

1 Introducción

Las redes de Internet de las Cosas (IoT) pueden ser implementadas utilizando diversos protocolos de comunicaciones. Los mas habituales al crear estas redes son: HTTP, CoAP, MQTT y XMPP, en la selección de estos protocolos, dejamos de un lados los medios de comunicación que, exclusivamente, son M2M (Machine to Machine) debido a que estos últimos, por su definición de comunicación exclusivamente entre maquinas, no entran en la categoría de protocolo IoT, propiamente tal. En la Tabla 1 se comparan los protocolos IoT antes mencionados, se expone las diversas características discriminatorias al momento de realizar la elección del mejor protocolo, según la necesidad.

Característica	HTTP [7]	CoAP [15]	MQTT [1]	XMPP [10][14]
Request/ Response	✓	✓	✗	✓
Publish/ Subscribe	✗	✗	✓	✓
Multicast	✗	✓	✗	✓
Events or Push	✓	✓	✓	✓
Bypasses firewall	✗	(✓)	✓	✓
Federation	✗	✗	✗	✓✓
Authentication	✓	✓	✓	✓
Network Identity	(✓)	(✓)	✗	✓
Authorization	✗	✗	✗	✓✓
Encryption	✓	✓	✓	✓
End-to-end encryption	✗	✗	✗	✓✓
Compression	✓	✗	✗	✓
Streaming	✓	✗	✓	✓
Reliable messaging	✗	✗	✓✓	✗
Message Queues	✗	✗	✗	✗

Table 1: Comparativa protocolos en IoT mas utilizados, donde el símbolo "✗" significa, no cumple la característica, el signo "✓" refiere a que la cumple parcialmente, "(✓)" que lo hace parcialmente, y finalmente "✓✓" que hay más de un componente que realiza la función.

En cada uno de los protocolos listados en la Tabla 1 existen desafíos en el área de seguridad. Por ejemplo, con HTTP se pueden crear sistemas altamente robustos y seguros, sin embargo, debido a que patrón cliente-servidor los sistemas a crear con este protocolo son virtualmente inamovibles, en el aspecto de complejidad, ya que, es muy complejo cambiar algún componente del sistema, esto es un gran contra en ambientes flexibles como necesitamos en la actualidad. Por otro lado CoAP[15], es un protocolo de transferencia RESTful para nodos y redes con restricciones. Es escalable, tiene la posibilidad de implementar Datagram Transport Layer Security (DTLS)[9], sin embargo, aún así posee una pobre capa de seguridad y el Bypass de Firewall es bastante limitado, lo cual hace que exponer servicios a través de Internet es bastante engorroso, monetariamente costoso y complejo, por otro lado el protocolo Message Queuing Telemetry Transport (MQTT) esta basado en el patrón de diseño Publish-Suscribe, a diferencia de CoAP realiza un eficaz bypass de los firewalls, y soporta encriptación SSL/TLS, características realmente importantes al cuando hablamos de redes IoT , sin embargo, el protocolo MQTT tiene serias vulnerabilidades conocidas[11], a tal nivel de gravedad, que, incluso gobiernos han llamado a no usarlo y han prohibido su uso en sistemas gubernamentales.

Al final, esta listado el Protocolo extensible de mensajería y comunicación de presencia (XMPP - por su siglas en ingles)[10][14], este es un protocolo abierto basado en flujos XML para la comunicación en tiempo real. XMPP fue desarrollado en el año 1998 con el nombre Jabber, después estandarizado por la Internet Engineering Task Force (IETF) liberando así su primera versión oficial bajo el nombre de Jabber, para luego en el año 2002 pasarse a conocer con el nombre XMPP. El protocolo es estandarizado por Fundación de Estándares XMPP (XSF - por sus siglas en ingles), quienes corrigen errores y extienden las funcionalidades de este protocolo, por medio de lo que se conocen por Protocolos de Extensión de XMPP (XEPs)[12]. Inicialmente fue desarrollado para mensajería instantánea, pero con el paso del tiempo, ha sido utilizado ampliamente para diversos fines

iniciando desde IM, como para videojuegos, notificaciones Push, Base de datos descentralizadas, redes sociales, microblogging, sistemas de videoconferencias, y finalmente Internet de las Cosas. Es un protocolo muy versátil, probado, con una larga data, y escalable debido a esto sumado al soporte de XSF, es lo hace que XMPP sea un protocolo actualizado y seguro.

Como podemos apreciar en la Tabla 1 XMPP es el que cumple de mejor manera las características listadas para seleccionar un protocolo IoT con difusión en la red de Internet.

A pesar de lo mencionado, existe un problema habitual al momento de crear una red XMPP-Iot, el cual reside en brindar una experiencia transparente para el usuario final. El cual desea adicionar nuevas “cosas” (tipo Plug and Play), tales como sensores, actuadores, controladores o concentradores a la red. Lamentablemente, para este, el uso interoperable de Things de diversos fabricantes de dispositivos, no es viable, ya que cada uno de estos, utiliza protocolos distintos o redes cerradas. Para solucionar esta problemática existe en XMPP, el registro en banda, mecanismo descrito en el XEP-0077: In-Band Registration, el que permite la creación de nuevas identidades - donde cada una representa un dispositivo - utilizando el mismo canal de comunicaciones para conexión a los servidores de la red XMPP-IoT, de este modo, cada fabricante puede crear identidades para sus dispositivos en la red, simplificando el control unificado de estos. El problema en este caso, es que no hay control sobre cuantas identidades crea que fabricante, ni mucho menos un limite en la cantidad del registro de nuevas cuentas para los dispositivos de un fabricante.

En efecto, es útil cuando hablamos redes XMPP-IoT de un sólo fabricante. En este caso, la gran mayoría de los servidores, tienen implementada el XEP-0077, para realizar el registro en banda, usando el mismo protocolo XMPP, así se evita usar el método alternativo de crear una webAPI para crear nuevas identidades, añadiendo posibles vulnerabilidades extra a nuestra red y reduciendo la flexibilidad de esta. El problema fundamental con el Registro en banda, propiamente tal, es que deja totalmente expuesta nuestra red XMPP-IoT al ataque de usuarios/bot maliciosos, ya que pueden sobrecargar nuestros servidores, con solicitudes virtualmente ilimitadas de creación de nuevas cuentas, y los servidores no tienen la capacidad de discriminar las solicitudes benignas de las nocivas. Frente a tal desafío, en los inicios de XMPP, se desarrollo el XEP-0158: CAPTCHA Forms, que describe el mecanismo que opera en conjunto del registro en-banda, para añadirle CAPTCHAs a este ultimo. Esta solución, fue sumamente útil para proteger las redes XMPP de los ataques de bots maliciosos, durante los primeros años de la popularidad de la mensajería instantánea, cuando el poder de computo de los ordenadores personales era bastante inferior al de la actualidad. Pero hoy, no es suficiente, debido a que por medio de visión computacional es relativamente sencillo resolver los CAPTCHAs[3].

