException(key)
42.
43.
44.     # method to get data from a queue with key
45.     @staticmethod
46.     def getItemFromQueue(key, timeout=None):
47.         try:
48.             if key in QueueManager.queues:
49.                 logging.getLogger('application').debug("Get item from queue with key: \
    '%s\'", key)
50.                 return QueueManager.queues[key].get(timeout=timeout)
51.             else:
52.                 logging.getLogger('application').error("key \'%s\' not found!", key)
53.                 raise QueueManagerException(key)
54.         except Exception as e:
55.             logging.getLogger('application').debug('Queue timeout for key {0}'.format(r
    epr(key)))
56.
57.
58.     # method to get queue with key
59.     @staticmethod
```

```
60.     def getQueue(key):
61.         if key in QueueManager.queues:
62.             logging.getLogger('application').debug("Get queue with key: \'%s\'", key)
63.             return QueueManager.queues[key]
64.         else:
65.             logging.getLogger('application').error("key \'%s\' not found!", key)
66.             raise QueueManagerException(key)
67.
68.
69. class QueueManagerException(Exception):
70.
71.     def __init__(self, key):
72.         self.key = key
73.
74.     def __str__(self):
75.         return "%s not found!" % repr(self.key)
```

managers/FilterManager.py

```
1.  import os
2.  import yaml
3.  from managers.FileManager import FileManager
4.
5.  class FilterManager():
6.
7.      configuration = None
8.      path_filter_list = ['Windows\\Prefetch',
9.                          'Windows\\LogFiles',
10.                         'Windows\\inf', 'Windows\\System32\\config',
11.                         'endpoint-loader-agent', '.PyCharm50',
12.                         'redBorder Malware EndPoint',
13.                         'System Volume Information',
14.                         'SPP',
15.                         'ntuser',
16.                         'AppData',
17.                         'Temp',
18.                         'ProgramData',
19.                         'Recycle',
20.                         'NTUSER',
21.                         'RECYCLE']
22.
23.     type_filter_list = []
24.
25.     @staticmethod
26.     def load_config(config, path):
27.         #reading configuration files content
28.         FilterManager.configuration = config.getConfigProperties()
29.         filter_type = os.path.join(path, 'filters', "file_filters.yml")
30.         FileManager.openFile(filter_type, 'type_filter', FileManager.READ_MODE)
31.         content_type = yaml.load(FileManager.readFileContent('type_filter'))
32.         FilterManager.type_filter_list = content_type['file_filters']
33.
34.
35.     @staticmethod
36.     def get_parameter(parameter):
37.         # return a configuration parameter
38.         return FilterManager.configuration[parameter]
39.
40.     @staticmethod
41.     def get_filter_type(type_string):
42.         return type_string in FilterManager.type_filter_list
43.
44.     @staticmethod
45.     def get_filter_path(filename):
46.         return any(path in filename for path in FilterManager.path_filter_list)
47.
48.     @staticmethod
49.     def get_tamanio_filter(filename):
50.         # get a file size
51.         match = False
52.         tamanio_config = int(FilterManager.get_parameter('file_size'))
53.         tamanio = os.stat(filename).st_size
54.         if tamanio:
55.             if tamanio > tamanio_config:
56.                 match = True
57.         else:
58.             match = "error"
59.
60.         return match
```

managers/FileManager.py

```
1.  import logging
2.  import os
3.
4.
5.  # Static class for file management
6.  class FileManager():
7.
8.      # modes
9.      WRITE_MODE = 'w'
10.     READ_MODE = 'r'
11.     APPEND_MODE = 'a'
12.     WRITE_BINARY_MODE = 'wb'
13.     READ_BINARY_MODE = 'rb'
14.
15.     # dict of files
16.     openedfiles = dict()
17.
18.     # method to open files and add to dict
19.     @staticmethod
20.     def openFile(path, key, mode):
21.
22.         if key and path and mode:
23.             if not key in FileManager.openedfiles:
24.                 file = open(path, mode=mode)
25.                 FileManager.openedfiles[key] = file
26.                 logging.getLogger('application').info("Created new file with key \'%s\'
    and mode \'%s\' in path \'%s\'", key, mode, path)
27.             else:
28.                 logging.getLogger('application').error("Key \'%s\' exist", key)
29.         elif path and mode and not key:
30.             return open(path, mode=mode)
31.
32.
33.     # method to close file with key
34.     @staticmethod
35.     def closeFile(key):
36.         if key in FileManager.openedfiles:
37.             FileManager.openedfiles[key].close()
38.             FileManager.openedfiles.__delitem__(key)
39.             logging.getLogger('application').info("Closed file with key \'%s\'", key)
40.         else:
41.             logging.getLogger('application').error("Key \'%s\' not found", key)
42.             raise FileManagerException(key)
43.
44.
45.     # method to write to file with key
46.     @staticmethod
47.     def writeInFile(key, msg):
48.         if key in  FileManager.openedfiles:
49.             FileManager.openedfiles[key].write("%s\n" % msg)
50.             logging.getLogger('application').info("Write new message in file with key:
    \'%s\'", key)
51.             logging.getLogger('application').debug("Message: %s", msg)
52.         else:
53.             logging.getLogger('application').error("Key \'%s\' not found", key)
54.             raise FileManagerException(key)
55.
56.
57.     # method to read from file with key
58.     @staticmethod
59.     def readFileContent(key):
60.         if key in FileManager.openedfiles:
61.             logging.getLogger('application').info("Readed content from file with key: \
    '%s\'", key)
```

```
62.             return FileManager.openedfiles[key].read()
63.         else:
64.             logging.getLogger('application').error("Key \'%s\' not found", key)
65.             raise FileManagerException(key)
66.
67.     # method to read from file with key
68.     @staticmethod
69.     def readFileContentLines(key):
70.         if key in FileManager.openedfiles:
71.             logging.getLogger('application').info("Readed content from file with key: \
    '%s\'", key)
72.             return FileManager.openedfiles[key].readlines()
73.         else:
74.             logging.getLogger('application').error("Key \'%s\' not found", key)
75.             raise FileManagerException(key)
76.
77.     # method to get file descriptor with key
78.     @staticmethod
79.     def getFileDescriptor(key):
80.         if key in FileManager.openedfiles:
81.             logging.getLogger('application').info("Getted file descriptor with key: \'%
    s\'", key)
82.             return FileManager.openedfiles[key]
83.         else:
84.             logging.getLogger('application').error("Key \'%s\' not found", key)
85.             raise FileManagerException(key)
86.
87.
88.     # method to get file status
89.     # return:
90.     #   st_mode         :    File mode
91.     #   st_ino          :     Inode number
92.     #   st_dev          :     Identifier of the device on which this file resides.
93.     #   st_nlink        :   Number of hard links.
94.     #   st_uid          :     User identifier of the file owner.
95.     #   st_gid          :     Group identifier of the file owner.
96.     #   st_size         :    Size of the file in bytes
97.     #   st_atime        :   Time of most recent access expressed in seconds
98.     #   st_mtime        :   Time of most recent content modification expressed in secon
    ds.
99.     #   st_ctime        :   Platform dependent
100.          #                           * the time of most recent metadata change on Uni
    x
101.          #                           * the time of creation on Windows, expressed in
    seconds
102.          #   st_atime_ns     :   Time of most recent access expressed in nanoseconds
    as an integer
103.          #   st_mtime_ns     :   Time of most recent content modification expressed i
    n nanoseconds as an integer
104.          #   st_ctime_ns     :   Platform dependent
105.          #                           * the time of most recent metadata change on Uni
    x
106.          #                           * the time of creation on Windows, expressed in
    nanoseconds as an integer
107.          @staticmethod
108.          def getFileInfo(path=None, key=None):
109.
110.              if key:
111.                  if key in FileManager.openedfiles:
112.                      fileDescriptor = FileManager.openedfiles[key]
113.                      return os.fstat(fileDescriptor)
114.              elif path:
115.                  return os.stat(path)
116.
117.          # method to close all opened files
118.          @staticmethod
119.          def closeAllFiles():
```

