# EL DIARIO DEL TFG

## Anteproyecto

Bien, lo primero a decir es que cuando se me propuso la idea inicial no me llamaba mucho la atención principalmente por las expectativas que tenía sobre el TFG y el enfoque que había ido modelando durante años. Todo esto sumado a que no tenía ni puta idea del tema y no sabía como enfocarlo me desmotivaba bastante. Pero esto no deja de ser una asignatura más, así que tiempo, esfuerzo y me guste más o menos toca hacer las cosas bien por lo que dando mi mejor cara con los tutores empecé a hablar con ellos y a informarme.

Al principio no yo sabía buscar muy bien información ni tenía muy claro el enfoque, solo contaba con papers largos en inglés que hablaban de temas que no conocía. Pero poco a poco iba haciendo preguntas, informándome más y montando una idea preliminar de lo que había que hacer, pero siempre con la sensación de que por mucho que podía entender qué pretendíamos crear, con qué objetivos y demás, yo no sabía cómo materializarlo.

A todo esto, había que hacer el anteproyecto, y aunque realmente tardé en ponerme debido a mi desmotivación y a que tampoco tenía muy claro qué debía poner y qué no, al final, apurando las fechas casi a lo máximo, hice una primera versión que pareció ser bastante top según mis tutores, y tras unas breves correcciones se entregó todo a tiempo en el segundo plazo de entrega de anteproyectos.

Después de esto admito que estuve una temporada tocándome los cojones, primera tarea hecha, más o menos bien, tocaba lo gordo y no sabía cómo enfocarlo, además que faltaban cosas por definir en el proyecto, así que hasta que no llegó el paper de “Preliminary Document TFG” no tenía ganas de ponerme.

Pero tocaba mini hostia de realidad, EMPEZABA EL DESARROLLO, así que a ponerse las pilas de nuevo.

## Desarrollo del TFG

El paper era escasillo y no se explicaba demasiado bien. Sumando esto a que hacía referencia a otros de los que había visto y que no me acordaba de nada de ellos, supuso necesitar tiempo, pausas y leerlo bastantes veces para plantearme posibles preguntas que me sirvieran para obtener información de cómo empezar esta mierda.

Las preguntas surgieron, las apunté en el repo de git, pero entre que yo no me explico cuando no controlo del tema (y es el caso) y que los tutores son un poco meh, pues llegó la tutoría, pregunté, entendí algo más algunas cosas, pero muchas otras siguen siendo un misterio. Pero bueno, son problemas del futuro, el caso es que hay que empezar YA a programar y redactar y para eso está este documento y las primeras pruebas que voy haciendo en base al framework que me pasó Davide y la información que estoy buscando.

Lo primero que destaco es que hay muy poca información sobre brokers y sobre MQTT, porque o te aprovechas de los servicios ya construidos o si quieres buscarte la vida o indagar en cómo se construyeron es o demasiado difícil o no hay nada. Sigo sin tener del todo claro todo lo que debo hacer, pero al menos entendí mejor el tema del gestor de anomalías y la comunicación con nuestro gestor de confianza y por ahora lo que creo que debo hacer es conseguir una comunicación bidireccional entre estos módulos y en base a la anomalía del dispositivo/sujeto X recalcular para este la confianza de alguna forma y reenviarla a anomalías.

Para hacer esto estoy probando a desglosar tareas, ver cómo puedo mandar datos con java entre distintos puertos y cómo funciona el protocolo MQTT, con sus librerías. Me he hecho un esquema a mano que quiero pasar a ordenador, para tenerlo más claro y poder apuntar las preguntas de forma visible y así también compartirlas con los tutores sobre algo que puedan ver, quizás así me explique mejor, pero antes de eso tengo que llevar algo, no puedo ir sin nada, así que ya sea desarrollo de la memoria o pruebas de implementación toca empezar de verdad.

He encontrado Paho, pero la verdad que la documentación es escasa y pese a que tenía una idea inicial de cómo comunicarme con MQTT haciendo algo similar a los sockets, no parece funcionar porque todo lo que he visto online para MQTT requiere de un bróker externo y la gracia es que realmente nosotros creo que no queremos usarlo porque precisamente queremos ser ese bróker, así que veremos cómo me apaño. Por ahora sigo buscando información sobre esta librería, indagando si hay más o probando de mientras con sockets a mandar, recibir información e ir modelando un poco qué datos requerimos, qué clases, qué funciones…

Mi idea primero es conseguir comunicar bidireccionalmente el módulo de anomalías con el de confianza, estudiar qué datos (variables) definir en cada uno y qué funciones hacen falta. Después seguramente debería definir los objetos/dispositivos/cosas para poder comprobar si el ID que envía anomalías existe entre lo que tenemos y entrar entonces o no en el cálculo de confianza.

Esto creo que será lo más difícil porque no tengo ni puta idea de cómo calcular esa confianza, de qué valores depende, si hace falta una reputación almacenada en una base de datos que haga las veces de reputación inicial… además si hiciera falta eso la reputación debería actualizarse cuando se calcule la confianza, pero cómo?? Qué valor se pondría, la nueva confianza calculada o en base a esta con alguna función se calcula la reputación??

Hola de nuevo, continúo contando. Primero debo decir que estoy bastante desmotivado con la vida ahora mismo así que no tengo ganas de nada, lo cual repercute de una manera u otra en el TFG. Pero soy consciente de que necesito al menos una tutoría o dos antes de Semana Santa para preguntar las dudas del Excalidraw y las que me vayan surgiendo y para presentar al menos un mínimo de lo que estoy haciendo. Por mucho que me joda, toca continuar.

Continuo mi narración y debo decir que el protocolo MQTT es un asco, necesito ayuda, porque si queremos hacer las veces de bróker para no utilizar uno no sé cómo cojones vamos a comunicarnos, además que siendo un protocolo de pub/sub qué gracia tiene entre anomalías y confianza?? Se quieren enviar datos, no seguir un sistema así. Obviando esa duda y para comenzar de alguna forma estoy jugando un poco con Sockets, comunicando un servidor que hace las veces del módulo de confianza con un cliente que sería anomalías. El cliente me envía un ID y una anomalía asociada a ese ID y en el servidor se verifica que el ID existe entre los objetos que se crean previamente de forma aleatoria y por ahora he llegado a poco más. Me surge ahí otra duda y es acerca de la anomalía 🡪 cómo va a venir ese dato? Y cómo lo utilizamos, me explico: viene un ID y sobre ese una anomalía, pero esta debe estar registrada o tener un valor o algo con lo que recomputar la reputación y confianza en base a lo conocido sobre ella o no. No es la mejor explicación que he dado, pero creo que me entiendo.

Dicho esto, también tengo mis dudas sobre los objetos a crear, tanto porque no sé qué variables y funciones deberían tener para que sean útiles para calcular la confianza, tampoco sé si HUMANOS cuenta como objeto a crear y tampoco sé si entre estos objetos debe haber comunicación, que en teoría sí, también con el MQTT, ¿¿pero con qué fin?? ¿¿¿De qué me sirve que se comuniquen entre sí y con el Broker???

