Implementando un mecanismo contra telefonía en internet no solicitada basándose en identidad *Session Initiation Protocol* (SIP)[1]

Implementing an Anti-SPIT Mechanism Based on SIP Identity[2]

Implementando um mecanismo contra telefonia na internet não solicitada baseando-se em identidad *Session Initiation Protocol* (SIP)[3]

Octavio Salcedo-Parra[4]
Lilia Castellanos-Jaimes[5]
José Jairo Camacho-Vargas[6]

[1] Fecha de recepción: 19 de noviembre de 2010. Fecha de aceptación: 30 de mayo de 2011. Este artículo fue desarrollado por el grupo de investigación Internet Inteligente de la Universidad Distrital Francisco José de Caldas, Bogotá, Colombia.

[2] Submitted on: November 19, 2010. Accepted on: May 30, 2011. This article was developed by the research group on Intelligent Internet of the Universidad Distrital Francisco José de Caldas, Bogotá, Colombia.

[3] Data de recepção: 19 de novembro de 2010. Data de aceitação: 30 de maio de 2011. Este artigo foi desenvolvido pelo grupo de pesquisa Internet Inteligente da Universidad Distrital Francisco José de Caldas, Bogotá, Colômbia.

[4] Ingeniero de sistemas, Universidad Distrital, Bogotá, Colombia. Magíster en Teleinformática, Universidad de los Andes, Bogotá, Colombia. Decano de la Facultad de Ingeniería, docente de la Maestría en Ciencias de la Información y las Comunicaciones y director científico del Grupo Internet Inteligente, Universidad Distrital Francisco José de Caldas. Correo electrónico: osalcedo@udistrital.edu.co.

[5] Ingeniero de sistemas, Universidad Industrial de Santander, Bucaramanga, Colombia. Estudiante Maestría en Ciencias de la Información y las Comunicaciones, Universidad Distrital Francisco José de Caldas, Bogotá, Colombia. Integrante del Grupo Internet Inteligente, Universidad Distrital Francisco José de Caldas. Correo electrónico: lilia.castellanos@gmail.com.

[6] Ingeniero de Sistemas, Universidad Distrital Francisco José de Caldas, Bogotá, Colombia. Integrante del Grupo Internet Inteligente, Universidad Distrital Francisco José de Caldas. Correo electrónico: joscjai@gmail.com.

Resumen

En años recientes, el *Session Initiation Protocol* (SIP) se ha convertido en un importante protocolo de señalización para VoIP, dada su potencialidad para realizar llamadas multimedia desde internet, luego de TCP, IP y HTTP; sin embargo, SIP es un protocolo flexible, por su capacidad para extender métodos y atributos. Actualmente, los proveedores de servicios de voz sobre redes IP VoIP son sistemas cerrados. Por eso, el *Spam Over Internet Telephony* (SPIT) como VoIP no solicitado se ha visto como un problema poco grave hoy en día. Desde una perspectiva reducida, se plantea que el SPIT es problema de saber quién se va a comunicar con quién y el SIP Identity como una solución, dado su potencial para manejar las identidades dentro de SIP. JAIN SIP API es uno de los API más robustos para SIP sobre Java; por eso se sugiere como herramienta para el desarrollo de *software* para SIP, dadas las bondades de Java como la independencia de plataforma, movilidad, entre otras.

Palabras clave

SIP (protocolo de redes de computadores), protocolos de redes de computadores, telefonía por internet.

Abstract

Session Initiation Protocol (SIP) has become an important VoIP signaling protocol for its advantages in making Internet-based multimedia calls. The flexibility of the SIP protocol is due to its capacity to extend methods and attributes. In present, Ip VoIP voice service providers are non-open systems. This is why the Spam Over Internet Telephony (SPIT) does not currently pose any problem. Hence, SIP Identity is a solution for handling identities within SIP. In order to develop SIP software packages, JAIN SIP API, one of the most robust API's for Java-based SIP is used, thanks to advantages such as platform independence and mobility.

Key words

Session Initiation Protocol (Computer network protocol), computer network protocols, internet telephony.

Resumo

Nos últimos anos, o *Session Initiation Protocol* (SIP) converteu-se em um importante protocolo de sinalização para VoIP, dada seu potencialidade para realizar ligações multimídia a partir da internet, depois do TCP, IP e HTTP; contudo, o SIP é um protocolo flexível, pela sua capacidade para estender métodos e atributos. Atualmente, os provedores de serviços de voz sobre redes IP VoIP são sistemas fechados. Por isso, o *Spam Over Internet Telephony* (SPIT) como VoIP não solicitado tem sido visto como um problema pouco grave hoje em dia. Desde uma perspectiva reduzida, se propõe que no SPIT o problema é saber quem vai comunicar-se com quem e o SIP Identity como uma solução, dado seu potencial para gerenciar as identidades dentro do SIP. JAIN SIP API é um dos API mais robustos para SIP sobre Java; por isso é sugerida como ferramenta para o desenvolvimento de *software* para SIP, dadas as bondades de Java como a independência de plataforma, mobilidade, entre outras.