```
120.                 if bool(FileManager.openedfiles):
121.                     logging.getLogger('application').info("Close all files ...")
122.                     for fileKey in FileManager.openedfiles:
123.                         FileManager.openedfiles[fileKey].close()
124.                         logging.getLogger('application').info("Closed file with key: \'%
    s\'", fileKey)
125.
126.                     FileManager.openedfiles.clear()
127.
128.
129.         class FileManagerException(Exception):
130.
131.             def __init__(self, key):
132.                 self.key = key
133.
134.             def __str__(self):
135.                 return "%s not found!" % repr(self.key)
```

managers/FileParameters.py

```
1.  import os
2.  import hashlib
3.  import time
4.  import binascii
5.  import win32security
6.
7.  class FileParameters():
8.
9.      path = None
10.
11.     @staticmethod
12.     def config(path):
13.         FileParameters.path = path
14.
15.     @staticmethod
16.     def get_file_size(filename):
17.         if os.path.exists(filename):
18.             sizeFilename = None
19.             try:
20.                 sizeFilename = os.stat(filename).st_size
21.                 return sizeFilename
22.             except:
23.                 return None
24.
25.     @staticmethod
26.     def get_md5(filename, timer):
27.         #get md5 hash value from file
28.         md5_hash = None
29.         if os.path.exists(filename):
30.             #flag to finish function
31.             flag = True
32.             # Max intents
33.             count = 0
34.             while flag==True:
35.                 try:
36.                     #sentences must be split because CPU performance is better
37.                     file_open = open(filename, "r+b")
38.                     content = file_open.read()
39.                     md5_normal = hashlib.md5(content)
40.                     md5_hash = md5_normal.hexdigest()
41.                     file_open.close()
42.                     flag = False
43.                 except:
44.                     if count < 20:
45.                         time.sleep(timer)
46.                         count += 1
47.                     else:
48.                         flag = False
49.         return md5_hash
50.
51.     @staticmethod
52.     def get_sha256(filename, timer):
53.         sha_256 = None
54.         if os.path.exists(filename):
55.             # flag to finish function
56.             flag = True
57.             # Max intents
58.             count = 0
59.             while flag==True:
60.                 try:
61.                     #sentences must be split because CPU performance is better
62.                     file_open = open(filename, "rb")
63.                     content = file_open.read()
64.                     #pysha3 build a sha256 in SSE and CPU is decrement
```

```
65.                     sha_normal = hashlib.sha256(content)
66.                     sha_256 = sha_normal.hexdigest()
67.                     file_open.close()
68.                     flag = False
69.                 except:
70.                     if count < 20:
71.                         time.sleep(timer)
72.                         count += 1
73.                     else:
74.                         flag = False
75.         return sha_256
76.
77.     @staticmethod
78.     def get_file_type(filename):
79.         # get a type from file
80.         try:
81.             message = open(filename, 'rb').read()
82.             dir_file = os.path.join(FileParameters.path,'filters', 'snort_file.txt')
83.             snort_file = open(dir_file, 'r').readlines()
84.             tam = 0
85.             tabla_uno = []
86.             #read line by line a snort file
87.             while tam < len(snort_file):
88.                 words = snort_file[tam].split()
89.                 # if last word is 1 meaning that next line must be satisfy
90.                 if words[-1] == "1":
91.                      while words[-1] == "1":
92.                          #variable final is a last byte to check
93.                          final = int(words[1]) + int(words[0])
94.                          #the content of a file limited by an initial byte and other en
    d is caught in hexadecimal
95.                          content = binascii.hexlify(message[int(words[0]):final])
96.                          match = str(content.decode('ascii')).upper()
97.                          #If the content matches the typical of a file of that type com
    ing in our snort put 1 in the table file. In other case put 0
98.                          if match == words[2]:
99.                              tabla_uno.append(1)
100.                                 else:
101.                                     tabla_uno.append(0)
102.                                 tam += 1
103.                                 words = snort_file[tam].split()
104.                             words =  snort_file[tam].split()
105.                             final = int(words[1]) + int(words[0])
106.                             content = binascii.hexlify(message[int(words[0]):final])
107.                             match = str(content.decode('ascii')).upper()
108.                             if match == words[2]:
109.                                 tabla_uno.append(1)
110.                             else:
111.                                 tabla_uno.append(0)
112.                             flag = 1
113.                             for attribute in tabla_uno:
114.                                 if attribute == 0:
115.                                     flag = 0
116.                             if flag == 1:
117.                                 return words[3]
118.                             tabla_uno = []
119.                             tam += 1
120.                         else:
121.                             final = int(words[1]) + int(words[0])
122.                             content = binascii.hexlify(message[int(words[0]):final])
123.                             match = str(content.decode('ascii')).upper()
124.                             if match == words[2]:
125.                                 return words[3]
126.                             tam += 1
127.             except:
128.                 type_file_none = 'None'
```

```
129.                     return type_file_none
130.
131.             @staticmethod
132.             def get_acl(filename):
133.                 try:
134.                     #Get dacl(data access control list)
135.                     security_file = win32security.GetFileSecurity(filename, win32securit
    y.DACL_SECURITY_INFORMATION)
136.                     #Get DACL Descriptor.
137.                     dacl = security_file.GetSecurityDescriptorDacl()
138.                     if dacl != None and dacl != '':
139.                         string_dacl = str(dacl)
140.                         parameter_dacl = string_dacl.split()[-1]
141.                         # Take permits and turn them into hexadecimal
142.                         permission = parameter_dacl[0:-1]
143.                         permission_decimal_format = int(permission, 16)
144.                         return permission_decimal_format
145.                     else:
146.                         return None
147.                 except:
148.                     return None
149.                     pass
```

managers/WindowsValueManager.py