De todos modos, cuando se utiliza el XEP-0077 en conjunto de CAPTCHAs Forms para filtrar a los usuarios maliciosos, las “cosas” - que en estricto rigor son bots - no pueden registrarse de forma automatizada en la red XMPP-IoT. Así que esta opción, si fuese segura, sabiendo que no lo es, no es una opción viable para crear dispositivos “plug and play”. Para este fin, es decir, crear una red XMPP-IoT con la capacidad de registrar en-banda nuevas identidades, generalmente se activa el registro en banda, sin habilitar CAPTCHAs Forms. Lo cual, nos permite tener dispositivos “plug and play”. Esto, al mismo tiempo, abre una brecha en la seguridad de la red XMPP-IoT. Permitiendo que cualquiera - cliente inocuo o maligno, bot benigno o malicioso - pueda visualizar a nivel de red el servidor y este plenamente facultado para registrar identidades en el servidor. Lo cual abre una brecha en la seguridad en la red XMPP-IoT.

Este trabajo posee las siguientes secciones. Sección 2 presenta la propuesta para disminuir los ataques de red y dar una alternativa de interoperabilidad a los fabricantes de dispositivos. Sección 3 describe como esta propuesta es implementada en un servidor y cliente XMPP para verificar con un experimento que efectivamente los ataques se reducen a cero. Sección 4 describe las conclusiones y trabajos futuros.

2 Propuesta

Como ya hemos visto en la Tabla 1, XMPP es más robusto en términos de seguridad que el resto de protocolos comparados, ya que existen XEPs que tributan en esta área. Tal como lo es el protocolo de extensión de XMPP XEP-0348: Signing Forms[16], que describe el mecanismo para realizar el firmado de formularios, con credenciales que no tienen directa relación con la conexión - usando el algoritmo modificado de OAuth 1.0[8] - para así concretar el registro de nuevas identidades en la red XMPP, esto es, usando el protocolo de extensión XEP-0077[13] que describe el mecanismo para realizar un registro en banda de nuevas cuentas en el servidor XMPP.

En términos prácticos, al unificar estas dos tecnologías, es decir, el registro en banda junto al proceso de firmado de formularios de registro, se dota a la red XMPP-IoT la capacidad de otorgar el permiso para registrar

nuevas identidades a los fabricantes de dispositivos IoT (Cosas). Donde las credenciales para crear estas nuevas identidades en la red, no tienen relación con la conexión o ingreso a la red XMPP, estas toman el nombre de credenciales de consumidor que es el par “consumer key” y “consumer secret”, las que a su vez se asocian a un cantidad de registro de nuevas identidades permitidas en la red por dicha credencial, como por ejemplo: la consumer key=00000000 perteneciente al fabricante de dispositivos BinaryLamp, sólo podrá crear 1000 identidades en el servidor XMPP-IoT. Posibilitando, de este modo, la interoperabilidad de dispositivos IoT de distintas compañías, sacando provecho de la federalización de XMPP o mediante la concesión de credenciales de consumidor a los fabricantes de dispositivos IoT, para que estos puedan registrar nuevas identidades en la red, por medio de estas credenciales o dos opciones anteriores en su conjunto.

Para dar solución a la problemática expuesta, debemos seleccionar un servidor XMPP. En la actualidad existe una gran cantidad de servidores XMPP. Existen de uso libre o de pago, de código abierto o propietario y desarrollado en diversos lenguajes de programación, documentación variada, con una comunidad existente en torno al servidor, que operan sobre diferentes sistemas operativos y con diversas XEPs implementadas. *IoT Broker* (<https://waher.se/Broker.md>), servidor XMPP de uso no libre, no es open-source y no hay una comunidad incipiente en torno al proyecto. *AstraChat Isode M-Link*, su uso es de pago. *ejabberd*, *Tigase* y *Openfire* son de uso libre. Los primeros dos poseen una comunidad también incipiente en torno al desarrollo de nuevas características. Por otro lado, tenemos *Openfire* que es de uso libre, open-source, está codificado en lenguaje Java, funciona bajo los Sistemas Operativos Linux, macOS, Solaris y Windows, inicialmente desarrollado por la empresa Jive Software, quienes ayudador a formar la gran comunidad que mantiene activo el proyecto en la actualidad, llamada Igniterealtime. La misma comunidad posee *SMACK*, una librería que tiene una potente API para crear clientes XMPP, la que ostenta las mismas características mencionadas del servidor.

El servidor XMPP Openfire incluye soporte completo de RFC XMPP, así como las extensiones más comunes. La Tabla 2 a continuación detalla el nivel de soporte para los requisitos establecidos por XEP-0302: XMPP Compliance Suites 2012.

Especificación	Soportado
RFC 6120: XMPP Core	Sí
RFC 6121: XMPP IM	Sí
RFC 7622: XMPP ADDR	Sí
XEP-0030: Service Discovery	Sí
XEP-0114: Jabber Component Protocol	Sí

Table 2: Tabla: Openfire Soporte XMPP Core

El cumplimiento de soporte de nivel avanzado incluye el conjunto básico completo, así como características más avanzadas de uso común para los clientes XMPP. La Tabla 3 detalla el nivel de soporte para los requisitos establecidos por XMPP Compliance Suites 2012.

Especificación	Soportado
XEP-0115: Entity Capabilities	Sí
XEP-0191: Blocking Command	No
XEP-0124: Bidirectional-streams Over Synchronous HTTP (BOSH)	Sí
XEP-0206: XMPP Over BOSH	Sí
XEP-0054: vcard-temp	Sí
XEP-0163: Personal Eventing Protocol	Sí
XEP-0045: Multi-User Chat	Sí
XEP-0198: Stream Management	Parcialmente

Table 3: Tabla: Openfire soporte avanzado

En la Tabla 4 se listan los XEP compatibles con Openfire. Los XEP que solo requieren soporte del lado del cliente se omiten.

Para crear el cliente usamos **SMACK**, el cual soporta las siguientes XEPs, listadas en la Tabla 5 y Tabla 6.

Especificación
XEP-0004: Data Forms
XEP-0012: Last Activity
XEP-0013: Flexible Offline Message Retrieval
XEP-0030: Service Discovery
XEP-0033: Extended Stanza Addressing
XEP-0049: Private XML Storage
XEP-0050: Ad-Hoc Commands
XEP-0054: vcard-temp
XEP-0055: Jabber Search
XEP-0059: Result Set Management
XEP-0060: Publish-Subscribe
XEP-0065: SOCKS5 Bytestreams
XEP-0077: In-Band Registration
XEP-0078: Non-SASL Authentication
XEP-0082: XMPP Date and Time Profiles
XEP-0086: Error Condition Mappings
XEP-0092: Software Version
XEP-0096: File Transfer
XEP-0106: JID Escaping
XEP-0114: Jabber Component Protocol
XEP-0115: Entity Capabilities
XEP-0124: HTTP Binding
XEP-0126: Invisibility
XEP-0128: Service Discovery Extensions
XEP-0138: Stream Compression
XEP-0160: Best Practices for Handling Offline Messages
XEP-0163: Personal Eventing via Pubsub
XEP-0198: Stream Management (partial)
XEP-0202: Entity Time
XEP-0203: Delayed Delivery
XEP-0280: Message Carbons