Palavras chave

SIP (Protocolo de redes de computadores), protocolos de redes de computadores, telefonia pela internet.

Introducción

El *Session Initiation Protocol* (SIP) como protocolo para iniciar, mantener y terminar una llamada es muy eficiente en cuanto a la cantidad de pasos que propone; sin embargo, los mecanismos existentes en SIP son inadecuados para asegurar criptográficamente la identidad del usuario final que origina las peticiones SIP, sobre todo en un escenario interdominios. Para ello SIP Identitiy define dos campos nuevos, para la cabecera SIP:

- *Identity*: usada para llevar la firma usada para validad la identidad.
- *e Identitiy-Info*: para llevar una referencia hacia el certificado del firmante (Peterson y Jennings, 2006).

Una identidad es definida como un U*niform resource Identifier* (URI) SIP, también llamada dirección de registro (*Address of Record {AoR}*), empleado para alcanzar a un usuario (por ejemplo, sip:jose@ud.com). La especificación de SIP no señala una forma para que el receptor de la petición SIP verifique que el campo de la cabecera *From* ha sido correctamente diligenciada en ausencia de algún tipo de autenticación criptográfica (Figura 1).

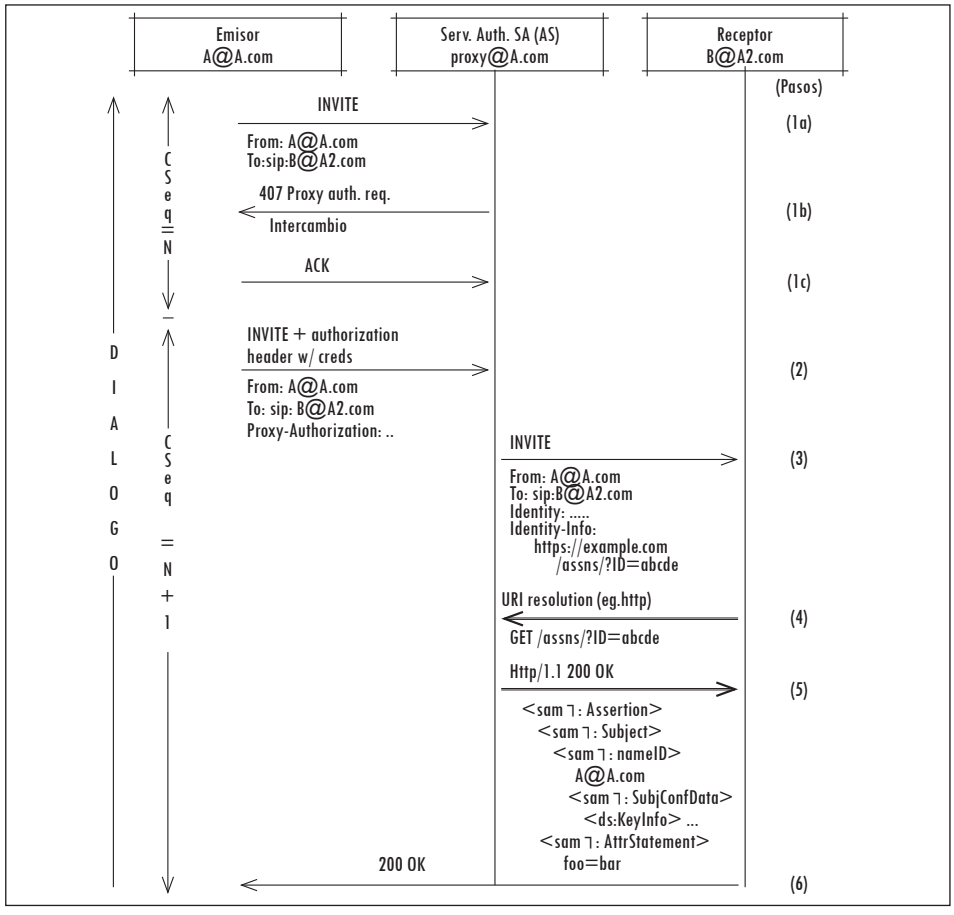
Figura 1. Cabecera SIP con adición de los campos *Identity* e *Identity-Info*

```
INVITE sip:bob@biloxi.example.org SIP/2.0
From: Alice <sip:alice@atralta.example.com>;tag=19481767
To: Bob <sip:bob@biloxi.example.org>
Call-ID: a84b4c76e66710
Cseq: 314159 INVITE
Contact: <spi:alice@pc33.atlanta.example.com>
Date: Thu, 21 Feb 2002 13:02:03 GMT
Identity:"A50h1tSWpbmXTyXJDhaCiHjT2xR2PAwBroi5Y8tdJ+CL3ziY72N3Y+lP8eoiXlr
        ZOuwboDicF9GGxA5vw2mCTUxcoXGoKJOhpBnzoXnuPNAZdcZEWsVOQAKj/ERsYR9
        BfxNPazWmJZjGmDoFDbUNamJRjiEPOKn13uAZIcuf9zM="
Identity-Info: https://biloxi.example.org/cert
```