```
1.  import socket
2.  import wmi
3.  import logging
4.
5.  class WindowsValueManager():
6.
7.      #windows values dictionary
8.      values = dict()
9.
10.     # Method to get MAC, dns domain, IP, IP subnet, profile and hostname in PC.
11.     @staticmethod
12.     def set_values():
13.         try:
14.             WindowsValueManager.values['sensor_name'] = socket.gethostbyname_ex (socket
    .gethostname ())[0]
15.             for parameters_network in wmi.WMI ().Win32_NetworkAdapterConfiguration ():

16.                 if parameters_network.MACAddress != None:
17.                     WindowsValueManager.values['client_mac'] = parameters_network.MACAd
    dress.lower()
18.
19.                 if parameters_network.DNSDomain != None:
20.                     WindowsValueManager.values['domain_name'] = parameters_network.DNSD
    omain
21.
22.                 if parameters_network.IPAddress != None:
23.                     WindowsValueManager.values['endpoint_agent'] = parameters_network.I
    PAddress[0]
24.                     WindowsValueManager.values['dst'] = parameters_network.IPAddress[0]

25.                     WindowsValueManager.values['sensor_ip'] = parameters_network.IPAddr
    ess[0]
26.
27.                 if parameters_network.IPSubnet != None:
28.                     WindowsValueManager.values['src_net_name'] = parameters_network.IPS
    ubnet[0]
29.
30.             for parameters_profile in wmi.WMI ().Win32_NetworkLoginProfile ():
31.                 if parameters_profile.Name != None:
32.                     WindowsValueManager.values['client_id'] = parameters_profile.Name
33.         except Exception as e:
34.             logging.getLogger('application').error(e)
35.
36.
37.
38.     @staticmethod
39.     def get_values():
40.         return WindowsValueManager.values
```

Anexo C: Código del módulo controlador

Controllers/MonitorController.py

```
1.  from concurrent.futures import ThreadPoolExecutor
2.  import logging
3.
4.  class MonitorController():
5.
6.      # Constructor
7.      def __init__(self):
8.          self.executorService = ThreadPoolExecutor(max_workers=100)
9.          self.workerList = list()
10.         self.taskList = list()
11.         logging.getLogger('application').info("Created new monitor controller")
12.
13.
14.     # Reload monitor controller
15.     def reload(self):
16.         pass
17.
18.
19.     # Run single task
20.     def runTask(self, task, replication=1):
21.         # Deploy task with replication
22.
23.         self.taskList.append(task)
24.
25.         task.prepare()
26.
27.         for replica in range(replication):
28.             self.workerList.append(self.executorService.submit(task.doTask))
29.             logging.getLogger('application').info("Worker for task {0} created".format(
    repr(task.getTaskName())))
30.
31.
32.     # Get result from every worker and return a list
33.     def getWorkersResults(self):
34.         results = list()
35.         for worker in self.workerList:
36.             results.append(worker.result())
37.
38.         return results
39.
40.
41.     def stopWorkers(self):
42.
43.         logging.getLogger('application').info("Stopping workers")
44.
45.         up_workers = len(self.workerList)
46.         down_workers = 0
47.
48.         for task in self.taskList:
49.             task.shutdown()
50.
51.         for worker in self.workerList:
52.             worker.cancel()
```

```
53.             down_workers += 1
54.
55.         logging.getLogger('application').info("Stopped %d of %d workers", down_workers,
    up_workers)
56.
57.
58.     def getWorkersStatus(self):
59.         pass
```

controllers/tasks/Task.py

```
1.  from enum import Enum
2.  import logging
3.
4.  class Task():
5.
6.      class Status(Enum):
7.          # task is starting (config for example)
8.          STARTING = 0
9.
10.         # task ready for begin
11.         READY = 1
12.
13.         # task is running
14.         RUNNING = 2
15.
16.         # task is trying stop
17.         STOPPING = 3
18.
19.         # task is stopped
20.         STOPPED = 4
21.
22.         # task fail
23.         FAILED = 5
24.
25.         # task cancelled
26.         CANCELLED = 6
27.
28.         # task finished
29.         FINISHED = 7
30.
31.         # task aborted
32.         ABORTED = 8
33.
34.     client_id = None
35.
36.     def __init__(self, taskName=None, config=None):
37.         if taskName:
38.             self.taskName = taskName
39.
40.         self.configuration = config
41.
42.         if config:
43.             Task.client_id = config['client_id']
44.
45.
46.         if not self.configuration and self.taskName:
47.             logging.getLogger('application').warning('Configuration is not set for task
    {0}'.format(repr(self.taskName)))
48.
49.         self.status = Task.Status.STOPPED
50.
51.     # Method to prepare the task
52.     def prepare(self):
53.         # If task is stopped pass to starting
54.         if self.status is Task.Status.STOPPED: self.status = Task.Status.STARTING
55.
56.         # Call to prepare implementation
57.         self.prepareImpl()
58.
59.         # If task is starting pass to ready
60.         if self.status is Task.Status.STARTING: self.status = Task.Status.READY
61.
62.         if self.taskName: logging.getLogger('application').info("Task %s prepared" % re
    pr(self.taskName))
```

```
63.
64.     # Method to run the task
65.     def doTask(self):
66.         # If task is ready then pass to starting
67.         if self.status is Task.Status.READY: self.status = Task.Status.RUNNING
68.         # Call to do task implementation and get the result
69.
70.         if self.taskName: logging.getLogger('application').info("Running task %s" % rep
    r(self.taskName))
71.
72.         lastResult = self.doTaskImpl()
73.
74.         if self.status is Task.Status.RUNNING: self.status = Task.Status.FINISHED
75.
76.         if self.taskName: logging.getLogger('application').info("Task %s done" % repr(s
    elf.taskName))
77.
78.         return lastResult
79.
80.
81.     # Method to shutdown the task
82.     def shutdown(self):
83.         if self.status is Task.Status.RUNNING: self.status = Task.Status.STOPPING
84.
85.         if self.taskName: logging.getLogger('application').info("Stopping task %s" % re
    pr(self.taskName))
86.
87.         # Call to shutdown implementation
88.         self.shutdownImpl()
89.
90.         if self.status is Task.Status.STOPPED: self.status = Task.Status.ABORTED
91.
92.         if self.status is Task.Status.STOPPING: self.status = Task.Status.STOPPED
93.
94.         if self.taskName: logging.getLogger('application').info("Task %s stopped" % rep
    r(self.taskName))
95.
96.
97.     # Get task name
98.     def getTaskName(self):
99.         return self.taskName
100.
101.            # Set task name
102.            def setTaskName(self, taskName):
103.                self.taskName = taskName
104.
105.            # Get task status
106.            def getTaskStatus(self):
107.                return self.status
108.
109.            # Get last result
110.            def getLastResult(self):
111.                return self.lastResult
112.
113.            # Shutdown user implementation
114.            def shutdownImpl(self):
115.                pass
116.
117.            # DoTask user implementation
118.            def doTaskImpl(self):
119.                pass
120.
121.            # Preprare task implementation
122.            def prepareImpl(self):
123.                pass
```

controllers/tasks/windows/MonitorDiskPartition.py