Table 4: XEPs soportadas por Openfire

Especificación
n/a-Google GCM JSON payload
n/a-Group Chat Invitations
n/a-Jive Properties
XEP-0004-Data Forms
XEP-0012-Last Activity
XEP-0013-Flexible Offline Message Retrieval
XEP-0016-Privacy Lists
XEP-0022-Message Events
XEP-0030-Service Discovery
XEP-0033-Extended Stanza Addressing
XEP-0045-Multi User Chat
XEP-0047-In-Band Bytestreams
XEP-0048-Bookmarks
XEP-0049-Private Data
XEP-0050-Ad-Hoc Commands
XEP-0054-vcard-temp
XEP-0055-Jabber Search
XEP-0059-Result Set Management
XEP-0060-PubSub
XEP-0065-SOCKS5 Bytestreams
XEP-0071-XHTML-IM
XEP-0077-In-Band Registration
XEP-0079-Advanced Message Processing
XEP-0080-User Location
XEP-0082-XMPP Date Time Profiles
XEP-0085-Chat State Notifications
XEP-0090-Time Exchange
XEP-0092-Software Version
XEP-0093-Roster Item Exchange
XEP-0095-Stream Initiation
XEP-0096-SI File Transfer
XEP-0115-Entity Capabilities
XEP-0116-Jingle
XEP-0122-Data Forms Validation
XEP-0133-Service Administration

Table 5: XEPs soportadas por librería SMACK

Especificación
XEP-0138-Stream Compression
XEP-0141-Data Forms Layout
XEP-0163-Personal Eventing Protocol
XEP-0184-Message Delivery Receipts
XEP-0191-Blocking Command
XEP-0199-XMPP Ping
XEP-0202-Entity Time
XEP-0203-Delayed Delivery
XEP-0206-XMPP Over BOSH
XEP-0224-Attention
XEP-0231-Bits of Binary
XEP-0280-Message Carbons
XEP-0296-Best Practices for Resource Locking
XEP-0308-Last Message Correction
XEP-0313-Message Archive Management
XEP-0319-Last User Interaction in Presence
XEP-0323-Internet of Things - Sensor Data
XEP-0324-Internet of Things - Provisioning
XEP-0325-Internet of Things - Control
XEP-0332-HTTP over XMPP transport
XEP-0333-Chat Markers
XEP-0334-Message Processing Hints
XEP-0335-JSON Containers
XEP-0347-Internet of Things - Discovery
XEP-0352-Client State Indication
XEP-0357-Push Notifications
XEP-0359-Stable and Unique Stanza IDs
XEP-0363-HTTP File Upload
XEP-0372-References
XEP-0382-Spoiler Messages
XEP-0384-OMEMO Multi End Message and Object Encryption
XEP-0392-Consistent Color Generation
XEP-0394-Message Markup
XEP-xxxx-Multi-User Chat Light

Table 6: XEPs soportadas por librería SMACK

En este trabajo proponemos desarrollar e implementar el mecanismo descrito en la especificación **XEP-0348: Signing Forms** sobre la **XEP-0077: In-band Registration**, tanto en el servidor como en el cliente, para que, de este modo, se pueda asignar la facultad sólo a cierto lote definido de clientes crear identidades en la red XMPP-IoT. Con esto reduciremos a cero la creación de identidades por usuarios o bot maliciosos mediante la vulnerabilidad de la XEP-0077. En particular para validar nuestra propuesta realizaremos un experimento implementaremos nuestra propuesta en el cliente *SMACK* y en el servidor *Openfire* tal como se aprecia en las Figuras 1 y 2.

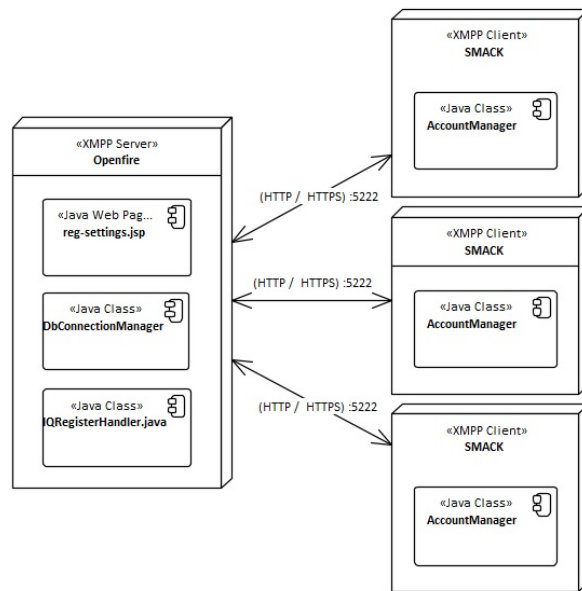


Figure 1: Diagrama de despliegue

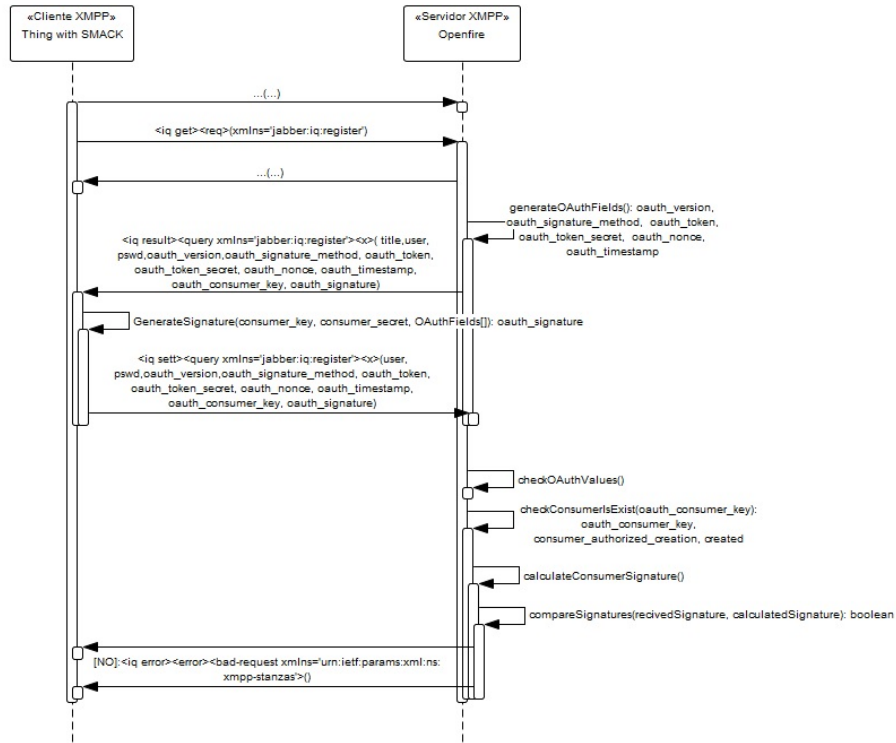
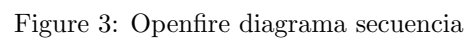


