**Experimento 2**

Grupo 6

Camilo Montenegro

Tomas Venegas

Juan Diego González

Juan Manuel Lovera

Carlos Peñaloza

Colmines

**Problemática**

Se quiere implementar un sistema con 2500 microcontroladores, cada uno con 4 sensores que envían información a intervalos con el fin de identificar condiciones de alerta en una mina. En este caso se tienen que asegurar las distintas partes relacionadas con la solución del problema para que la información, infraestructura o servicios no se vean afectados.

**Objetivo del experimento**

Se desea implementar un mecanismo de seguridad integrado, el cual evite cualquier tipo de ataque que pueda afectar el funcionamiento de la mina. Se debe asegurar las entidades físicas (autenticación y autorización para poder escribir en los tópicos del servidor de mensajería que se está utilizando), también el aseguramiento de las entidades virtuales que se tienen ( bridges, kafka, zookeeper, etc…) y por último los actores humanos, quienes son los que se deben autenticar y autorizar para que tengan acceso a ciertos servicios.

**Descripción del experimento**

En este experimento se probó la implementación de tácticas de seguridad en los distintos niveles manejados en la aplicación.

**Entidad Física:**

El envío de datos de bajo nivel desde los microcontroladores, pasando por el middleware y luego enviados al servidor de mensajería utilizando tópicos, maneja el protocolo de seguridad SSL.

Inicialmente, esta implementación se realiza utilizando SSL y haciendo publicaciones por tópico en CloudMQTT para que luego estos sean consumidos por un Bridge.

**Entidad Virtual:**

Al momento que los datos son recibidos desde el servidor de mensajería MQTT por el Bridge, esté se encarga de descifrar el mensaje para que pueda ser publicado en otro tópico en Kafka. Dicha comunicación entre el Bridge y Kafka+Zookeeper se realiza de igual forma con SSL utilizando un cifrado asimétrico y certificados digitales en ambos nodos de comunicación. De esta forma, se está asegurado tanto la lectura como escritura de los datos en Kafka+Zookeeper.

**Actores Humanos:**

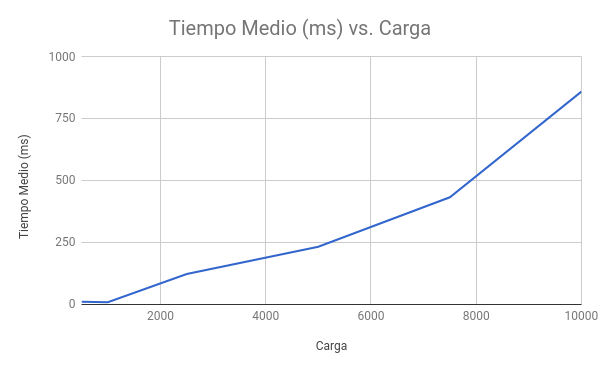
Para el experimento, se implementó un sistema de autenticación por hashing. Existen diferentes grupos de usuarios y cada grupo cuenta con permisos diferentes en cuanto al uso de la aplicación. Las credenciales de los usuarios se almacenan en una base de datos luego de que se les realiza un hashing con el algoritmo SHA-256.

**Pruebas de Carga y Comparación**

En esta ocasión se hicieron pruebas de carga sobre los servicios de Post de la aplicación. Esto se debe a que no cualquiera puede hacer post sobre la base de datos. Se realizaron pruebas sobre el

servicio que inserta las mediciones de cada uno de los sensores que se tienen en la mina. En un principio se hizo la prueba sin tener algún tipo de autorización o autenticación, es decir que no se tenía seguridad. Los siguientes son los resultados obtenidos para esta operación:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Post Sin Seguridad** | | | | |
| **Carga** | **Tiempo Medio (ms)** | **Mínimo (ms)** | **Maximo (ms)** | **Error** |
| 500 | 11 | 3 | 121 | 0% |
| 1000 | 9 | 3 | 196 | 0% |
| 2500 | 123 | 3 | 1207 | 0% |
| 5000 | 233 | 3 | 2042 | 0% |
| 7500 | 433 | 3 | 3241 | 0% |
| 10000 | 860 | 3 | 4096 | 0% |



Podemos ver que el servidor aguanta aproximadamente más de 1300 peticiones por segundo con un tiempo de respuesta de promedio menor a 1000 milisegundos. Esto satisface desempeño que se esperaba para la aplicación sin seguridad.