```
1.  import os
2.  import threading
3.  import time
4.  import win32con
5.  import win32file
6.  import wmi
7.  import logging
8.  import pythoncom
9.  from controllers.tasks.Task import Task
10. from managers.FilterManager import FilterManager
11. from managers.WindowsValueManager import WindowsValueManager
12. from managers.QueueManager import QueueManager
13.
14.
15. class MonitorDiskPartition(Task):
16.
17.     def prepareImpl(self, config=None):
18.         pass
19.
20.     def doTaskImpl(self):
21.         try:
22.             pythoncom.CoInitialize()
23.             connection = wmi.WMI()
24.             # lista que controla que particion logica del disco existe y esta activo en
     el sistema y cual no
25.             threads_state = []
26.             # lista de particion logica del disco.
27.             threads_list = []
28.             # lista con las posibles particiones logicas del discos del sistema
29.             partitions_logical_disk = []
30.
31.             while self.status is Task.Status.RUNNING:
32.                 try:
33.                     WindowsValueManager.set_values()
34.                     # si es una particion logica del disco fisico se incluira a la list
    a para monitorizarlo y evitar asi los discos de red
35.                     for physical_disk in connection.Win32_DiskDrive():
36.                         for partition in physical_disk.associators("Win32_DiskDriveToDi
    skPartition"):
37.                             for logical_disk in partition.associators("Win32_LogicalDis
    kToPartition"):
38.                                 if len(partitions_logical_disk) == 0:
39.                                     # si la lista de discos esta vacia es porque acabam
    os de empezar
40.                                     # se introducen en la lista el disco encontrado
41.                                     disk_partition = logical_disk.Caption + "\\"
42.                                     partitions_logical_disk.append(disk_partition)
43.                                     threads_state.append(0)
44.                                 else:
45.                                     # si existe en la lista algun disco se compara
46.                                     # si coincide no se introduce en la lista (match =
    1)
47.                                     # si no existe ningun disco en la lista igual que e
    se, se introduce y se mete en la lista de estados en su misma posicion un 0.
48.                                     match = 0
49.                                     for element in partitions_logical_disk:
50.                                         disk_partition = logical_disk.Caption + "\\"
51.                                         if element == disk_partition:
52.                                             match = 1
53.
54.                                     if match == 0:
55.                                         disk_partition = logical_disk.Caption + "\\"
56.                                         partitions_logical_disk.append(disk_partition)
```

```
                                        threads_state.append(0)
                    threads_position = 0
                    # se recorre la lista de discos y comprobamos que su ruta existe
                    for path_logical_disk in partitions_logical_disk:
                        if os.path.exists(path_logical_disk):
                            # si existe la ruta y el disco estaba inactivo se crea hilo

                            # como estaba inactivo ponemos un uno en referencia a que a
hora si lo esta en su posicion correspondiente
                            if threads_state[threads_position] == 0:
                                thread_monitor_disk = threading.Thread(target=self.chan
ges_files, args=(path_logical_disk,),name=path_logical_disk)
                                threads_list.append(thread_monitor_disk)
                                thread_monitor_disk.start()
                            threads_state[threads_position] = 1
                        # Si no existe la ruta pero el disco estaba activo en la ultima
 comprobacion se pone como inactivo en su posicion correspondiente.
                        else:
                            if threads_state[threads_position] == 1:
                                threads_state[threads_position] = 0
                        threads_position = threads_position + 1

                    time.sleep(2)
                except:
                    continue
        except Exception as e:
            logging.getLogger('application').error(e)
        finally:
            pythoncom.CoUninitialize()

    def shutdownImpl(self):
        pass

    def changes_files(self, logical_disk):
        # Acciones que pueden ocurrir para que salte un evento
        ACTIONS = {
            1: "Created",
            2: "Deleted",
            3: "Updated",
            4: "Renamed from something",
            5: "Renamed to something"
        }

        FILE_LIST_DIRECTORY = 0x0001

        # Ruta a monitorizar
        path_to_watch = logical_disk
               # configuracion de la monitorizacion
               hDir = win32file.CreateFile(
                   path_to_watch,
                   FILE_LIST_DIRECTORY,
                   win32file.FILE_SHARE_READ | win32file.FILE_SHARE_WRITE | win32file.F
ILE_SHARE_DELETE,
                   None,
                   win32file.OPEN_EXISTING,
                   win32con.FILE_FLAG_OVERLAPPED | win32con.FILE_FLAG_BACKUP_SEMANTICS,

                   None
               )
               exit_thread = 1
               while exit_thread == 1:

                   try:

                       # Monitorizacion (ruta, buffer, subcarpetas a monitorizar (si o
no), cambios con los que salta el evento)
```

```
116.                         results = win32file.ReadDirectoryChangesW(
117.                             hDir,
118.                             5012,
119.                             True,
120.                             win32con.FILE_NOTIFY_CHANGE_FILE_NAME |
121.                             win32con.FILE_NOTIFY_CHANGE_DIR_NAME |
122.                             win32con.FILE_NOTIFY_CHANGE_ATTRIBUTES |
123.                             win32con.FILE_NOTIFY_CHANGE_SIZE |
124.                             win32con.FILE_NOTIFY_CHANGE_LAST_WRITE |
125.                             win32con.FILE_NOTIFY_CHANGE_SECURITY,
126.                             None,
127.                             None
128.                         )
129.                         # mientras exista resultados asociados a un evento
130.                         send_list = []
131.
132.                         # Del evento capturado se coge la accion y el archivo implicado
    en el evento.
133.                         for action, file in results:
134.                             # completamos el nombre del fichero implicado en un evento.

135.                             full_filename = os.path.join(path_to_watch, file)
136.                             temporal_list = [action, full_filename]
137.                             # introducimos en la lista de eventos la accion y el fichero
     implicado.
138.                             match = FilterManager.get_filter_path(full_filename)
139.                             send_list.append(temporal_list)
140.                             # si la ruta existe y es un fichero.
141.                             if os.path.exists(full_filename) and os.path.isfile(full_fil
    ename):
142.                                 # si el fichero no esta filtrado por ruta y la accion no
     es borrar, es decir, el fichero existe para trabajar con el mismo.
143.                                 if match == False and action != 2:
144.                                     file_action = dict()
145.                                     file_action['filename'] = full_filename
146.                                     # se asigna una accion dependiendo del numero relaci
    onado a la accion
147.                                     if action == 1:
148.                                         file_action['action'] = 'created'
149.                                     else:
150.                                         file_action['action'] = 'updated'
151.                                     # se encola el evento en la cola normal
152.                                     QueueManager.putOnQueue('file_capture',file_action)