Figure 2: Diagrama de secuencia de Signing Forms

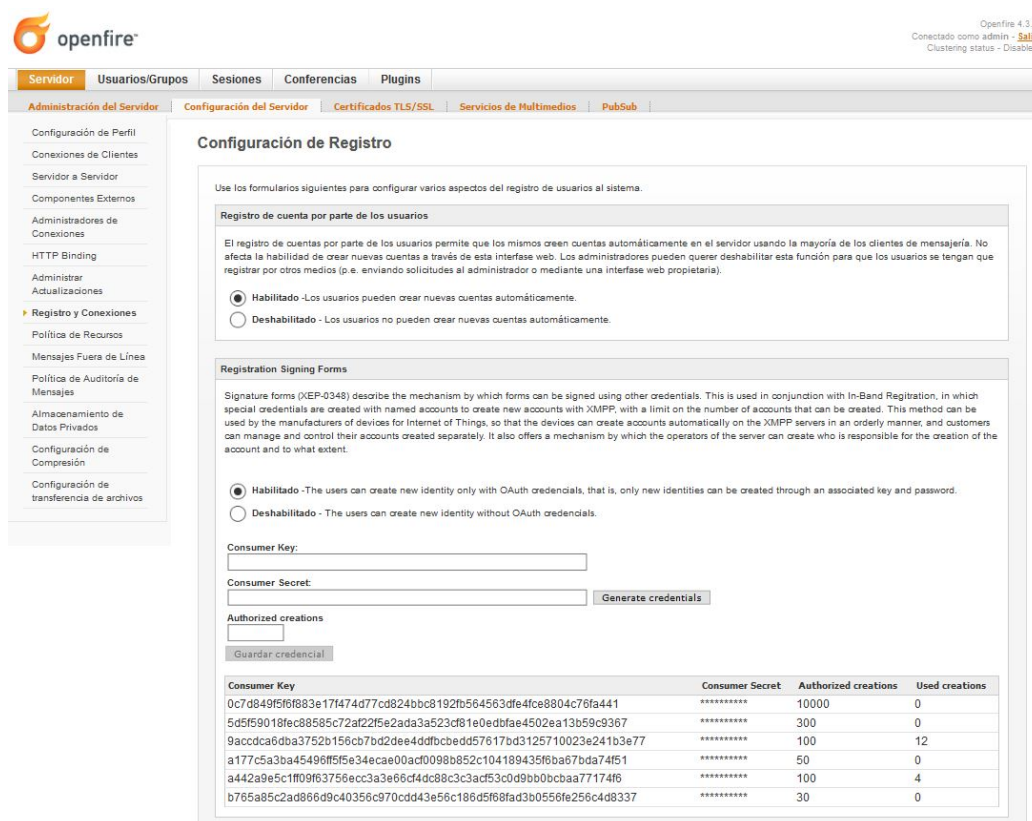
Para ello se planifico el desarrollo de este nuevo mecanismo en el servidor XMPP Openfire y en la librería SMACK, desarrollo que consto de cuatro liberaciones relevantes:

1. Bot que demuestra la vulnerabilidad existente en la XEP-0077 en el contexto de IoT.
2. Servidor Openfire con las mejoras para incorporar el XEP-0348.
3. Librería SMACK incorporando mejoras para implmentando el XEP-0348.
4. Cliente Thing que utiliza la versión mejorada de SMACK, para validar la propuesta.

Inicialmente en el desarrollo se codifico un pequeño bot, utilizando la librería SMACK, el cual tiene la facultad de crear, virtualmente, ilimitadas identidades en el servidor Openfire. Los resultados de este, se pueden ver en la Sección 3. Después de la creación del bot, se continuo con el análisis, diseño, codificación y de respectivas pruebas de la mejora del Servidor XMPP. Genero los resultados visualizados en Figura 4 en donde se muestra el Diagrama de clases de Openfire, en donde se puede visualizar las clases afectadas en el proyecto, incluyendo “IqRegisterHandler.class”, “Form.class”, “DataForm.class”, “DataForms.Item.class”, DbConnection.class y “reg-settings”. Este ultimo responsable de la visualización de la consola web, que como su nombre dice, es el responsable de la configuración de los mecanismos de registro al interior del servidor Openfire. Para completar la explicación del funcionamiento de la implementación del XEP-0348 en Openfire, añadimos la Figura 3.



Luego del desarrollo, pruebas y refactoring, tenemos que la consola web del servidor, el cual es el resultado evidentemente visible, tal como lo podemos apreciar en la Figura 5 la segunda sección lleva el titulo de “Registration Signing Forms”, el administrador de la red XMPP, puede generar credenciales de consumidor, a la que se le asigna un cantidad de nuevas cuentas que se pueden crear con estas credenciales. Al mismo tiempo, se puede inferir que cuando se llega a limite de la cantidad permitida, no se pueden crear más cuenta con dicha credencial.



Openfire 4.3.0
Conectado como admin - [Salir](#)
Clustering status - Disabled

Servidor Usuarios/Grupos Sesiones Conferencias Plugins

Administración del Servidor Configuración del Servidor Certificados TLS/SSL Servicios de Multimedia PubSub

Configuración de Perfil
Conexiones de Clientes
Servidor a Servidor
Componentes Externos
Administradores de Conexiones
HTTP Binding
Administrar Actualizaciones
Registro y Conexiones
Política de Recursos
Mensajes Fuera de Línea
Política de Auditoría de Mensajes
Almacenamiento de Datos Privados
Configuración de Compresión
Configuración de transferencia de archivos

Configuración de Registro

Use los formularios siguientes para configurar varios aspectos del registro de usuarios al sistema.

Registro de cuenta por parte de los usuarios

El registro de cuentas por parte de los usuarios permite que los mismos creen cuentas automáticamente en el servidor usando la mayoría de los clientes de mensajería. No afecta la habilidad de crear nuevas cuentas a través de esta interfase web. Los administradores pueden querer deshabilitar esta función para que los usuarios se tengan que registrar por otros medios (p.e. enviando solicitudes al administrador o mediante una interfase web propietaria).

☒ **Habilitado** - Los usuarios pueden crear nuevas cuentas automáticamente.

☐ **Deshabilitado** - Los usuarios no pueden crear nuevas cuentas automáticamente.

Registration Signing Forms

Signature forms (XEP-0348) describe the mechanism by which forms can be signed using other credentials. This is used in conjunction with In-Band Registration, in which special credentials are created with named accounts to create new accounts with XMPP, with a limit on the number of accounts that can be created. This method can be used by the manufacturers of devices for Internet of Things, so that the devices can create accounts automatically on the XMPP servers in an orderly manner, and customers can manage and control their accounts created separately. It also offers a mechanism by which the operators of the server can create who is responsible for the creation of the account and to what extent.

☒ **Habilitado** - The users can create new identity only with OAuth credentials, that is, only new identities can be created through an associated key and password.

☐ **Deshabilitado** - The users can create new identity without OAuth credentials.