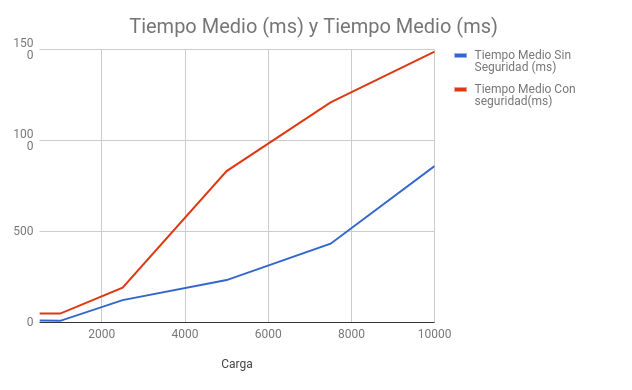
Al implementar la autenticación y autorización por medio de Auth0, se redujo este tiempo de respuesta lo cual afecta el rendimiento de la aplicación. Este es uno de los precios que se debe pagar para poder tener una aplicación segura por lo cual toca tomar decisiones con respecto a la verdadera implementación de este artefacto. Las siguientes son los datos recolectados con las pruebas realizadas:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Post Con Seguridad** | | | | |
| **Carga** | **Tiempo Medio (ms)** | **Mínimo (ms)** | **Maximo (ms)** | **Error** |
| 500 | 50 | 47 | 156 | 0% |
| 1000 | 49 | 47 | 175 | 0% |
| 2500 | 192 | 48 | 403 | 0% |
| 5000 | 832 | 138 | 2139 | 0% |
| 7500 | 1209 | 75 | 3303 | 0% |
| 10000 | 1488 | 74 | 5834 | 13% |



Se puede ver claramente que el tiempo medio de respuesta aumentó considerablemente, como también el porcentaje de error en el caso de los 10000 posts por segundo. En el caso de la mina, se espera tener 5000 posts por segundo, pero debe escalar al doble. En este último caso no se cumple la meta totalmente. Por ello se deben estudiar distintas alternativas para manejar la seguridad en este aspecto.

A continuación se puede ver un gráfico en donde se ve claramente la diferencia entre los dos modelos (con seguridad y sin seguridad).



**Decisiones de arquitectura**

Utilizamos tácticas de autorización y autenticación en todos los intercambios de mensajes. Y adicionamos tácticas de verificación y encriptación a los mensajes que tenían conexión con entidades externas a los servicios centrales.

La base de datos que contiene la información de autenticación está aislada de la base datos con la información del negocio. Esto lo hicimos con la intención de aumentar la seguridad, separar los objetivos y afectar el desempeño de la aplicación lo menos posible.

**Artefactos construidos**

Debido a que la aplicación se basa en un patrón de diseño Model View Template implementado con Django, fue necesario definir los modelos y los views asociados a las entidades involucradas en el funcionamiento de la aplicación, en este caso, fue posible definir dos artefactos generados (Modelos y

views), estos dos artefactos fueron definidos en torno al funcionamiento de la aplicación, es decir, de acuerdo a los diferentes tipos de sensores que existen en cada uno de los microcontroladores.

Se implementaron servicios de autenticación y autorización en todos los niveles de la aplicación y además de esto se implementaron tácticas de encriptación y certificados.

**Análisis**

Los resultados indican que el servidor fue capaz de soportar la demanda de peticiones

máxima de alrededor de 7000 solicitudes por minuto, con un porcentaje de error mínimo. Esto significa que los patrones e implementación del diseño no fueron suficientes para cumplir totalmente con los requisitos del sistema bajo condiciones normales y tener un excelente desempeño bajo condiciones de estrés.

La pruebas de unidad fueron exitosas.

Las pruebas sobre el servidor de mensajería nos muestran dos cosas: la primera es que el servidor de mensajería es apto para recibir todas las peticiones requeridas y cumple con los atributos de calidad. La segunda es que el script publicador que hace las publicaciones en el servidor de Django no es lo suficientemente rápido.

**Conclusiones**

Vemos que la arquitectura implementada no satisface los requerimientos no funcionales.

Para mejorar el tiempo de respuesta de las peticiones POST proponemos utilizar una base de datos no relacional con una arquitectura hadoop.

Para solucionar el problema del publicador proponemos implementar en una segunda instancia un mecanismo multi-thread para el publicador,ya que la división en tópicos no parece ser suficiente para atacar este problema.