153.
154.                                 # en caso de ser un directorio solo se mandara el evento
     si es nuevo.
155.                             if os.path.isdir(full_filename) and action == 1 and match ==
     False:
156.                                 #si no esta filtrado el directorio por ruta se trabajara
     con el siempre y cuando se haya creado, ya que si es actualizado sera un fichero de de
    ntro.
157.                                 file_action = dict()
158.                                 file_action['filename'] = full_filename
159.                                 file_action['action'] = 'created'
160.                                 QueueManager.putOnQueue('file_capture',file_action)
161.                         # se mete toda la informacion asociado a un evento en la cola de
     informacion
162.                         QueueManager.putOnQueue('file_event',send_list)
163.                         # si la ruta raiz (en este caso la particion de windows, ya que
    puede ser un usb y desaparecer) deja de existir se acaba el bucle.
164.                         if not os.path.exists(path_to_watch):
165.                             exit_thread = 0
166.
167.                     except Exception:
168.                         continue
```

controllers/tasks/general/FileCaptureTask.py

```
1.  import json
2.  import logging
3.  import os
4.  import time
5.  from controllers.tasks.Task import Task
6.  from managers.CacheManager import CacheManager
7.  from managers.FileParameters import FileParameters
8.  from managers.FilterManager import FilterManager
9.  from managers.S3Manager import S3Manager
10. from managers.WindowsValueManager import WindowsValueManager
11. from managers.QueueManager import QueueManager
12.
13.
14. class FileCaptureTask(Task):
15.
16.     def prepareImpl(self):
17.         pass
18.
19.     def doTaskImpl(self):
20.
21.         while self.status is Task.Status.RUNNING:
22.             # Get item from file_capture queue with timeout
23.             fileStats = QueueManager.getItemFromQueue('file_capture', timeout=15)
24.
25.             try:
26.                 if fileStats:
27.
28.                     logging.getLogger('application').debug('Getted new file {0}'.format
    (fileStats))
29.
30.                     filename = fileStats['filename']
31.                     action = fileStats['action']
32.
33.                     if (os.path.exists(filename) and os.path.isfile(filename)):
34.                         self.treatFile(filename, action)
35.                         time.sleep(0.5)
36.                     elif (os.path.isdir(filename) and os.path.exists(filename)):
37.
38.                         for dirpath, dirnames, filenames in os.walk(filename):
39.
40.                             for files in filenames:
41.                                 path = os.path.join(dirpath, files)
42.
43.                                 if os.path.exists(path):
44.                                     self.treatFile(path, action)
45.                                     time.sleep(0.5)
46.
47.             except Exception as e:
48.                 if self.status is Task.Status.RUNNING:
49.                     logging.getLogger('application').error(str(e))
50.                 continue
51.
52.
53.     def shutdownImpl(self):
54.         pass
55.
56.
57.     # Open files with retries
58.     def file_open(self, filename, timeout):
59.
60.         self.flag = False
61.         count = 0
62.
63.         while (count < 20) and (self.flag == False):
```

```
64.             try:
65.                 if os.path.exists(filename):
66.                     filename_open = open(filename, 'rb')
67.                     filename_open.close()
68.                     self.flag = True
69.                 else:
70.                     count = 20
71.             except Exception:
72.                 time.sleep(timeout)
73.                 count += 1
74.                 continue
75.
76.
77.     def treatFile(self, filename, action):
78.
79.         if os.path.exists(filename):
80.
81.             timer = 2
82.             self.file_open(filename, timer)
83.
84.             if self.flag == True:
85.
86.                 # Get if is filtered by size
87.                 filtered_by_size = FilterManager.get_tamanio_filter(filename)
88.
89.                 # Get file type
90.                 type_file = FileParameters.get_file_type(filename)
91.
92.                 time.sleep(1)
93.
94.                 # Get if is filtered by type
95.                 filtered_by_type = FilterManager.get_filter_type(type_file)
96.
97.                 # If file is filtered by size or type
98.                 if filtered_by_type or filtered_by_size:
99.
100.                            file_slow_queue = dict()
101.                            file_slow_queue['filename'] = filename
102.                            file_slow_queue['action'] = action
103.                            file_slow_queue['type'] = type_file
104.
105.                            QueueManager.putOnQueue('slow', file_slow_queue)
106.
107.                        else:
108.                            # New event
109.                            event = dict()
110.
111.                            # Identifiers
112.                            event['endpoint_uuid'] = Task.client_id
113.                            event['namespace_uuid'] = Task.client_id
114.
115.                            # Calculate sha256
116.                            sha256 = FileParameters.get_sha256(filename, timer)
117.
118.                            time.sleep(2)
119.
120.                            if sha256 is not None:
121.
122.                                # Event type
123.                                event['type'] = 'file_capture'
124.
125.                                event['action'] = action
126.                                event['filepath'] = filename
127.                                event['file_extension'] = os.path.splitext(filename)[1]
128.                                event['filename'] = os.path.basename(filename)
129.                                event['sha256'] = sha256
```

```
130.
131.                             if type_file is not None and type_file is not 'None':
132.                                 event['file_type'] = type_file
133.
134.                             # Calculate md5
135.                             md5_hash = FileParameters.get_md5(filename, timer)
136.
137.                             time.sleep(1)
138.
139.                             if md5_hash is not None:
140.                                 event['md5'] = md5_hash
141.
142.                             acl = FileParameters.get_acl(filename)
143.                             time.sleep(1)
144.
145.                             if acl is not None:
146.                                 event['acl'] = acl
147.
148.                             find_new = CacheManager.checkIfExists(sha256)
149.
150.                             if find_new is False:
151.
152.                                 # Upload to S3
153.                                 result_s3 = S3Manager.upload_s3(filename, sha256)
154.                                 time.sleep(1)
155.
156.                                 if result_s3 is 'error_copy':
157.                                     logging.getLogger('application').error('Error to cre
    ate a temporally file to upload to S3')
158.                                 elif result_s3 is 'error_upload':
159.                                     logging.getLogger('application').error('Error to upl
    oad {0} to S3'.format(repr(filename)))
160.                                 elif result_s3 is 'upload':
161.                                     score = -1
162.                                     event['cache_score'] = score
163.                                     logging.getLogger('application').info('{0} uploaded
    to S3'.format(filename))
164.                                     self.enrichment(filename, sha256, score)
165.
166.                             elif find_new is True:
167.
168.                                 score = CacheManager.get_score(sha256)
169.
170.                                 if score != 1000:
171.                                     event['cache_score'] = score
172.                                     self.enrichment(filename, sha256, score)
173.
174.                             event['timestamp'] = int(time.time())
175.
176.                             json_event = json.dumps(event)
177.
178.                             logging.getLogger('binnacle').info(json_event)
179.                             logging.getLogger('application').debug('Sending kafka info :
     {0}'.format(json_event))
180.
181.                             # Create new kafka message
182.                             kafka_message = dict()
183.                             kafka_message['topic'] = 'rb_ioc'
184.                             kafka_message['content'] = json_event
185.
186.                             # Put message in kafka queue
187.                             QueueManager.putOnQueue('kafka', kafka_message)
188.                     else:
189.                         logging.getLogger('application').error('Error to open ' + filena
    me)
190.
```