Voy a hacer una pequeña lista de tareas que me gustaría completar antes de la tutoría que quiero pedir antes de las vacaciones y quizás también incluya el recopilatorio de preguntas para tenerlo todo centralizado.

|  |  |
| --- | --- |
| TAREA | DESCRIPCIÓN |
| Ordenar el código | Quiero que la estructura quede bien separada entre el main, las distinas funciones y variables, y los propios archivos bien ordenador en paquetes. |
| Investigar java + BD | Quizás sea conveniente usar una BD de cara a almacenar la reputación inicial de cada objeto o algo así, aunque sino se podría simular con una variable que se genere aleatoriamente. |
| Investigar cálculo de confianza | Cómo calculo la confianza, qué valores necesito, qué tiene que ver la reputación y como computo si es necesario la actualización de reputación. |
|  |  |
|  |  |

*19-3-24*

He estado revisando la clase dispositivos y nombrando las variables de forma que queden más claras. Me surge la duda de si Dispositivos Físicos y Virtuales deben ser subtipos de Dispositivo y tener las mismas propiedades o alguna diferente partiendo de lo mismo, o deben ser completamente distintas clases.

Fuera de esto lo que más me preocupa ahora que lo pienso es cómo cojones computar la confianza. Necesito que se me pase un ID y una anomalía, rescatar el dispositivo/objeto con ese ID, verificar cómo de grabe es la anomalía porque quizás ya se conocía y se corrigió o es nueva o se conocía, pero es más o menos grave… Esto junto con la reputación previa del ID si existe o si el dispositivo es nuevo sin este campo o algo así computar un nivel de confianza (cómo sé si es nuevo o cómo consigo que haya dispositivos nuevos sin reputación inicial y otros con reputación inicial que ya sean más antiguos).

*21-03-24*

Ayer miré algo sobre Docker MQTT, y al final no deja de ser un Docker compose y juegos de comandos, pero sigue necesitando de Mosquitto.

Lista preguntas tutoría:

* **Requisitos 🡪 cálculo de confianza en base a lo que llegue de anomalía sobre el objeto que nos diga y teniendo en cuenta qué anomalía sucede para recomputar la confianza | comunicación MQTT confiable y segura entre objetos (esto de qué nos sirve)**
  + Respuesta 🡪 las anomalías pueden ser vulns, que se comuniquen con quien no puede y demás. El entorno del dispositivo.
  + Caso de uso: alguien que quiere entrar a su casa, teléfono, y cuando entra se abran las ventanas y se enciendan las luces.
* **Humano es objeto?? 🡪 Excalidraw**
  + Respuesta 🡪 el humano es objeto, pero las anomalías son intencionadas.
  + Mejor centrarse por ahora en dispositivos 🡪 heredan ambos de cosa.
* **pub/sub qué gracia tiene entre anomalías y confianza?? Se quieren enviar datos, no seguir un sistema así**
  + Respuesta 🡪 por ahora sockets y luego se podría extrapolar
* **El ID que se envía desde anomalía debe verificarse que exista entre los objetos/dispositivos existentes??**
  + Respuesta 🡪 sí
* **Me surge ahí otra duda y es acerca de la anomalía 🡪 cómo va a venir ese dato? Y cómo lo utilizamos, me explico: viene un ID y sobre ese una anomalía, pero esta debe estar registrada o tener un valor o algo con lo que recomputar la reputación y confianza en base a lo conocido sobre ella o no**
  + Respuesta 🡪 se pueden enviar más datos aparte. GRADO DE ANOMALÍA.
  + Si la anomalía no existe hay que añadirla a una base de datos con las anomalías contempladas y el grado de gravedad. Anomalías asociadas a un escenario concreto.
* **cómo calcular esa confianza, de qué valores depende 🡪 variables de los objetos?? Cuáles serían necesarias??**

**si hace falta una reputación almacenada en una base de datos que haga las veces de reputación inicial o simulada con un número de ramdom que exista o no en base a si el dispositivo/objeto tiene o no una reputación inicial… además si hiciera falta eso la reputación debería actualizarse cuando se calcule la confianza, pero cómo?? Qué valor se pondría, la nueva confianza calculada o en base a esta con alguna función se calcula la reputación??**

* + Respuesta 🡪 reputación necesaria, se actualiza con la confianza.
* **Paho**
  + Respuesta **🡪**
* **Docker MQTT**
  + Respuesta 🡪

TENGO QUE CENTRARME EN UN ESCENARIO CONCRETO. PARA CREAR EL MODELO DE CONFIANZA Y TODO LO DEMÁS.

*31-3-24*

Yepaaaaaaaaaaa, he vuelto. En semana santa no he tocado absolutamente nada, sigo bastante desmotivado con todo, pero bueno, hay que darle, la vida sigue.

Primero y principal, hay que avanzar con el código y quizás ir empezando con el chasis de la memoria, pero lo principal es el código. Dicho esto, vamos a hacer un breve resumen/recapitulación de los puntos a tener en cuenta hasta el momento para empezar a hacer algo.

Recapitulación desde la última tutoría. Cosas que hacer o avanzar YA DE YA:

1. Establecer un caso de uso en el que centrarme. (quizás como el del paper o algo así).
2. Es necesario crear una base de datos con diferentes tablas 🡪 para la reputación, para las incidencias/amenazas…
3. Hay que crear una clase COSA y de ahí dos hijos que sean los tipos de dispositivos.
4. Basarme en los papers porque el sistema será muy similar.

### Caso de uso.

Tengo que definir si o si el caso de uso en el que centrarme antes de nada para así definir la metodología con la que calcular la confianza, las variables de los diferentes agentes/objetos involucrados, las tablas de BD necesarias y las columnas de estas…

Red conformada por una bombilla, un móvil y una televisión. El móvil puede comunicarse con ambos dispositivos, la tele también, pero la bombilla solo con el móvil. (Se podría decir que el móvil y la tele están en la red externa, la bombilla en la interna??)  
  
Hay que conseguir esa comunicación segura y gestionar las incidencias que nos mande el gestor de anomalías, ya sean vulns, intentos de comunicaciones que no deberían darse…  
  
Hace falta una base de datos con información sobre la reputación inicial de estos dispositivos y las amenazas conocidas y su gravedad.

Pongamos ese caso de uso, aunque realmente tampoco sé si está bien definido del todo, pero bueno, por probar… Dicho esto qué toca ahora? Qué variables hacen falta en cada dispositivo:

|  |  |
| --- | --- |
| Dispositivo | |
| Virtual | Físico |
| Variables:   * Tipo (indique si un dispositivo es físico o virtual) * ID * Fecha Última supervisión * number of past vulnerabilities * Certificación de seguridad sí o no * Reputación inicial | Variables:   * Tipo (indique si un dispositivo es físico o virtual) * ID * Fecha Última supervisión * number of past vulnerabilities * Certificación de seguridad sí o no * Reputación inicial |

*2-4-24*

Ha costado bastante, pero al final he conseguido que funcione el putísimo JDBC en VSCode. He probado en todos los IDEs, pero finalmente haciendo uso de XAMPP y con el controlador de MySQL he conseguido incluirlo en el proyecto java y que funcione la conexión a la base de datos.

Ahora toca crear las bases de datos e incluir esto en el código principal. La primera va a ser la de reputación que por ahora va a tener las siguientes columnas:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Reputación** | | | | | | |
| ID dispositivo | Nombre dispositivo | Tipo dispositivo | Reputación | Last\_update | Estado dispositivo | Comentarios/Descripción |

Creando la base de datos de reputación me ha surgido una duda 🡪 Sería más conveniente crear una base de datos de la reputación que enlace el ID\_Dispositivo con su reputación y algunos datos más, o una BD para los dispositivos y todos sus atributos que además incluya una columna reputación con un valor si el dispositivo tiene alguna antigüedad en la red o null si no. Respecto a esto creo que he llegado a la conclusión de que la tabla de reputación no va a ser como se explicaba, va a ser la que contenga los datos de los dispositivos con los que hacer el cálculo de confianza y una columna reputación que puede ser nullable

Creo que también sería óptimo ir haciendo un diagrama de flujo del MAIN para saber qué está ocurriendo, en qué orden y demás.

