**Nombre(s):** JULIAN ANDRES BERMUDEZ VALDERRAMA (201519648)

**Ingeniería de Sistemas y Computación**

**Pregrado: Arquitectura y Diseño de Software**

**Web:** [**https://profesores.virtual.uniandes.edu.co/~isis2503/dokuwiki/doku.php**](https://profesores.virtual.uniandes.edu.co/~isis2503/dokuwiki/doku.php)

**Semestre: 2017-20**

ANGEL CAMILO CABEZAS VARELA (200911288)

NICOLAS DAVID MUÑOZ CUERVO (201422403)

SERGIO ANDRES PARDO SANCHEZ (201415928)

DIEGO FRANCISCO SANABRIA DIMATE (201217484)

# Reporte entrega final Experimento 2



A continuación, se describe el proceso de pruebas y análisis sobre el requerimiento de calidad de seguridad.

El aseguramiento sobre el protocolo de mensajería MQTT, se realizó en primera instancia por las entidades físicas que tienen acceso a la escritura sobre la cola de mensajes, en éste caso se implementó autenticación y autorización de aquellos usuarios simulados por medio de Node-red. Un inconveniente presentado en el despliegue de los módulos de telemetría, mensajería y seguridad en el protocolo de mensajería en un S.O. distinto (Linux) en comparación con la infraestructura usada anteriormente (máquinas virtuales, Windows 7) fue la interoperabilidad entre el destino de la información por medio de MQTT (Linux) y su destino (máquinas virtuales). Por otra parte, el procesamiento en la escritura en la cola de mensajes de los tópicos requería un mayor procesamiento por el hecho de que la entidad física debe autenticarse y ser autorizada para realizar dichas operaciones.

El aseguramiento de la aplicación de persistencia, se realizó por medio del servicio de autenticación Auth0, que provee servicio de seguridad para la API de persistencia.

Los resultados de las pruebas experimentales son los siguientes:

* Prueba con 100 Threads



* Prueba con 200 Threads



* Prueba con 400 Threads



* Prueba con 800 Threads



* Prueba con 1600 Threads



* Prueba con 2500 Threads



* Prueba con 5000 Threads



De acuerdo a los resultados, se puede concluir que introducir seguridad en distintos niveles de la arquitectura genera mayor procesamiento y una disminución de desempeño en el comportamiento del sistema como un todo.

No obstante, éste problema se puede minimizar con pequeños cambios en cada uno de los niveles de la arquitectura. Ejemplo claro, es el uso de Redis como base de datos donde se persiste la información referente a los microcontroladores para posteriormente verificar su autenticación; al proveer un sistema de liviano procesamiento, se reduce costo de procesamiento. Otro ejemplo que podría introducirse, sería el uso de MQTT directamente sin uso de Bridge, ya que, si bien funciona como un punto de comunicación y almacenamiento de mensajes desde MQTT hasta el componente de procesamiento y persistencia, genera que el protocolo de comunicación en sí sea más pesado.

Como idea general, cabe mencionar que la garantía de seguridad es un factor que debe ser analizado y confrontado con los requerimientos del negocio o de los stakeholders. En éste punto es donde tanto arquitecto como stakeholders definen un margen de aceptabilidad de cumplimento de aquellos requerimientos de calidad que causan conflicto entre ellos.

Analizando este incumplimiento de los requerimientos encontramos a que se puede atribuir a la inclusión de seguridad en cada uno de los niveles de la arquitectura. Proponemos implementar tecnologías más livianas en el costo de procesamiento de seguridad para no afectar de manera drástica el desempeño del sistema.

A continuación, se evidencia la gráfica modelada a partir de los resultados experimentales:

|  |  |  |  |
| --- | --- | --- | --- |
| Creación de medidas (Seguridad/Persistencia) | |  |  |
| Fila | # Muestras (# Threads) | Media (ms) | % Error |
| 1 | 100 | 995 | 0.00% |
| 2 | 200 | 1204 | 0.00% |
| 3 | 400 | 2022 | 0.00% |
| 4 | 800 | 2295 | 0.00% |
| 5 | 1600 | 5507 | 0.00% |
| 6 | 2500 | 5985 | 14.80% |
| 7 | 5000 | 10097 | 13.72% |

**Patrón de diseño utilizado**

Para el manejo de usuarios y sus respectivos accesos y autenticaciones al interior del experimento se utilizó el Patrón *Intercepting filter* este patrón favorece la seguridad al implementar filtros que cruzan la información, procesándola de una manera estandarizada y así mismo designando los accesos correspondientes a los usuarios correctos.

Al ingresarse una solicitud de usuario de acceso a la información, y en específico a los métodos REST, esta se filtra y se definen las especificaciones a las cuales el usuario en cuestión tiene acceso, limitándolas y generando una nueva capa de seguridad.