```
191.
192.             def enrichment(self, filename, sha256, score):
193.                 values = WindowsValueManager.get_values()
194.                 name_file = os.path.basename(filename)
195.                 values['file_uri'] = filename
196.                 values['file_name'] = name_file
197.                 values['hash'] = sha256
198.                 values['action'] = 'forward'
199.                 values['timestamp'] = int(time.time())
200.                 values['endpoint_uuid'] = Task.client_id
201.
202.                 sizeFilename = FileParameters.get_file_size(filename)
203.                 if sizeFilename != None:
204.                     values['file_size'] = sizeFilename
205.                 else:
206.                     logging.getLogger('application').error('Error to get size of ' + fil
    ename)
207.
208.                 if score != 1000:
209.                     values['hash_probe_score'] = score
210.
211.                 message = json.dumps(values)
212.
213.                 kafka_message = dict()
214.                 kafka_message['content'] = message
215.                 kafka_message['topic'] = 'rb_endpoint'
216.
217.                 QueueManager.putOnQueue('kafka', kafka_message)
```

controllers/tasks/general/FileCaptureSlowTask.py

```
1.  import json
2.  import logging
3.  import os
4.  import time
5.  from controllers.tasks.Task import Task
6.  from managers.FileParameters import FileParameters
7.  from managers.QueueManager import QueueManager
8.  from managers.WindowsValueManager import WindowsValueManager
9.
10.
11. class FileCaptureSlowTask(Task):
12.
13.     def prepareImpl(self, config=None):
14.         pass
15.
16.     def doTaskImpl(self):
17.
18.         while self.status is Task.Status.RUNNING:
19.
20.             # Get item with timeout
21.             fileStats = QueueManager.getItemFromQueue('slow', timeout=15)
22.
23.             try:
24.                 if fileStats:
25.
26.                     logging.getLogger('application').debug('Getted new file {0}'.format
    (fileStats))
27.
28.                     filename = fileStats['filename']
29.                     file_type = fileStats['type']
30.                     action = fileStats['action']
31.
32.                     event = dict()
33.                     event['endpoint_uuid'] = str(Task.client_id)
34.                     event['namespace_uuid'] = str(Task.client_id)
35.
36.                     if os.path.exists(filename):
37.                         timer = 10
38.
39.                         sha256 = FileParameters.get_sha256(filename, timer)
40.
41.                         time.sleep(2)
42.
43.                         if sha256:
44.                             event['type'] = 'file_capture'
45.                             event['action'] = action
46.                             event['filepath'] = filename
47.                             event['file_extension'] = os.path.splitext(filename)[1]
48.                             event['filename'] = os.path.basename(filename)
49.                             event['sha256'] = sha256
50.
51.
52.                         if file_type != None and file_type != 'None':
53.                             event['file_type'] = file_type
54.
55.                         md5_hash = FileParameters.get_md5(filename, timer)
56.
57.                         if md5_hash:
58.                             event['md5'] = md5_hash
59.
60.                         acl = FileParameters.get_acl(filename)
61.
62.                         if acl != None:
63.                             event['acl'] = acl
```

```
64.
65.                         event['timestamp'] = int(time.time())
66.
67.                         json_event = json.dumps(event)
68.
69.                         logging.getLogger('binnacle').info(json_event)
70.
71.                         # Create new message for kafka
72.                         kafka_message = dict()
73.                         kafka_message['topic'] = 'rb_ioc'
74.                         kafka_message['content'] = json_event
75.
76.                         # Put kafka message in kafka queue
77.                         QueueManager.putOnQueue('kafka', kafka_message)
78.
79.                         self.enrichment(filename, sha256, -1)
80.
81.                     else:
82.                         logging.getLogger('application').error('File {0} not exists'.fo
    rmat(filename))
83.
84.             except Exception as e:
85.                 if self.status is Task.Status.RUNNING:
86.                     logging.getLogger('application').info(str(e))
87.                 continue
88.
89.
90.     def shutdownImpl(self):
91.         pass
92.
93.     def enrichment(self, filename, sha256, score):
94.
95.         values = WindowsValueManager.get_values()
96.
97.         basename_file = os.path.basename(filename)
98.
99.         values['file_uri'] = filename
100.                values['file_name'] = basename_file
101.                values['hash'] = sha256
102.                values['action'] = 'forward'
103.                values['timestamp'] = int(time.time())
104.                values['endpoint_uuid'] = str(Task.client_id)
105.
106.                sizeFilename = FileParameters.get_file_size(filename)
107.                if sizeFilename != None:
108.                    values['file_size'] = sizeFilename
109.                else:
110.                    logging.getLogger('application').error('Error to get size of ' + fil
    ename)
111.
112.                if score != 1000:
113.                    values['hash_probe_score'] = score
114.
115.                message = json.dumps(values)
116.
117.                # Create new kafka message
118.                kafka_message = dict()
119.                kafka_message['content'] = message
120.                kafka_message['topic'] = 'rb_endpoint'
121.
122.                QueueManager.putOnQueue('kafka', kafka_message)
```

controllers/tasks/general/FileCaptureEventTask.py

```
1.  import logging
2.  import os
3.  import json
4.  from controllers.tasks.Task import Task
5.  from managers.FilterManager import FilterManager
6.  from managers.QueueManager import QueueManager
7.
8.
9.  class FileCaptureEventTask(Task):
10.
11.     def prepareImpl(self, config=None):
12.         pass
13.
14.
15.     def doTaskImpl(self):
16.
17.         while self.status is Task.Status.RUNNING:
18.
19.             try:
20.                 # Get item from file_event queue
21.                 event = QueueManager.getItemFromQueue('file_event', timeout=15)
22.
23.                 if event:
24.                     logging.getLogger('application').debug('Getted new file {0}'.format
    (event))
25.
26.                     for elements in event:
27.
28.                         # Is file filtered?
29.                         filtered = FilterManager.get_filter_path(elements[1])
30.
31.                         # Create new event
32.                         event = dict()
33.
34.                         # Client id
35.                         event['namespace_uuid'] = Task.client_id
36.                         event['endpoint_uuid'] = Task.client_id
37.
38.                         # Event type
39.                         event['type'] = 'file_information'
40.
41.                         if elements[0] == 2: # File is deleted
42.
43.                             if not filtered:
44.
45.                                 event['filename'] = elements[1]
46.                                 event['action'] = 'deleted'
47.
48.                         if elements[0] == 4: # File is renamed
49.
50.                             if not filtered:
51.                                 action = 'file ' + elements[1] +' renamed to '
52.
53.                         if elements[0] == 5:
54.
55.                             if not filtered and (os.path.isfile(elements[1]) or os.path
    .isdir(elements[1])):
56.
57.                                 action += elements[1]
58.
59.                                 event['filename'] = elements[1]
60.                                 event['action'] = action
61.
62.                         if 'action' in event:
```