***Un caso: desde anomalías llega el ID y el grado de anomalía de una anomalía que consiste en que un dispositivo se quiere comunicar con otro 🡪 en este caso se debe evaluar la confianza de este dispositivo y determinar si pueden o no comunicarse, en caso de que sí, que se establezca la comunicación y en caso de que no, se bloque. Esto sería un caso de uso de la comunicación entre dispositivos junto con la comunicación confianza-anomalía.***

*5-4-24*

Tengo hasta el momento generación aleatoria de dispositivos, conexión con la base de datos y poder guardar estos dispositivos. Ahora bien, necesito conseguir relacionar anomalías y lo que me manda con lo que tengo de los dispositivos y reputación porque ahora mismo no le veo la relación. Además, tengo que desarrollar un modelo de cálculo de confianza aunque por ahora tenemos la idea de los pesos para cada parámetro del dispositivo por relevancia y la suma y un threshold.

Mi idea por el momento creo que va a ser la siguiente: desde amenazas nos van a llegar los datos para la gestión de riesgo, los cuales van a ser los del paper Segragated para riesgo, con sus niveles y sus mierdas. Con esto vamos a hacer una primera estimación del riesgo y en base a ese nivel se quedará el dispositivo en prohibido o su estado se tendrá que seguir evaluando si el riesgo no es demasiado alto, calculando la confianza con sus parámetros, reputación… Este estado debe quedar reflejado en la base de datos para saber si el dispositivo queda o no en “cuarentena”.

En cuanto deje la idea terminada y funcional explico el código función a función, clase a clase.

*7-4-24*

Vale, voy a hacer un poco diagrama de flujo de los mains, del cliente y del servidor y así vemos que está pasando, explicamos sus partes y qué consigo, dudas, etc.

Dudas:

* ID\_existente 🡪 Sería necesario obtener los datos de la BD antes de comparar el thingID que nos envía anomalía con el de los distintos dispositivos creados, o podemos aprovecharnos de la lista de dispositivos creada (operatorThing) ????
* Nivel\_riesgo 🡪 se evalúa el riesgo multiplicando los factores de detectabilidad, criticidad y probabilidad que nos envía anomalías sobre la anomalía asociada al dispositivo, es la forma correcta de hacerlo??? En caso de que sí, habría que actualizar el estado\_actual de los dispositivos en la BD pero al ser un método externo no tengo las variables de la conexión ni nada para actualizar esta data.
* Nivel\_confianza 🡪 la forma que he decidido por ahora es asignando un peso a las variables relevantes de un dispositivo que varía entre el 0-1 y multiplicando este por el valor del atributo. Después se suma todo y si pasa un umbral determinado pues es confiables y sino pues queda en cuarentena a la espera de algo, aunque no sabría decir de qué.
* Base de datos anomalías 🡪 hace falta?? Porque en mi planteamiento no. Si hace falta para qué?
* Caso de uso 🡪 no tengo muy claro cuál, definirlo mejor por favor y gracias.

Reputación inicial 🡪 o se cambia de nombre o hace falta una reputación actual.

Historial reputación para las estadísticas

Mqtt básico entre dispositivos e intentar incorporarlo a la comunicación hasta ahora

*21-04-2024*

Hola de nuevooo, cuánto tiempo… He vuelto y toca hacer cositas, mi idea de hoy es desarrollar texto y continuar con lo dicho en la última reunión. Para empezar, cuántos apartados va a tener mi TFG?

* Portadas
* Abstract and key words
* Resumen y palabras clave
* Índice
* Introducción
  + Motivación
  + Objetivos
  + Metodología
  + Estructura de la memoria
* Estudio del arte / Tecnologías a utilizar
* …
* Conclusiones y líneas futuras
* Manual de usuario
* Apéndice

<https://www.youtube.com/watch?v=r7wKPz3KObA> 🡪 usar esto y el mensaje que se publica que sea lo que se genera de parte de anomalía o lo de confianza desde el trust\_manager.

Lo que yo me he imaginado es lo siguiente: necesitamos si o si un bróker para mqtt (mosquitto, hiveqm…). PARA LA COMUNICACIÓN ENTRE TRUST Y ANOMALÍAS 🡪 el programa Java devuelve un String que es el que se manda como mensaje con este protocolo como en el vídeo.

*25-04-24*

Hoy hay tutoría y voy FATAL. Voy a intentar hacer u compendio de preguntas y hay que ponerse las putísimas pilas.

1. Mosquitto o MQTTCli 🡪 es posible ejecutar el programa de Java tanto por parte de anomalías como por parte del trust\_manager y que la salida de texto se trata en un oneliner de bash para poder usarla con los comandos de consola de estas herramientas?? Si no, cómo usamos el MQTT 🡪 PAHO?? Pero no sé cómo funciona, la mayoría de códigos que he encontrado no consigo que vayan bien o no sé cómo hacerlo
2. El documento de overleaf 🡪 lo voy a hacer en LaTeX porque queda más chulo, tengo las portadas y el resumen/abstract (las keywords no sé cómo de bien estarán). Qué apartados debería poner???
3. Reputación, base de datos de histórico 🡪 hablamos de que hacía falta una comparativa con el dato de reputación inicial en la BD de los dispositivos. Además una base de datos histórica de las reputaciones de los distintos dispositivos vinculadas a sus IDs o algo así para ver la evolución a nivel estadístico.

A nivel teórico queremos saber el histórico de reputación para estudios sobre vulnerabilidades puntuales o cosillas así. Pero no sirve para el ahora, es solo teórico, pero debe estar.

1. Securizar el protocolo MQTT 🡪 login de usuarios o algo así, credenciales… he leído opciones y sería interesante quizás mirarlo.
2. GUI para el MQTT??

Cifrar el string java de los programas antes del pub/sub.

*28-04-24*

Hoy he estado escribiendo los primeros apartados en LaTeX. Debo decir que requiero mucho de ChatGPT porque no sé cómo enfocarlos, pero luego siempre estoy intentando darle mi toque, retocarlos, cambiar la estructura o la forma de expresión, etc. Al final lo que quiero es una base para poder darle yo mi arte.

Dicho esto, se me ocurren algunas dudas:

* ¿De cara a la programación voy a tener que simular distintos casos? Me explico. Un posible caso sería que los dispositivos no tengan reputación y todos sean nuevos, otro que si tengan ya algunos datos en el histórico de reputación, otro que entre un dispositivo nuevo y sea confiable, otro caso sería que entre y no lo sea, etc. ¿¿Cuántos casos y cuáles habría que representar??
* La tabla del histórico de reputación podría ser: Columna 1 – ID+TimeStamp (primary key), Columna 2 – Id, Columna 3 – TimeStamp, Columna 4 – Valor reputación. Así a la hora de querer buscar sobre un dispositivo concreto estarían ordenados y la fecha determinaría también los datos más antiguos y recientes de cada dispositivo.
* El dato al crear los dispositivos simulados en el Main de la reputación actual debe recuperarse del histórico, haya o no más datos, para que en el caso de que sí pille el último valor.
* ¿Quizás convendría crear una columna en la tabla histórico reputación que sea el valor actual?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID+timestamp | id | timestamp | Valor\_reputación | Reputación\_actual |

* En la estructura del documento debería poner el estado del arte como si fuera el punto de estudio previo o dentro de introducción, o no ponerlo e incluir la información de MQTT, Trust y demás en otros puntos.