Consumer Key:

Consumer Secret: [Generate credentials](#)

Authorized creations:

[Guardar credencial](#)

Consumer Key	Consumer Secret	Authorized creations	Used creations
0c7d849f5f6f83e17f474d77cd824bbc8192fb564563dfe4fce8804c76fa441	*****	10000	0
5d5f59018fec88585c72af22fe2ada3a523d81e0edbf4e4502ea13b59c9367	*****	300	0
9accdca6dba3752b156cb7bd2dee4d098b852c104189435f6ba67bda74f51	*****	100	12
a177c5a3ba45496f5f5e34e3ae00ad0098b852c104189435f6ba67bda74f51	*****	50	0
a442a9e5c1f09f63756ecc3a3e66c4dc88c3c3ad53c0d9bb0bcbaa77174f6	*****	100	4
b765a85c2ad866d9c40356c970cdd43e56c186d5f68fad3b0556fe256c4d8337	*****	30	0

Figure 5: Openfire Consola Web

Por ultimo en Openfire, se agregaron tablas a la base de datos, esto se puede apreciar en la Figura 6, cabe que mencionar que la manera de crear las relaciones es mediante los indxs, y no por relaciones SQL propiamente dicho. La tabla agregada es “ofOAuth” para los fines del proyecto.

quedo acorde al diagrama ilustrado en la Figura 9.

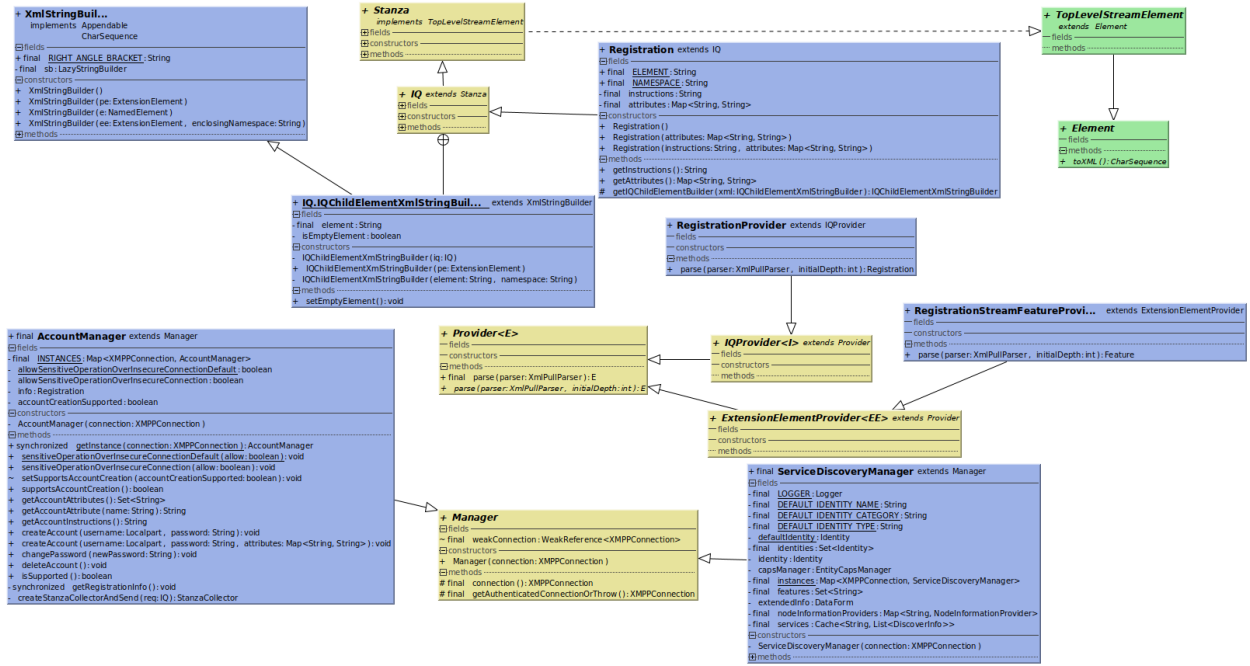


Figure 8: SMACK Diagrama de clases previo a la intervención de la librería.

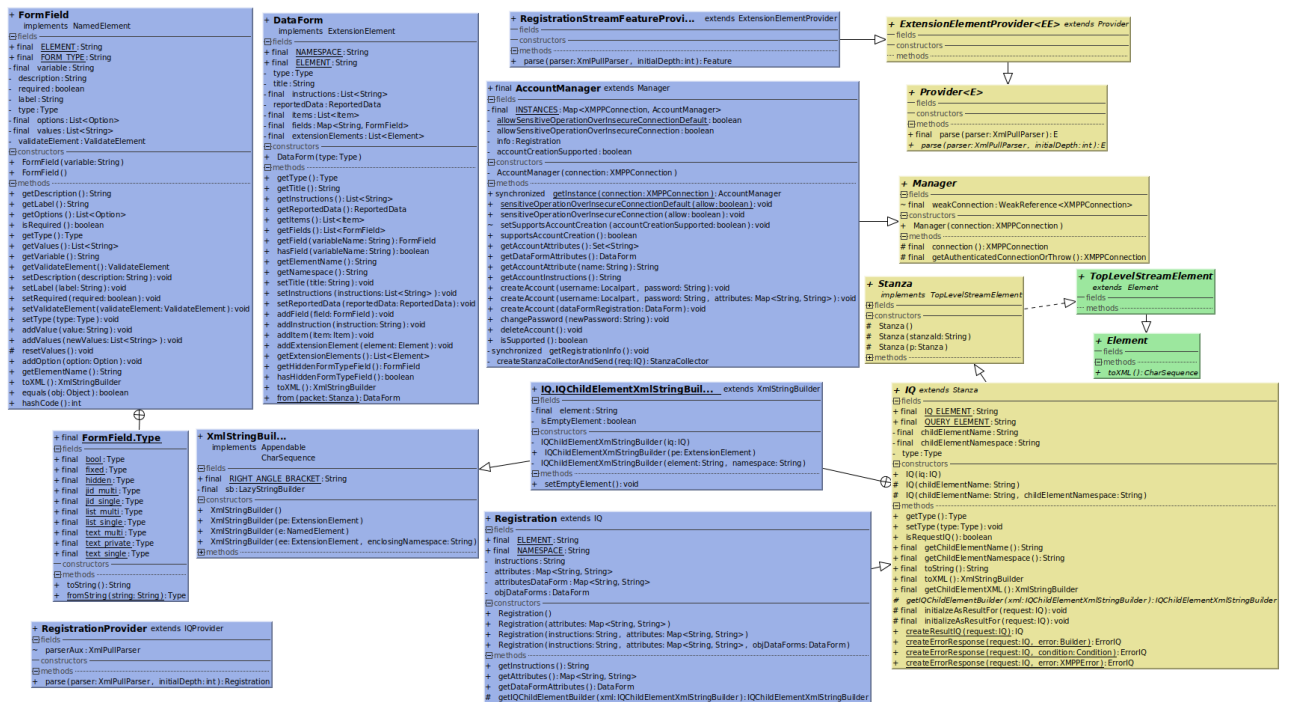


Figure 9: SMMACK Diagrama de clases posterior a la intervención de la librería.