```
63.
64.                                 # Dict to JSON
65.                                 json_event = json.dumps(event)
66.
67.                                 # Create new kafka message
68.                                 kafka_message = dict()
69.                                 kafka_message['topic'] = 'rb_ioc'
70.                                 kafka_message['content'] = json_event
71.
72.                                 # Logging in binnacle
73.                                 logging.getLogger('binnacle').info(json_event)
74.
75.                                 # And put message in kafka queue
76.                                 QueueManager.putOnQueue('kafka', kafka_message)
77.
78.             except Exception as e:
79.                 if self.status is Task.Status.RUNNING:
80.                     logging.getLogger('application').error(e)
81.
82.                 continue
83.
84.     def shutdownImpl(self):
85.         pass
```

controllers/tasks/windows/CreationProcessTask.py

```
1.  import json
2.  import logging
3.  import os
4.  import time
5.  import psutil
6.  import wmi
7.  import pythoncom
8.  import win32api
9.  import win32con
10. import win32process
11. from controllers.tasks.Task import Task
12. from managers.QueueManager import QueueManager
13.
14. class CreationProcessTask(Task):
15.
16.     def prepareImpl(self, config=None):
17.         pass
18.
19.
20.     def doTaskImpl(self):
21.         try:
22.
23.             pythoncom.CoInitialize()
24.             connection = wmi.WMI()
25.
26.             watcher = connection.Win32_Process.watch_for("creation")
27.             while self.status is Task.Status.RUNNING:
28.                 try:
29.
30.                     new_process = watcher()
31.                     event = dict()
32.
33.                     # Get executable
34.                     executable = self.get_executable(new_process, new_process.ProcessID
    , new_process.Caption, new_process.CommandLine, new_process.Description)
35.
36.                     event['namespace_uuid'] = Task.client_id
37.                     event['endpoint_uuid'] = Task.client_id
38.                     event['type'] = 'process'
39.                     event['action'] = 'creation'
40.
41.                     # Get PID if exists
42.                     if new_process.ProcessID is not None:
43.                         event['pid'] = new_process.ProcessID
44.
45.                     # Get caption if exists
46.                     if new_process.Caption is not None:
47.                         event['name'] = new_process.Caption
48.
49.                     # Get executable if exists
50.                     if executable is not None:
51.                         event['application'] = executable
52.
53.                     self.characterist(new_process, event)
54.
55.                     try:
56.
57.                        # Get parent process
58.                        parent_process = psutil.Process(new_process.ParentProcessId)
59.
60.                        if parent_process:
61.                            try:
62.                                # Get executable parent
```

```
                                executable_parent = self.get_parent_executable(new_proc
    ess.ParentProcessId, parent_process.name())
                                event['parent_application'] = executable_parent
                            except Exception as e:
                                logging.getLogger('application').error(e)
                        except Exception as e:
                            logging.getLogger('application').error(e)
                            continue

                        event['timestamp'] = int(time.time())

                        json_event = json.dumps(event)
                        logging.getLogger('binnacle').info(json_event)

                        # Create new kafka message
                        kafka_message = dict()
                        kafka_message['topic'] = 'rb_ioc'
                        kafka_message['content'] = json_event

                        # Put message in kafka queue
                        QueueManager.putOnQueue('kafka', kafka_message)

                    except Exception as e:
                        logging.getLogger('application').error(e)
                        continue

            except Exception as e:
                logging.getLogger('application').error(e)
            finally:
                pythoncom.CoUninitialize()


    def shutdownImpl(self):
        pass

    def characterist(self, process, bitacora_string):
        try:

                    # Get priority
                    if process.Priority:
                        bitacora_string['priority'] = process.Priority

                    #Get session ID
                    if process.SessionId:
                        bitacora_string['sessionId'] = process.SessionId
                except Exception as e:
                    logging.getLogger('application').error(e)
                    pass

            # Allow to get parent executable
            def get_parent_executable(self, pid, name):

                try:
                    # Get process by PID
                    executable = self.get_process_executable(pid)

                    # If not, get executable by name
                    if not executable:
                        executable = self.get_process_executable_by_name(name)

                    if not executable:
                        return None

                    else:
                        return executable

                except Exception as e:
```

```
129.                     logging.getLogger('application').error(e)
130.                     return None
131.                     pass
132.
133.
134.             # Allow to get executable by name
135.             def get_process_executable_by_name(self, filename, cmdline=None):
136.                 try:
137.
138.                     if not cmdline:
139.                         system_path = os.environ['WINDIR'] + "\\System32\\"
140.                         full_path = system_path + filename
141.                         if os.path.exists(full_path):
142.                             executable = full_path
143.                         else:
144.                             executable = None
145.
146.                     else:
147.                         if "\\SystemRoot\\".lower() in cmdline.lower():
148.                             os_path = os.environ['WINDIR']
149.                             system_path = cmdline.replace("\\SystemRoot",os_path)[:cmdli
    ne.rfind("\\")]
150.                             full_path = system_path + filename
151.                             if os.path.exists(full_path):
152.                                 executable = full_path
153.                             else:
154.                                 executable = None
155.                         else:
156.                             executable = None
157.                 except Exception as e:
158.                     logging.getLogger('application').error(e)
159.                     executable = None
160.                 return executable
161.
162.
163.             #Allow to get process by PID
164.             def get_process_executable(self, pid):
165.
166.                 if not pid or pid == 4:
167.                     return None
168.
169.                 try:
170.                     handle = win32api.OpenProcess(win32con.PROCESS_ALL_ACCESS, False, pi
    d)
171.                     executable = win32process.GetModuleFileNameEx(handle,0)
172.                     win32api.CloseHandle(handle)
173.                     if "\\SystemRoot\\".lower() in executable.lower():
174.                         executable = None
175.                     if "\\??\\" in executable:
176.                         executable = executable.replace("\\??\\", "")
177.
178.                 except Exception as e:
179.                     logging.getLogger('application').error(e)
180.                     executable = None
181.                 return executable
182.
183.
184.             # Allow to get executable
185.             def get_executable(self, process, pid, name, commandLine, description):
186.
187.                try:
188.
189.                   executable = process.ExecutablePath
190.
191.                   if not executable:
192.
```

```
193.                     # Try to get executable by PID
194.                     executable = self.get_process_executable(pid)
195.
196.                     if not executable:
197.                         executable = process.ExecutablePath
198.
199.                     if not executable:
200.                         # Try to get executable by name
201.                         executable = self.get_process_executable_by_name(name)
202.
203.                     if not executable:
204.
205.                         if commandLine:
206.                             # Try to get executable by name and CLI
207.                             executable = self.get_process_executable_by_name(name, com
    mandLine)
208.
209.                     if not executable:
210.
211.                         if commandLine:
212.                             # Try to get executable by description and CLI
213.                             executable = self.get_process_executable_by_name(descripti
    on, commandLine)
214.
215.                 if not executable:
216.                     return None
217.                 else:
218.                      return executable
219.             except:
220.                 pass
```

controllers/tasks/windows/DeletionProcessTask.py