*1-05-24*

Hoy me he dado cuenta de cosas. Primero de todo, escribir es demencialmente difícil, no sé qué seleccionar, qué secciones poner o no, en qué orden, cuánta información, qué información corresponde a qué sección…

Por otro lado, tema código. Voy a relatar el funcionamiento actual para marcar así qué he aprendido.

Primero de todo creamos los dispositivos para la simulación, en este caso pongamos que hay uno solo para simplificar. Después se verifica si la tabla de dispositivos está o no en la BD para meter la info del dispositivo, si no está se crea. Luego se sigue un proceso similar con la de reputación, se debe verificar si existe o no la tabla y crearla en consecuencia, pero a la hora de meter los datos hay que recuperarlos de la otra tabla.

Lo primero que hay que tener en cuenta es que de primeras las tablas están vacías, así que debe haber un proceso de PRIMERA CARGA que va a ser distinto a las actualizaciones que sufran luego. En la primera carga la tabla de dispositivos va a obtener los datos del dispositivo y como acaba de entrar ese dispositivo su estado debe estar pendiente de evaluación. En base a estos datos se va a recuperar el ID, la reputación inicial y se genera un timestamp para introducir todo esto en la tabla de reputación. En esta primera carga la reputación actual y la que se guarda sobre el dispositivo es la misma (me acabo de dar cuenta que la columna de reputación actual no tiene mucho sentido porque si cada vez que se actualice la tabla se va a añadir una nueva fila siempre se añadirá la última versión y la actual será esa misma, a no ser que se actualice la reputación actual en toda la tabla, cosa que tampoco tendría mucha lógica). De ahí en adelante hay que determinar cómo va a cambiar la reputación del dispositivo según la confianza calculada, si el valor va a ser double, int, del 0 la 1, del 0 al 100…

Una vez hecho todo esto digamos que la primera carga estaría más o menos lista. Ahora bien, cada vez que entre un dispositivo nuevo se debe seguir un proceso similar, pero si el dispositivo ya existía y tenemos datos (cosa que hay que verificar de alguna forma) simplemente habrá que ir actualizando su información en la tabla (estado y demás movidas) y añadiendo nuevas filas en la de reputación que serán más actuales porque el timestamp se genera en ese momento y eso sí, habrá que actualizar la tabla dispositivo con la última actualización de reputación que se tenga que quizás se deba sacar de la tabla o se pueda obtener por otro lado porque antes se deberá calcular.

Además de esto hay que averiguar alguna forma, método o algo así para el cálculo de confianza después del nivel de riesgo, y quizás el nivel de riesgo se pudiera complicar un poco más por darle chicha.

También hay que obviar los sockets y contar que todo va a ir por parámetros del main ya que se va a jugar con bash para pasar las cosas por onliners y mqtt. Los dispositivos existentes podrían quizás tener un usuario y contraseña o alguna forma de securizarse, las comunicaciones pueden cifrarse... todo esto es por bash, hay que darle duro.

5-5-24

Ante mi duda existencial del día me encuentro sin hacer nada porque sé que tengo que tocar código, pero no sé cómo hacerlo.

La duda es: yo simulo unos dispositivos con unas características y los meto en una base de datos que almacena esta información. Esta tabla se crea y se rellena con la info de los dispositivos que conforman el sistema, pero de cara a incorporar uno nuevo? Y de cara a tratar luego los datos para el cálculo de confianza, si llega un ID que no existe no significa que se quiera meter. La cosa es, queremos poder meter dispositivos, o simplemente se simulan unos, se almacenan y sobre eso se juega y nada más??? Porque si queremos que se puedan incorporar hay que hacer algún método de verificación de que no exista ya o comprobar su reputación previa o algo así. 🡪 definimos unos contextos concretos

Otra duda que tengo es tema topics, defino los que quiera, que anomalía mande uno, ¿¿que queden definidos como finals??

También el tema de por qué se repite el mensaje por consola una vez recibido y de cómo puedo conseguir simular los dispositivos en sí para que cuando desde el trust\_manager se manda que se acepta o no al dispositivo o algo así ellos puedan recibir y hacer como un ACK.

La primera carga y demás actualizaciones en las bbdd

*25-5-24*

Después de mucho tiempo he vuelto.

Toca centrarse y darle caña y para eso creo que lo más importante es definir unos casos de uso para programar y escribir en base a eso.

Qué escenario queremos contemplar, qué situaciones dentro de este, y en base a eso acercarnos lo más posible.

#### Smart Home

En nuestro escenario van a interactuar dispositivos, que tienen una serie de atributos que nos van a permitir hacer los cálculos y operaciones pertinentes y determinar en mejor medida qué hacer con cada uno y cómo actuar.

Podemos englobarlos en las siguientes categorías:

|  |  |  |
| --- | --- | --- |
| DISPOSITIVOS | | |
| Dispositivo físico | Id | Número autogenerado que es el identificador del dispositivo |
| Fecha última revisión | Fecha en formato Date de la última revisión de seguridad que tuvo el dispositivo |
| Número de vulns pasadas | Cuántas vulns ha tenido o le han sido detectadas |
| Certificado de seguridad | ¿Tiene o no certificado de seguridad? |
| Tipo de dispositivo | Es físico o virtual |
| Reputación inicial | Número de reputación más actual del dispositivo |
| Estado actual | Cuarentena pendiente\_evaluación  Admitido  … |
| Dispositivo Virtual | Id | Número autogenerado que es el identificador del dispositivo |
| Fecha última revisión | Fecha en formato Date de la última revisión de seguridad que tuvo el dispositivo |
| Número de vulns pasadas | Cuántas vulns ha tenido o le han sido detectadas |
| Certificado de seguridad | ¿Tiene o no certificado de seguridad? |
| Tipo de dispositivo | Es físico o virtual |
| Reputación inicial | Número de reputación más actual del dispositivo |
| Estado actual | Cuarentena pendiente\_evaluación  Admitido  … |

Hay que definir un escenario que va a ser lo más probable una Smart home y en este escenario qué dispositivos encontramos (si tele, móvil… cosas así, además hay que ver el tema de segregar la red en externa e interna y por ende añadir ese nivel de clasificación en dispositivos), cómo clasificarlos y cómo interactúan entre sí o al incorporarse, irse y demás.

En caso de contar con más tiempo se puede intentar definir otro escenario.

##### ¿Con qué va a contar nuestro escenario?

* Móvil
* Televisión
* Portátil
* Smart Termostato
* Smart bombilla

Hay que tener en cuenta algo y es si queremos partir de un historial vacío en reputación para estos dispositivos o queremos que tenga ya datos. Ya que dependiendo de esto la ejecución del programa debería crear las tablas de una forma u otra.

Suponiendo que acaban de “entrar” estos dispositivos y por tanto aún no hay historial. Lo primero es evaluar cada uno de ellos y hacer una primera carga de datos en las BD de información y reputación para tener unos primeros valores con los que operar.

|  |  |
| --- | --- |
| Casos de uso | |
| Entra dispositivo nuevo y todo bien |  |
| Entra dispositivo nuevo y algo falla |  |
| Entra dispositivo con historial y bien |  |
| Entra dispositivo con historial y algo falla |  |

Docker-compose 🡪 simular dispositivos

*9-6-24*

El Método AHP (Analytic Hierarchy Process) es una técnica estructurada que permite tomar decisiones complejas basadas en múltiples criterios. En tu caso, el AHP puede ser utilizado para determinar los pesos de los atributos que influyen en la confianza de un dispositivo IoT. A continuación, te explico cómo aplicar AHP paso a paso, incluyendo las fórmulas y tablas necesarias.