Finalmente para depurar, en su conjunto, el sistema XMPP-IoT se creo un cliente usando la librería SMACK modificada, por ende, se incorpo las credenciales de consumidor. El cliente funciona como una Cosa que posee sensores de humedad, temperatura, gas, luz ambiente, movimiento y actual ores: interruptores de luces y un display LED de 16x2, mediante el cual veremos información relevante de nuestro dispositivo. El programa

```

<iq type='get'
  from='xmpp.binarylamp.cl'
  to='device@xmpp.binarylamp.cl'
  id='disco1'>
  <query xmlns='http://jabber.org/protocol/disco#info' />
</iq>

```

Figure 10: Consulta de Features soportadas de cliente al servidor

```

<iq type='get'
  from='xmpp.binarylamp.cl'
  to='device@xmpp.binarylamp.cl'
  id='disco1'>
  <query xmlns='http://jabber.org/protocol/disco#info' />
</iq>

```

Figure 11: Respuesta de Features soportadas del Servidor al cliente

establece la conexión con el servidor Openfire, luego envía la stanzas para crear automáticamente una identidad en este, utilizando el registro en-banda en conjunto del mecanismo de firmado de formularios. El servidor Openfire, el que debido a las añadiduras esta facultado para responder la solicitud, responde a esta, para finalmente crear la identidad en la red XMPP-Iot. Los detalles de las stanzas enviados en esta conversación entre el cliente y el servidor, se pueden apreciar en las Figuras continuación.

En el momento que se realiza la conexión con el servidor, el cliente XMPP se debe asegurar que el servidor soporte el registro en-banda utilizando formularios firmados, para eso el cliente envía la stanza en la Figura 10 y el servidor debe responder las “Features” que soporte, la que anuncia el soporte del XEP-0348, es “urn:xmpp:xdata:signature:oauth1”. En la Figura 11 se aprecia la stanza en donde el servidor responde todas las “Features” que soporta.

El proceso de creación de creación de identidades también se puede entender, por medio, del siguiente diagrama de secuencia en la Figura 12.

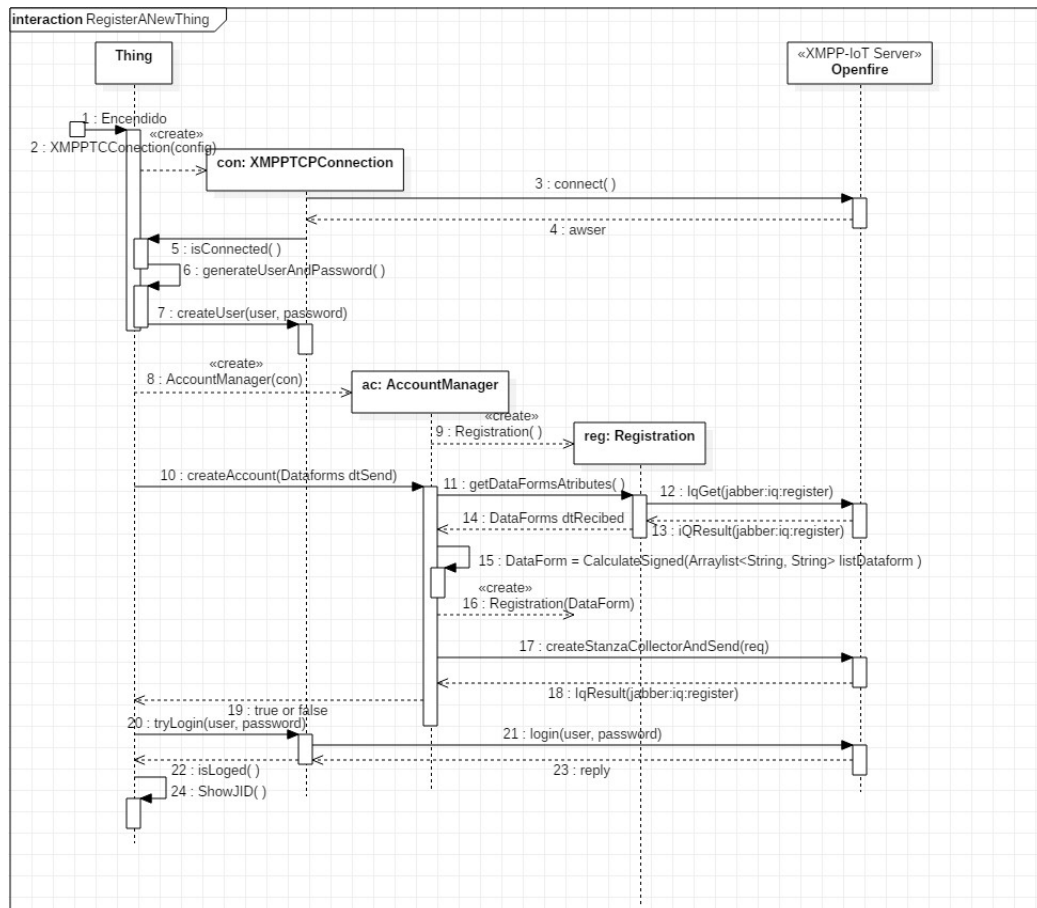


Figure 12: Registro de una nueva identidad con usando SMACK modificada en un cliente Thing.

También el diagrama de interacción en el momento que un usuario solicita algún valor en particular, mediante chat al sensor, esto se puede apreciar en la Figura 13.

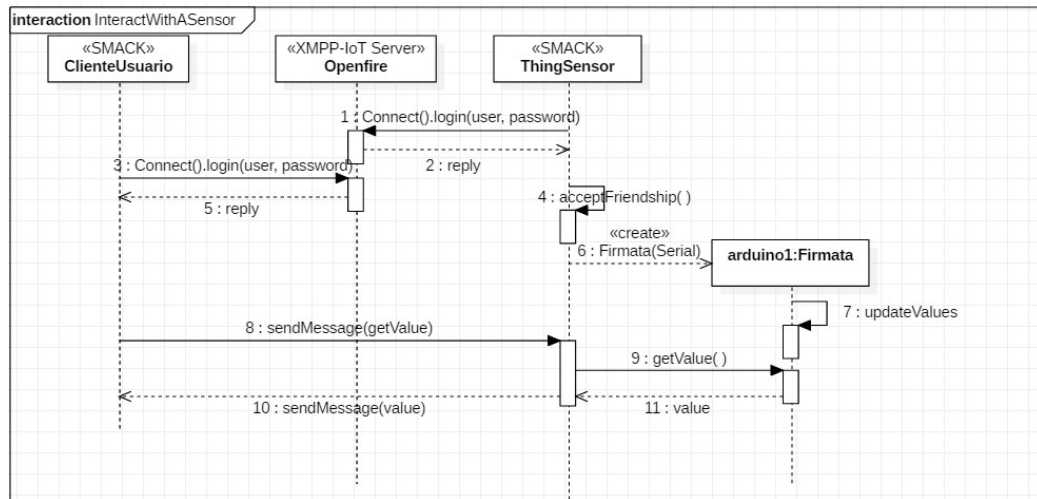


Figure 13: Solicitud del valor de un Sensor.