```
import json
import logging
import time
import wmi
import pythoncom
from controllers.tasks.Task import Task
from managers.QueueManager import QueueManager


class DeletionProcessTask(Task):

    def prepareImpl(self, config=None):
        pass

    def doTaskImpl(self):

        try:
            pythoncom.CoInitialize()

            connection = wmi.WMI()

            watcher = connection.Win32_Process.watch_for("deletion")

            while self.status is Task.Status.RUNNING:
                try:
                    deletedProcess = watcher()

                    event = dict()
                    event['namespace_uuid'] = Task.client_id
                    event['endpoint_uuid'] = Task.client_id
                    event['type'] = 'process'
                    event['action'] = 'deletion'
                    event['name'] = deletedProcess.Caption
                    event['pid'] = deletedProcess.ProcessID
                    event['timestamp'] = int(time.time())

                    json_event = json.dumps(event)

                    logging.getLogger('binnacle').info(json_event)

                    # Create new kafka message
                    kafka_message = dict()
                    kafka_message['topic'] = 'rb_ioc'
                    kafka_message['content'] = json_event

                    # Put message in kafka queue
                    QueueManager.putOnQueue('kafka', kafka_message)

                except Exception as e:
                    logging.getLogger('application').error(e)
                    continue

        except Exception as e:
            logging.getLogger('application').error(e)

        finally:
            pythoncom.CoUninitialize()

    def shutdownImpl(self):
        pass
```

controllers/tasks/windows/WebProcessTask.py

```
1.  import json
2.  import logging
3.  import time
4.  import psutil
5.  import wmi
6.  import pythoncom
7.  from controllers.tasks.Task import Task
8.  from managers.QueueManager import QueueManager
9.
10.
11. class WebProcessTask(Task):
12.
13.     def prepareImpl(self, config=None):
14.         pass
15.
16.     def doTaskImpl(self):
17.
18.         try:
19.             pythoncom.CoInitialize()
20.             connection = wmi.WMI()
21.             #Se inicia la monitoricion de procesos en tiempo real, en este caso se moni
    torizaran los procesos que se crean.
22.             watcher = connection.Win32_Process.watch_for("creation")
23.
24.             while self.status is Task.Status.RUNNING:
25.                 try:
26.                     #Cuando un nuevo proceso se cree en el sistema entra en el try ante
    rior y capturamos el proceso creado.
27.                     new_process = watcher()
28.
29.                     try:
30.                         parent_process = psutil.Process(new_process.ParentProcessId)
31.                         soon_process = psutil.Process(new_process.ProcessID)
32.                         connections_opened = ''
33.                         try:
34.                             if parent_process:
35.                                 #si el proceso padre existe, capturamos sus conexiones d
    e red.
36.                                 #Si el proceso es de red tendra conexiones, sino dara un
     error y se ignorara el proceso.
37.                                 connections_opened = parent_process.connections(kind="al
    l")
38.                             else:
39.                                 if soon_process:
40.                                     connections_opened = soon_process.connections(kind="
    all")
41.
42.                             ip_list = []
43.                             ip_non_repeat_list = []
44.
45.                             for tupla in connections_opened:
46.                             #recorremos conexion a conexion, se coge la IP y puerto y se
     mete en la lista de IP.
47.                                 try:
48.                                     conexion = tupla[4]
49.                                     ip = conexion[0]
50.                                     puerto = conexion[1]
51.                                     if ip != None and ip !='' and puerto != None and pue
    rto !='':
52.                                         ip_string = str(ip) + ":" + str(puerto)
53.                                         ip_list.append(ip_string)
54.                                 except:
55.                                     continue
56.                             #Recorremos la lista de conexiones para coger las no repetid
    as, ya que hay algunas redundantes.
57.                             #Para saber si esta repetida usaremos una bandera a 0 o 1.
58.                             for element in ip_list:
```

```
                                    flag = 0
                                    for element_ip in ip_non_repeat_list:
                                        if element == element_ip:
                                            flag = 1
                                    #Si la conexion no se repite se guarda en la lista de las no repetidas.
                                    if flag == 0:
                                        ip_non_repeat_list.append(element)
                                new_string = ''
                                #Como cada proceso puede tener varias conexiones pues enviamos a la bitacora todas.
                                for ip_connection in ip_non_repeat_list:
                                    new_string += str(ip_connection) + '; '

                                if len(ip_non_repeat_list) > 0:

                                    event = dict()
                                    event['namespace_uuid'] = str(Task.client_id)
                                    event['endpoint_uuid'] = str(Task.client_id)
                                    event['type'] = 'connection'
                                    event['application'] = str(new_process.Caption)
                                    event['pid'] = new_process.ProcessID
                                    event['connections'] = new_string
                                    event['timestamp'] = int(time.time())

                                    json_event = json.dumps(event)

                                    logging.getLogger('binnacle').info(json_event)
                                    logging.getLogger('application').debug('data for kafka : {0}'.format(json_event))

                                    # Create new kafka message
                                    kafka_message = dict()
                                    kafka_message['topic'] = 'rb_ioc'
                                    kafka_message['content'] = json_event

                                    # Put message in kafka queue
                                    QueueManager.putOnQueue('kafka', kafka_message)

                                except Exception as e:
                                     logging.getLogger('application').error(e)
                                     continue

                            except Exception as e:
                                    logging.getLogger('application').error(e)
                                    continue

                                except Exception as e:
                                    logging.getLogger('application').error(e)
                                    continue

                        except Exception as e:
                                logging.getLogger('application').error(e)

                        finally:
                            pythoncom.CoUninitialize()


                def shutdownImpl(self):
                    pass
```

controllers/tasks/general/KafkaSenderTask.py

```
import logging
```

```
2.  import sys
3.
4.  from controllers.tasks.Task import Task
5.  from managers.KafkaManager import KafkaManager
6.
7.  from managers.QueueManager import QueueManager
8.
9.  class KafkaSenderTask(Task):
10.
11.
12.     def prepareImpl(self):
13.
14.         # create new kafka manager
15.         try:
16.             self.kafkaManager = KafkaManager(self.configuration)
17.         except Exception as e:
18.             logging.getLogger('application').error(e)
19.
20.     def doTaskImpl(self):
21.         try:
22.             while self.status is Task.Status.RUNNING:
23.                 message = QueueManager.getItemFromQueue('kafka')
24.                 logging.getLogger('application').debug('Received message : {0}'.format(
    message))
25.                 self.kafkaManager.sendMessage(key=Task.client_id, msg=message['content'
    ], topic=message['topic'])
26.         except Exception as e:
27.             logging.getLogger('application').error(e)
28.
29.
30.     def shutdownImpl(self):
31.         pass
```