### Paso 1: Definir los Criterios

Primero, define los criterios (atributos) que quieres evaluar. En tu caso, los criterios pueden ser:

1. Número de vulnerabilidades pasadas

2. Certificado de seguridad

3. Última revisión

4. Tiempo de funcionamiento

5. Actualizaciones de firmware

6. Frecuencia de comunicación

7. Historial de fallos

### Paso 2: Construir la Matriz de Comparación por Pares

Construye una matriz de comparación por pares para los criterios. Cada entrada \( a\_{ij} \) en la matriz representa la importancia relativa del criterio \( i \) respecto al criterio \( j \). Los valores pueden ser tomados de una escala de 1 a 9, donde 1 indica igual importancia y 9 indica extrema importancia de un criterio sobre otro.

Por ejemplo, supongamos que tenemos la siguiente matriz de comparación por pares:

| | Vuln. | Cert. | Rev. | Func. | Act. | Com. | Fall. |

|---------------------|-------|-------|------|-------|------|------|-------|

| \*\*Vulnerabilidades\*\*| 1 | 3 | 5 | 7 | 3 | 5 | 3 |

| \*\*Certificado\*\* | 1/3 | 1 | 3 | 5 | 1 | 3 | 1 |

| \*\*Revisión\*\* | 1/5 | 1/3 | 1 | 3 | 1/3 | 1 | 1/3 |

| \*\*Funcionamiento\*\* | 1/7 | 1/5 | 1/3 | 1 | 1/5 | 1/3 | 1/5 |

| \*\*Actualizaciones\*\* | 1/3 | 1 | 3 | 5 | 1 | 3 | 1 |

| \*\*Comunicación\*\* | 1/5 | 1/3 | 1 | 3 | 1/3 | 1 | 1/3 |

| \*\*Fallos\*\* | 1/3 | 1 | 3 | 5 | 1 | 3 | 1 |

### Paso 3: Normalizar la Matriz

Para normalizar la matriz, divide cada entrada de una columna por la suma de esa columna. Luego, promedia los valores de cada fila para obtener el vector de pesos.

#### Suma de Columnas

| | Vuln. | Cert. | Rev. | Func. | Act. | Com. | Fall. |

|---------------------|-------|-------|------|-------|------|------|-------|

| \*\*Suma\*\* | 2.292 | 6.866 | 16.3 | 28.0 | 6.867| 13.33| 6.867 |

#### Matriz Normalizada

| | Vuln. | Cert. | Rev. | Func. | Act. | Com. | Fall. | \*\*Peso Promedio\*\* |

|---------------------|-------|-------|------|-------|------|------|-------|-------------------|

| \*\*Vulnerabilidades\*\*| 0.436 | 0.437 | 0.306| 0.250 | 0.437| 0.375| 0.437 | 0.382 |

| \*\*Certificado\*\* | 0.145 | 0.146 | 0.184| 0.179 | 0.145| 0.225| 0.145 | 0.167 |

| \*\*Revisión\*\* | 0.087 | 0.049 | 0.061| 0.107 | 0.049| 0.075| 0.048 | 0.068 |

| \*\*Funcionamiento\*\* | 0.062 | 0.029 | 0.020| 0.036 | 0.029| 0.025| 0.029 | 0.033 |

| \*\*Actualizaciones\*\* | 0.145 | 0.146 | 0.184| 0.179 | 0.145| 0.225| 0.145 | 0.167 |

| \*\*Comunicación\*\* | 0.087 | 0.049 | 0.061| 0.107 | 0.049| 0.075| 0.048 | 0.068 |

| \*\*Fallos\*\* | 0.145 | 0.146 | 0.184| 0.179 | 0.145| 0.225| 0.145 | 0.167 |

### Paso 4: Calcular el Peso Normalizado

Promediar los valores de cada fila de la matriz normalizada para obtener el vector de pesos de los atributos:

- \*\*Vulnerabilidades:\*\* 0.382

- \*\*Certificado:\*\* 0.167

- \*\*Revisión:\*\* 0.068

- \*\*Funcionamiento:\*\* 0.033

- \*\*Actualizaciones:\*\* 0.167

- \*\*Comunicación:\*\* 0.068

- \*\*Fallos:\*\* 0.167

### Paso 5: Comprobar la Consistencia

Calcular el índice de consistencia (CI) y la razón de consistencia (CR) para verificar la consistencia de la matriz de comparación por pares.

#### Calcular CI

\[

CI = \frac{\lambda\_{\text{max}} - n}{n - 1}

\]

donde \(\lambda\_{\text{max}}\) es el valor propio máximo de la matriz y \(n\) es el número de criterios.

#### Calcular CR

\[

CR = \frac{CI}{RI}

\]

donde \(RI\) es el índice de consistencia aleatoria, que depende del número de criterios. Para \(n = 7\), \(RI = 1.32\).

Si \(CR < 0.1\), la matriz es consistente.

### Paso 6: Aplicar los Pesos en el Cálculo de Confianza

Utiliza los pesos obtenidos para calcular la confianza del dispositivo:

```java

import java.time.LocalDate;

import java.time.temporal.ChronoUnit;

import java.util.HashMap;

import java.util.Map;

public class Dispositivo {

private int id;

private int numVulnerabilidades;

private boolean tieneCertificado;

private LocalDate ultimaRevision;

private int tiempoFuncionamiento;

private int actualizacionesFirmware;

private int frecuenciaComunicacion;

private int historialFallos;

private Map<String, Double> pesos;

public Dispositivo(int id, int numVulnerabilidades, boolean tieneCertificado, LocalDate ultimaRevision,

int tiempoFuncionamiento, int actualizacionesFirmware, int frecuenciaComunicacion, int historialFallos) {

this.id = id;

this.numVulnerabilidades = numVulnerabilidades;

this.tieneCertificado = tieneCertificado;

this.ultimaRevision = ultimaRevision;

this.tiempoFuncionamiento = tiempoFuncionamiento;

this.actualizacionesFirmware = actualizacionesFirmware;

this.frecuenciaComunicacion = frecuenciaComunicacion;

this.historialFallos = historialFallos;

this.pesos = new HashMap<>();

// Pesos obtenidos del método AHP

this.pesos.put("numVulnerabilidades", 0.382);

this.pesos.put("tieneCertificado", 0.167);

this.pesos.put("ultimaRevision", 0.068);

this.pesos.put("tiempoFuncionamiento", 0.033);

this.pesos.put("actualizacionesFirmware", 0.167);

this.pesos.put("frecuenciaComunicacion", 0.068);

this.pesos.put("historialFallos", 0.167);

}

public double calcularConfianza() {

double sumaPonderada = 0.0;

double sumaPesos = 0.0;

// Evaluar cada atributo y su peso

sumaPonderada += evaluarVulnerabilidades() \* pesos.get("numVulnerabilidades");

sumaPonderada += (tieneCertificado ? 1 : 0) \* pesos.get("tieneCertificado");

sumaPonderada += evaluarUltimaRevision() \* pesos.get("ultimaRevision");

sumaPonderada += evaluarTiempoFuncionamiento() \* pesos.get("tiempoFuncionamiento");

sumaPonderada += evaluarActualizacionesFirmware() \* pesos.get("actualizacionesFirmware");

sumaPonderada += evaluarFrecuenciaComunicacion() \* pesos.get("frecuenciaComunicacion");

sumaPonderada += evaluarHistorialFallos() \* pesos.get("historialFallos");

sumaPesos = pesos.values().stream().mapToDouble(Double::doubleValue).sum();

return sumaPonderada / sumaPesos; // Normalización

}

private double evaluarVulnerabilidades() {

if (numVulnerabilidades == 0

) return 1.0;

if (numVulnerabilidades <= 5) return 0.8;

if (numVulnerabilidades <= 10) return 0.5;

return 0.2;

}

private double evaluarUltimaRevision() {

long diasDesdeUltimaRevision = ChronoUnit.DAYS.between(ultimaRevision, LocalDate.now());

if (diasDesdeUltimaRevision <= 30) return 1.0;

if (diasDesdeUltimaRevision <= 90) return 0.8;

if (diasDesdeUltimaRevision <= 180) return 0.5;

return 0.2;

}

private double evaluarTiempoFuncionamiento() {

if (tiempoFuncionamiento > 1000) return 1.0;

if (tiempoFuncionamiento > 500) return 0.8;

return 0.5;

}

private double evaluarActualizacionesFirmware() {

if (actualizacionesFirmware > 10) return 1.0;

if (actualizacionesFirmware > 5) return 0.8;

return 0.5;

}

private double evaluarFrecuenciaComunicacion() {

if (frecuenciaComunicacion > 100) return 1.0;

if (frecuenciaComunicacion > 50) return 0.8;

return 0.5;

}

private double evaluarHistorialFallos() {

if (historialFallos == 0) return 1.0;

if (historialFallos <= 2) return 0.8;

return 0.5;

}

// Métodos getters y setters, si son necesarios

}