De forma similar, se pueden escribir valores en un actuador, este proceso para realizar esto, se encuentra diagramado en la Figura 14.

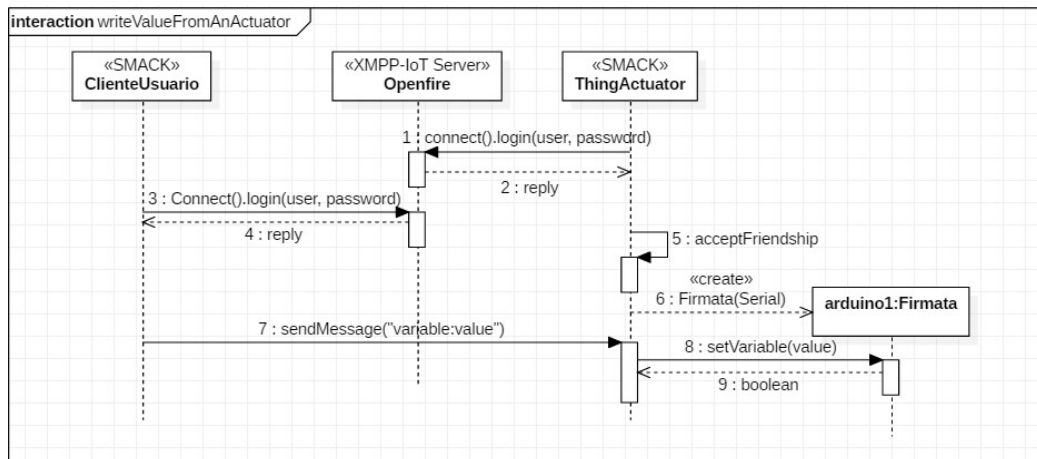


Figure 14: Escritura del valor en un actuador.

Con lo mencionado hasta el momento se concluye el desarrollo la mejora a la plataforma XMPP-IoT mediante la implementación de registro en-banda mediante firma de formularios. Sin embargo, queda un tema pendiente, la seguridad, ya que cuando hablamos de seguridad mediante la delegación de responsabilidades, que en el caso de una red XMPP-IoT nos referimos a la delegación de autorización para la creación de identidades, eso lo hacemos mediante el mecanismo implementado con la XEP-0348: Signing Forms. De nada sirve que un posea servidor esta característica, si la identidad del servidor puede ser fácilmente adulterada, es decir, que en el momento que un cliente XMPP se intenta conectar al servidor correspondiente, un oponente puede realizar un ataque “Man in the Middle”, para suplantar la identidad de nuestro servidor, abriendo así la posibilidad de sustraer información sensible. Para minimizar el riesgo, una opción razonable es utilizar SSL/TLS, pero debido a los agujeros de seguridad en SSL producidos por la corta proyección al momento del diseño de este protocolo, lo deja vulnerable en todas sus versiones, desde la 1.0 a la 3.0. Debido a esta razón la mejor opción es usar TLS, y no cualquier versión, ya que también hay vulnerabilidades conocidas, las cuales con un mediano poder de cómputo, pueden ser explotadas. La versión que utilizaremos será TLS 1.2 o la versión 1.1.

TLS se puede dividir en 3 grandes fases o etapas, las cuales consisten en:

1. Negociación de que algoritmo criptográfico se utilizará en la comunicación.
2. Intercambio de llaves publicas y la autenticación basada en certificados difitales.

3. Se cifra con criptografía simétrica el tráfico.

Security Level Cuando nos referimos a de algoritmos criptográficos, cabe mencionar que cada uno proporciona diversos “puntos fuertes” de seguridad, que están en función de qué tipo algoritmo y que tamaño de clave que este utilice. Los Security Levels o Bits of Security son usados para estimar la fortaleza del cifrado para proteger la información sensible, en función de fortaleza potencial histórico del adversario. Es decir, permite cuantificar cuan fuerte debe ser un algoritmo para ser catalogado como “algoritmo criptográficamente fuerte”. Generalmente está basado en el “mejor” ataque conocido sobre el algoritmo. Esto implica que cada vez el ataque avanza, también lo debe hacer el algoritmo y/o su clave.

Para cada algoritmo criptográfico existen diversas publicaciones que proporcionan recomendaciones y fórmulas para aproximarse al tamaño de clave adecuado para cada algoritmo. Si bien, estos estudios están disponibles, aún sigue siendo una tarea sumamente engorrosa seleccionar un algoritmo y tamaño de clave adecuado, ya que, para esto se debe leer y comprender todos estos documentos. Como una solución ante tal mar de información, nace keylength.com una web que pretende simplificar tal labor dando un resumen con los tamaños de clave recomendados basados en los Security Level. En donde uno de los factores discriminantes es el Bits of Security (BOS), a continuación, se puede ver dos tablas comparativa de los algoritmos comúnmente usados con sus respectivos valores. La Tabla 8 y la Tabla 7

Protección	Simetrica	Factoring Modulus	Logaritmo discreto Llave Grupo	Curva Elíptica	Hash
Nivel estándar legado No debería ser usado en sistemas nuevos	80	1024	160 1024	160	160
Protección a corto plazo Seguridad durante al menos diez años (2018-2028)	128	3072	256 3072	256	256
Protección a largo plazo Seguridad de treinta a cincuenta años (2018-2068)	256	15360	512 15360	512	512

Table 7: Recomendaciones Criptográficas ECRYPT, año 2018.

Fecha	Security Strength	Algoritmos Simétricos	Factoring Modulus	Log. Llave	Discreto Grupo	Curva Elíptica	Hash (A)	Hash (B)
(Legado)	80	2TDEA	1024	160	1024	160	SHA-1	
2016 - 2030	112	3TDEA	2048	224	2048	224	SHA-224 SHA-512/224 SHA3-224	
2016 - 2030 & después	128	AES-128	3072	256	3072	256	SHA-256 SHA-512/256 SHA3-256	SHA-1
2016 - 2030 & después	192	AES-192	7680	384	7680	384	SHA-384 SHA3-384	SHA-224 SHA-512/224
2016 - 2030 & después	256	AES-256	15360	512	15360	512	SHA-512 SHA3-512	SHA-256 SHA-512/256 SHA-384 SHA-512 SHA3-512

Table 8: Recomendaciones Criptográficas NIST, año 2016.

Luego del análisis de los algoritmos y sus respectivas claves, considerando a un oponente con alto poder de computo. Establecimos como protocolo criptográfico TLS Curva eliptica Diffie-Hellman RSA con AES-256_SHA128-256 y como método de hashing en el firma de OAuth en HMAC_SHA256, en el primer caso en conjunto tiene un Security Strength de 256, al igual que OAuth, superando lo recomendado para la actualidad según el NIST [4] y ECRYPT CSA de la Unión Europea [6] siendo ambas organizaciones posibles rivales, para tenerlos en consideración en caso de tener información sensible, según el criterio de ellos. En resumen, los metodos criptograficos que esta protegida la plataforma XMPP-IoT, lo podemos ver en las Tablas 9 y 10.