```

### Conclusión

El Método AHP proporciona una forma sistemática y rigurosa de determinar los pesos de los atributos en tu cálculo de confianza. Sigue los pasos de comparación por pares, normalización y verificación de consistencia para obtener los pesos que reflejen la importancia relativa de cada atributo. Luego, aplica estos pesos en tu cálculo de confianza para obtener una evaluación más precisa y fundamentada.

Vamos a actualizar el método AHP para incluir la reputación actual como el factor más importante. Esto implica ajustar la tabla AHP y recalcular los pesos.

### 1. Tabla de Comparación por Pares (AHP)

Primero, definamos la nueva matriz de comparación por pares, teniendo en cuenta que la reputación actual es el factor más importante.

Supongamos que ahora tenemos los siguientes factores:

1. Número de Vulnerabilidades (numVulnerabilidades)
2. Certificado de Seguridad (tieneCertificado)
3. Última Revisión (ultimaRevision)
4. Tiempo de Funcionamiento (tiempoFuncionamiento)
5. Actualizaciones de Firmware (actualizacionesFirmware)
6. Frecuencia de Comunicación (frecuenciaComunicacion)
7. Historial de Fallos (historialFallos)
8. Reputación Actual (reputacionActual)

### Distribución de Pesos Propuesta

1. **Reputación Inicial**: 35%
2. **Número de Vulnerabilidades**: 11.66%
3. **Certificado de Seguridad**: 11.66%
4. **Última Revisión**: 11.66%
5. **Actualizaciones de Firmware**: 7.5%
6. **Historial de Fallos**: 7.5%
7. **Tiempo de Funcionamiento**: 7.5%
8. **Frecuencia de Comunicación**: 7.5%

### Nueva Tabla AHP y Pesos

Vamos a construir la nueva matriz de comparación por pares, reflejando estos ajustes:

|  | **Vulnerabilidades** | **Certificado** | **Última Revisión** | **Actualizaciones Firmware** | **Historial Fallos** | **Tiempo de Funcionamiento** | **Frecuencia Comunicación** | **Reputación Inicial** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Vulnerabilidades | 1 | 1 | 1 | 1.5 | 1.5 | 1.5 | 1.5 | 1/3 |
| Certificado | 1 | 1 | 1 | 1.5 | 1.5 | 1.5 | 1.5 | 1/3 |
| Última Revisión | 1 | 1 | 1 | 1.5 | 1.5 | 1.5 | 1.5 | 1/3 |
| Actualizaciones Firmware | 2/3 | 2/3 | 2/3 | 1 | 1 | 1 | 1 | 1/5 |
| Historial Fallos | 2/3 | 2/3 | 2/3 | 1 | 1 | 1 | 1 | 1/5 |
| Tiempo de Funcionamiento | 2/3 | 2/3 | 2/3 | 1 | 1 | 1 | 1 | 1/5 |
| Frecuencia Comunicación | 2/3 | 2/3 | 2/3 | 1 | 1 | 1 | 1 | 1/5 |
| Reputación Inicial | 3 | 3 | 3 | 5 | 5 | 5 | 5 | 1 |

### Cálculo en Python

Vamos a calcular los nuevos pesos utilizando Python:

Python

import numpy as np

# Matriz de comparación por pares ajustada

matrix = np.array([

[1, 1, 1, 1.5, 1.5, 1.5, 1.5, 1/3],

[1, 1, 1, 1.5, 1.5, 1.5, 1.5, 1/3],

[1, 1, 1, 1.5, 1.5, 1.5, 1.5, 1/3],

[2/3, 2/3, 2/3, 1, 1, 1, 1, 1/5],

[2/3, 2/3, 2/3, 1, 1, 1, 1, 1/5],

[2/3, 2/3, 2/3, 1, 1, 1, 1, 1/5],

[2/3, 2/3, 2/3, 1, 1, 1, 1, 1/5],

[3, 3, 3, 5, 5, 5, 5, 1]

])

# Sumar cada columna

column\_sums = np.sum(matrix, axis=0)

# Normalizar la matriz

normalized\_matrix = matrix / column\_sums

# Promediar cada fila

weights = np.mean(normalized\_matrix, axis=1)

# Mostrar los pesos

Weights

### Resultado

Después de ejecutar el código en Python, obtendremos los nuevos pesos para cada atributo. Supongamos que obtenemos los siguientes valores (ejemplo):

weights = [0.362, 0.116, 0.116, 0.084, 0.084, 0.084, 0.084, 0.06]

### Ajuste de la Clase Dispositivo

Actualizamos los pesos en la clase Dispositivo:

package Dispositivos;

import java.sql.Date;

import java.time.LocalDate;

import java.time.temporal.ChronoUnit;

import java.util.HashMap;

import java.util.Map;

public class Dispositivo {

private int id;

private int numVulnerabilidades;

private boolean tieneCertificado;

private LocalDate ultimaRevision;

private int tiempoFuncionamiento;

private int actualizacionesFirmware;

private int frecuenciaComunicacion;

private int historialFallos;

private int reputacionInicial;

private Map<String, Double> pesos;

public Dispositivo(int id, int numVulnerabilidades, boolean tieneCertificado, LocalDate ultimaRevision,

int tiempoFuncionamiento, int actualizacionesFirmware, int frecuenciaComunicacion, int historialFallos, int reputacionInicial) {

this.id = id;

this.numVulnerabilidades = numVulnerabilidades;

this.tieneCertificado = tieneCertificado;

this.ultimaRevision = ultimaRevision;

this.tiempoFuncionamiento = tiempoFuncionamiento;

this.actualizacionesFirmware = actualizacionesFirmware;

this.frecuenciaComunicacion = frecuenciaComunicacion;

this.historialFallos = historialFallos;

this.reputacionInicial = reputacionInicial;

this.pesos = new HashMap<>();

// Pesos obtenidos del método AHP ajustado

this.pesos.put("numVulnerabilidades", 0.116);

this.pesos.put("tieneCertificado", 0.116);

this.pesos.put("ultimaRevision", 0.116);

this.pesos.put("actualizacionesFirmware", 0.075);

this.pesos.put("historialFallos", 0.075);

this.pesos.put("tiempoFuncionamiento", 0.075);

this.pesos.put("frecuenciaComunicacion", 0.075);

this.pesos.put("reputacionInicial", 0.35); // Mayor peso para la reputación inicial

}

// Métodos getters y setters...

// Calcular la confianza de un dispositivo

public double calcularConfianza() {

double sumaPonderada = 0.0;

double sumaPesos = 0.0;

// Evaluar cada atributo y su peso

sumaPonderada += evaluarVulnerabilidades() \* pesos.get("numVulnerabilidades");

sumaPonderada += (tieneCertificado ? 1 : 0) \* pesos.get("tieneCertificado");

sumaPonderada += evaluarUltimaRevision() \* pesos.get("ultimaRevision");

sumaPonderada += evaluarActualizacionesFirmware() \* pesos.get("actualizacionesFirmware");

sumaPonderada += evaluarHistorialFallos() \* pesos.get("historialFallos");

sumaPonderada += evaluarTiempoFuncionamiento() \* pesos.get("tiempoFuncionamiento");

sumaPonderada += evaluarFrecuenciaComunicacion() \* pesos.get("frecuenciaComunicacion");

sumaPonderada += evaluarReputacionInicial() \* pesos.get("reputacionInicial");

sumaPesos = pesos.values().stream().mapToDouble(Double::doubleValue).sum();

return sumaPonderada / sumaPesos; // Normalización

}

private double evaluarVulnerabilidades() {

if (numVulnerabilidades == 0) return 1.0;

if (numVulnerabilidades <= 5) return 0.8;

if (numVulnerabilidades <= 10) return 0.5;

return 0.2;

}

private double evaluarUltimaRevision() {

long diasDesdeUltimaRevision = ChronoUnit.DAYS.between(ultimaRevision, LocalDate.now());

if (diasDesdeUltimaRevision <= 30) return 1.0;

if (diasDesdeUltimaRevision <= 90) return 0.8;

if (diasDesdeUltimaRevision <= 180) return 0.5;

return 0.2;

}

private double evaluarActualizacionesFirmware() {

if

Entendido. Vamos a continuar con la implementación en Java teniendo en cuenta los nuevos pesos asignados a cada atributo del dispositivo según el método AHP ajustado:

### Implementación en Java

Primero, ajustaremos la clase Dispositivo para reflejar los nuevos pesos y la lógica para calcular la confianza:

package Dispositivos;

import java.time.LocalDate;

import java.time.temporal.ChronoUnit;

import java.util.HashMap;

import java.util.Map;

public class Dispositivo {

private int id;

private int numVulnerabilidades;

private boolean tieneCertificado;

private LocalDate ultimaRevision;

private int tiempoFuncionamiento;

private int actualizacionesFirmware;

private int frecuenciaComunicacion;

private int historialFallos;

private int reputacionInicial;

private Map<String, Double> pesos;

public Dispositivo(int id, int numVulnerabilidades, boolean tieneCertificado, LocalDate ultimaRevision,

int tiempoFuncionamiento, int actualizacionesFirmware, int frecuenciaComunicacion, int historialFallos, int reputacionInicial) {

this.id = id;

this.numVulnerabilidades = numVulnerabilidades;

this.tieneCertificado = tieneCertificado;

this.ultimaRevision = ultimaRevision;

this.tiempoFuncionamiento = tiempoFuncionamiento;

this.actualizacionesFirmware = actualizacionesFirmware;

this.frecuenciaComunicacion = frecuenciaComunicacion;

this.historialFallos = historialFallos;

this.reputacionInicial = reputacionInicial;

this.pesos = new HashMap<>();

// Pesos obtenidos del método AHP ajustado

this.pesos.put("numVulnerabilidades", 0.116);

this.pesos.put("tieneCertificado", 0.116);

this.pesos.put("ultimaRevision", 0.116);

this.pesos.put("actualizacionesFirmware", 0.075);

this.pesos.put("historialFallos", 0.075);

this.pesos.put("tiempoFuncionamiento", 0.075);

this.pesos.put("frecuenciaComunicacion", 0.075);

this.pesos.put("reputacionInicial", 0.35); // Mayor peso para la reputación inicial

}

// Métodos getters y setters...

// Calcular la confianza de un dispositivo

public double calcularConfianza() {

double sumaPonderada = 0.0;

double sumaPesos = 0.0;

// Evaluar cada atributo y su peso

sumaPonderada += evaluarVulnerabilidades() \* pesos.get("numVulnerabilidades");

sumaPonderada += (tieneCertificado ? 1 : 0) \* pesos.get("tieneCertificado");

sumaPonderada += evaluarUltimaRevision() \* pesos.get("ultimaRevision");

sumaPonderada += evaluarActualizacionesFirmware() \* pesos.get("actualizacionesFirmware");

sumaPonderada += evaluarHistorialFallos() \* pesos.get("historialFallos");

sumaPonderada += evaluarTiempoFuncionamiento() \* pesos.get("tiempoFuncionamiento");

sumaPonderada += evaluarFrecuenciaComunicacion() \* pesos.get("frecuenciaComunicacion");

sumaPonderada += evaluarReputacionInicial() \* pesos.get("reputacionInicial");

sumaPesos = pesos.values().stream().mapToDouble(Double::doubleValue).sum();

return sumaPonderada / sumaPesos; // Normalización

}

private double evaluarVulnerabilidades() {

if (numVulnerabilidades == 0) return 1.0;

if (numVulnerabilidades <= 5) return 0.8;

if (numVulnerabilidades <= 10) return 0.5;

return 0.2;

}

private double evaluarUltimaRevision() {

long diasDesdeUltimaRevision = ChronoUnit.DAYS.between(ultimaRevision, LocalDate.now());

if (diasDesdeUltimaRevision <= 30) return 1.0;

if (diasDesdeUltimaRevision <= 90) return 0.8;

if (diasDesdeUltimaRevision <= 180) return 0.5;

return 0.2;

}

private double evaluarActualizacionesFirmware() {

if (actualizacionesFirmware <= 1) return 1.0;

if (actualizacionesFirmware <= 3) return 0.8;

if (actualizacionesFirmware <= 5) return 0.5;

return 0.2;

}

private double evaluarHistorialFallos() {

if (historialFallos == 0) return 1.0;

if (historialFallos <= 2) return 0.8;

if (historialFallos <= 5) return 0.5;

return 0.2;

}

private double evaluarTiempoFuncionamiento() {

if (tiempoFuncionamiento >= 5) return 1.0;

if (tiempoFuncionamiento >= 3) return 0.8;

if (tiempoFuncionamiento >= 1) return 0.5;

return 0.2;

}

private double evaluarFrecuenciaComunicacion() {

if (frecuenciaComunicacion >= 5) return 1.0;

if (frecuenciaComunicacion >= 3) return 0.8;

if (frecuenciaComunicacion >= 1) return 0.5;

return 0.2;

}

private double evaluarReputacionInicial() {

if (reputacionInicial >= 80) return 1.0;

if (reputacionInicial >= 60) return 0.8;

if (reputacionInicial >= 40) return 0.5;

return 0.2;

}

public static void main(String[] args) {

// Ejemplo de uso

LocalDate ultimaRevision = LocalDate.of(2023, 6, 13);

Dispositivo dispositivo = new Dispositivo(1, 8, true, ultimaRevision, 2, 3, 4, 2, 85);

double confianza = dispositivo.calcularConfianza();

System.out.println("Confianza del dispositivo: " + confianza);

}

}

### Explicación

En esta implementación:

* La clase Dispositivo tiene atributos que representan características del dispositivo y pesos asociados a cada atributo según el método AHP.
* Se definen métodos para evaluar cada atributo en función de su estado actual y se calcula la confianza del dispositivo aplicando los pesos respectivos.
* En el método main se muestra un ejemplo de cómo se puede crear un objeto Dispositivo y calcular su confianza.