Protocolos	
SSLv2	No
SSLv3	No
TLSv1	No
TLSv1.1	Sí
TLSv1.2	Sí

Table 9: Protocolos utilizados.

Conjunto de cifrado	Bitsize	Confidencialidad directa	Infomación
ECDHE-RSA-AES256-GCM-SHA384	256	Sí	Curva: sect571r1
ECDHE-RSA-AES256-SHA384	256	Sí	Curva: sect571r1
ECDHE-RSA-AES256-SHA	256	Sí	Curva: sect571r1

Table 10: Métodos de cifrado utilizados.

De igual manera, se configuro las entradas DNS SRV[2], para la comunicación entre servidores y clientes fuese fiable, mediante lo siguientes entradas DNS:

- `_xmpp-client._tcp.domain.tld. TTL IN SRV priority weight port target`
- `_xmpp-server._tcp.domain.tld. TTL IN SRV priority weight port target`

Realizadas todas estas acotaciones en cuanto a seguridad, podemos decir que en cuanto al estado del arte del contexto criptográfico el sistema se encuentra a la vanguardia.

3 Validación

Tal como se ve en la Figura 1 el experimento consta un servidor XMPP *Openfire* modificado con varios clientes XMPP *SMACK* modificados, los cuales son:

- un Cliente XMPP Thing-Sensor (TS-1) que posee un sensor de humedad y temperatura
- un Cliente XMPP Thing-Sensor (TS-2) que posee un sensor de gas,
- un Cliente XMPP Thing-Controlador (TC-1)

Ya realizada las modificaciones, el procedimiento es el siguiente:

1. Ingresar a la consola de administración web de *Openfire*.
2. Ingresar a la sección de registro.
3. Generar Credenciales de Consumidor, es decir, *ConsumerKey*, *Consumer Secret*.
4. Ingresar 3 como cantidad de creación de identidades permitida al *ConsumerKey*.

```

C:\Windows\system32\cmd.exe
Conectando...
...
Conexión exitosa
Creando identidad en red XMPP...
Creación exitosa. Detalles en ts-01.properties
Ingresando
Ingreso exitoso
Temp.: 19.2 C \t Hum.: 27.8
IN : Solicitud MomentaryTemp
OUT: Message: Temp: 19.2 C
IN : Solicitud MomentaryHum
OUT: Message: Hum:27.5

C:\Windows\system32\cmd.exe
Conectando... exito
Creando identidad en red XMPP...
Creacion exitosa. Detalles en ts-02.properties
Ingresando... exito
Gas.: 0.0
IN : Solicitud MomentaryGas
OUT: Message: Hum:0.0

```

(a) Thing-Sensor 1

(b) Thing-Sensor 2

Figure 15: Retorno por consola de los clientes.

Para los clientes ingresamos el *ConsumerKey* y el *ConsumerSecret*, para que de este modo, se realice la conexión con el servidor exista el intercambio de Stanzas con dataforms [5] siendo exitosa la verificación de

Signatures este modo, se le delegue al cliente la facultad de crear identidades en el servidor XMPP Openfire. Así con el cliente TC-1 pudimos realizar consultas de temperatura, humedad al cliente TS-1 y al gas a TS-2. Realizado esto, se creo y ejecutó el Bot malicioso en ambos servidores, en el sin modificar como también en el modificado. Los resultados se pueden apreciar en las siguientes Tablas:

Table 11: Test de penetración a servidor Openfire sin implementación

Tiempo (sec)	Intentos de Creación por segundo	Identidades creadas exitosamente
50	2	100
100	2	200
300	3	900

Table 12: Test de penetración a servidor Openfire implementado el XEP-0348

Tiempo (sec)	Intentos de Creación por segundo	Identidades creadas exitosamente
50	2	0
100	2	0
300	3	0

4 Conclusiones y Trabajos Futuros

Este trabajo brinda un aporte a la comunidad de Internet de las Cosas dado que extiende un servidor libre, de código abierto y apoyado por una comunidad que lo extiende en el tiempo. Tal como se vio en el experimento, dada nuestra implementación es posible reducir el número de ataques de red en el servidor XMPP *OpenFire* que podría sufrir dada la implementación del Inband registración del XEP-0077 sin la implementación de signing forms.

Si bien, la brecha de seguridad presente en el XEP-0077 es abordada con la implementación del XEP-0348, quedan varios desafíos, uno de ellos es implementar un Plugin en Openfire para aprovisionamiento. Además, un Registro de Cosas que registre que permisos tiene una “cosa-cosa” y “humano-cosa”.

References

- [1] **Banks, A. & Gupta, R. (2014).** Mqtt version 3.1. 1. *OASIS standard*, 29.
- [2] **Consortium, I. S. (2000).** rfc-2782: A dns rr for specifying the location of services (dns srv). <https://tools.ietf.org/html/rfc2782>. [Online; accessed 02-July-2018].
- [3] **D. Scott Phoenix, D. W. K. M. C. B. X. Z. Y. H. A. (2017).** A generative vision model that trains with high data efficiency and breaks text-based captchas. <https://doi.org/10.1126/science.aag2612>. [Online; accessed 20-July-2018].
- [4] **E. Barker, A. R. (2015).** Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths. <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar1.pdf>. [Online; accessed 20-July-2018].
- [5] **Eatmon, R., Hildebrand, J., Miller, J., Muldowney, T., & Saint-Andre, P. (2006).** Jep-0004: Data forms. *online* Jan, 5.
- [6] **ECRYPT (2018).** Algorithms, key size and protocols report. <http://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf>. [Online; accessed 20-July-2018].
- [7] **Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999).** Hypertext transfer protocol-http/1.1. Technical report.
- [8] **Hammer-Lahav, E. (2010).** The oauth 1.0 protocol (2010). URL <https://tools.ietf.org/pdf/rfc5849.pdf>.
- [9] **IETF (2012).** Datagram transport layer security version 1.2. <https://tools.ietf.org/html/rfc6347>. [Online; accessed 20-July-2018].

- [10] **Internet Engineering Task Force (IETF), P. S.-A. (2011).** Rfc6120 extensible messaging and presence protocol (xmpp): Core.
- [11] **Lundgren, L. (2016).** LightWeight Protocol! Serious Equipment! Critical Implications! <https://media.defcon.org/DEF%20CON%2024/DEF%20CON%2024%20presentations/DEFCON-24-Lucas-Lundgren-Light-Weight%20Protocol-Critical-Implications.pdf>. [Online; accessed 19-July-2018].
- [12] **P. Saint-Andre D. Cridland (2015).** XEP-0001: XMPP Extension Protocols. <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar1.pdf>. [Online; accessed 20-July-2018].
- [13] **Saint-Andre, P. (2004).** Xep-0077: in-band registration. *Jabber Software Foundation*.
- [14] **Saint-Andre, P. (2011).** Extensible messaging and presence protocol (xmpp): Core.
- [15] **Shelby, Z., Hartke, K., & Bormann, C. (2014).** The constrained application protocol (coap).
- [16] **Waher, P. (2017).** Xep-0348: Signing forms.