Esta estructura te permitirá calcular la confianza de un dispositivo de acuerdo a los nuevos pesos asignados a cada atributo, priorizando la reputación inicial seguida de los demás factores según las ponderaciones especificadas.

*11-6-24*

Falta que la reputación se tenga en cuenta a la hora de calcular la confianza y que luego se compute de nuevo para actualizarse en la tabla de la reputación.

Luego toca todo lo gordo de MQTT que creo que va a requerir de muuuuuuuuuuuuuchas terminales, tantas como objetos creados.

*30-6-24*

Tienes razón, para una comunicación clara y efectiva, los componentes deben suscribirse directamente a los topics específicos a los que se envían los mensajes. Vamos a ajustar la jerarquía de topics y el ejemplo de uso en el código para reflejar esto:

### Jerarquía de Topics Ajustada

```

iot/

confianza/

to\_anomalia

from\_anomalia

to\_devices/

device1

device2

...

from\_devices/

device1

device2

...

anomalia/

to\_confianza

from\_confianza

to\_devices/

device1

device2

...

from\_devices/

device1

device2

...

devices/

device1/

to\_confianza

from\_confianza

to\_anomalia

from\_anomalia

to\_device2

from\_device2

...

device2/

to\_confianza

from\_confianza

to\_anomalia

from\_anomalia

to\_device1

from\_device1

...

```

### Explicación de la Jerarquía:

1. \*\*Comunicación entre "confianza" y "anomalía":\*\*

- `iot/anomalia/to\_confianza`: Topic donde "anomalía" manda mensajes a "confianza".

- `iot/confianza/to\_anomalia`: Topic donde "confianza" manda mensajes a "anomalía".

2. \*\*Comunicación entre "confianza" y los dispositivos:\*\*

- `iot/devices/device1/to\_confianza`: Topic donde `device1` manda mensajes a "confianza".

- `iot/confianza/to\_devices/device1`: Topic donde "confianza" manda mensajes a `device1`.

- `iot/devices/device2/to\_confianza`: Topic donde `device2` manda mensajes a "confianza".

- `iot/confianza/to\_devices/device2`: Topic donde "confianza" manda mensajes a `device2`.

3. \*\*Comunicación bidireccional entre dispositivos:\*\*

- `iot/devices/device1/to\_device2`: Topic donde `device1` manda mensajes a `device2`.

- `iot/devices/device2/to\_device1`: Topic donde `device2` manda mensajes a `device1`.

### Ejemplo de Uso en Código Ajustado:

```python

import paho.mqtt.client as mqtt

# Callback para cuando se recibe un mensaje

def on\_message(client, userdata, msg):

print(f"Message received on topic {msg.topic}: {msg.payload.decode()}")

# Crear cliente MQTT

client = mqtt.Client()

# Asignar callback

client.on\_message = on\_message

# Conectar al broker

client.connect("broker\_address", 1883, 60)

# Suscribirse a topics

client.subscribe("iot/anomalia/to\_confianza")

client.subscribe("iot/devices/device1/to\_confianza")

client.subscribe("iot/devices/device2/to\_confianza")

client.subscribe("iot/confianza/to\_anomalia")

client.subscribe("iot/devices/device1/to\_anomalia")

client.subscribe("iot/devices/device2/to\_anomalia")

client.subscribe("iot/devices/device1/to\_device2")

client.subscribe("iot/devices/device2/to\_device1")

# Publicar un mensaje

client.publish("iot/confianza/to\_anomalia", "Mensaje de confianza a anomalía")

client.publish("iot/confianza/to\_devices/device1", "Mensaje de confianza a device1")

client.publish("iot/devices/device1/to\_device2", "Mensaje de device1 a device2")

# Loop para mantener la conexión y escuchar mensajes

client.loop\_forever()

```

### Consideraciones:

- \*\*Suscripción Directa\*\*: Cada componente se suscribe directamente a los topics desde los cuales espera recibir mensajes.

- \*\*Organización Clara\*\*: La estructura de topics sigue siendo organizada y facilita la administración de la comunicación.

- \*\*Flexibilidad\*\*: La jerarquía de topics es flexible y puede expandirse fácilmente para añadir más dispositivos o tipos de comunicación.

Con esta configuración, se asegura que los mensajes se envían y reciben correctamente entre los componentes del sistema.

## Bibliografía – Páginas a tener en cuenta/recordar

<https://blogs.oracle.com/javamagazine/post/java-mqtt-iot-message-queuing>

<https://mosquitto.org/download/>

<https://tutorials.solace.dev/java-mqtt-paho/publish-subscribe/>

<https://www.javacodegeeks.com/2017/01/getting-started-mqtt-java.html>

<https://learn.microsoft.com/es-es/windows/wsl/setup/environment>

<https://hivemq.github.io/mqtt-cli/docs/installation/>

<https://www.youtube.com/watch?v=r7wKPz3KObA>

<https://www.youtube.com/watch?v=GRsWhLtcgVs>

<https://www.youtube.com/watch?v=O10JqPeC5SA>

<https://manualdelatex.com/tutoriales>

<https://aprendeconalf.es/latex-manual/manual-latex.pdf>

<https://iep.edu.es/que-son-metodos-agiles-tipos-mas-comunes/>

<https://www.atlassian.com/es/agile/scrum>

<https://blog.hubspot.es/marketing/metodologia-agile>

<https://aprendeconalf.es/latex-manual/formateo.html>

<https://aws.amazon.com/es/what-is/iot/>

<https://www.incibe.es/sites/default/files/contenidos/guias/doc/guia-de-seguridad-iot.pdf>

<https://www.escueladenegociosydireccion.com/revista/business/scrum-framework-agiliza-trabajo-equipo/>

<https://internetdelascosas.xyz/articulo.php?id=3220>

<https://mqttx.app/cli>

<https://www.rabbitmq.com>

<http://desarrolloweb.dlsi.ua.es/cursos/2015/herramientas-investigacion/que-es-latex>

<https://en.wikipedia.org/wiki/Overleaf>

<https://aprendiendoarduino.wordpress.com/2018/11/19/mosquitto/>

<https://www.piensasolutions.com/blog/que-es-ubuntu-guia-sobre-este-sistema-operativo>

<https://www.emqx.com/en/blog/category/mqtt-programming>

<https://github.com/emqx/MQTTX?tab=readme-ov-file>

<https://mqtt.org/software>

<https://github.com/eclipse/paho.mqtt.python/tree/master/examples>

<http://www.steves-internet-guide.com/encrypting-the-mqtt-payload-python-example/>