Fuente: presentación propia de los autores

Actualmente son pocos los agentes de usuario (AU) que soportan certificados del usuario final, necesario para autenticarse entre ellos (por ejemplo, S/MIME); de igual forma, la autenticación *Digest* está limitada, en cuanto a que el emisor y el receptor deben compartir un secreto predeterminado (Franks et ál., 1999). El uso de muchas aplicaciones y servicios como SIP están regulados por políticas de autorización. Estas políticas pueden ser automatizadas o aplicadas por humanos: tal como en el celular aparece un identificador de la llamada. En SIP, *Caller-ID*. Una persona puede decidir si contesta o no. Automatizar dicha política sería un servicio que compare la identidad de los suscriptores potenciales contra una lista blanca (*WitheList*) antes de determinar si se acepta la llamada o no (Figura 2).

Figura 2. Servicio de autenticación AS, SIP SAML, perfil para obtener un atributo. Ejemplo: una transacción INVITE



Fuente: presentación propia de los autores.

El servicio de autenticación SIP SAML está basado en el URI y tiene la siguiente información requerida:

- Identificación: urn:ietf:params:sip:sip-saml-profile:as:uri:attr:1.0.
- Como todo URN debe estar registrado con la Internet Assigned Numbers Authority (IANA).
- Información del contacto: este incluye la información del contacto.
- Identificadores de los métodos de confirmación SAML: se toma de la especificación SAML 2.0.
- Descripción: descripción del perfil.

A continuación se describen los pasos ilustrados en la imagen anterior:

- Paso1. Transacción inicial entre el emisor y SA.
- Paso 2. El emisor envía un mensaje de petición SIP con credenciales de autorización al SA.
- Paso 3. El SA autoriza la petición SIP y la reenvía al receptor.
- Paso 4. El receptor de referencia URI SAML basado en HTTP.
- Paso 5. El SA retorna una aserción SAML.
- Paso 6. El receptor devuelve un 200 OK al emisor.

1. Comportamiento para el servicio de autenticación

Primero. Se debe extraer la identidad del emisor de la petición. El servicio de autenticación toma dicho valor del campo de la cabecera *From*. Esta *Address of Record* (AoR) será tomada como el campo de identidad (*identitiy field*). Si este campo contiene un URI SIP o SIPS, el servicio de autenticación debe extraer la porción del *host* del campo de identidad y compararlo con el del dominio, de lo cual está encargado (Sección 16.4 del RFC3261). Si el servicio de autenticación no está encargado de la identidad en cuestión, este debería procesar y responder la petición normalmente, pero no debe agregar una cabecera de identidad.

Segundo. El servicio de autenticación debe determinar si el emisor de la petición está autorizado para reclamar la identidad dada en el campo de identidad. Para llevar a cabo lo anterior, el servicio debe autenticar al emisor del mensaje. Por ejemplo: si el servicio de autenticación es instado por un intermediario SIP (*proxy*), este puede intercambiar la petición con una respuesta 407 usando el esquema de autenticación *Digest* (Franks et ál., 1999). Igualmente puede revisar la cabecera *Proxy-Authentication* enviada en la petición, que fue enviada antes del intercambio, usando credenciales en caché (sección 22.3 RFC3261). Si el

servicio de autenticación es instado por un AU SIP, se puede decir que autentica su usuario en cuanto a que el usuario puede otorgarle al AU la llave privada del dominio o una contraseña de desbloqueo.

Tercero. El servicio de autenticación debería asegurar que alguna cabecera de fecha *Date* en la petición sea correcta. Políticas locales pueden dictaminar cómo se debe hacer la precisión de este campo. El RFC3261 recomienda una discrepancia de máximo diez minutos, para asegurar que la petición no sobrepase ningún verificador. Si esta cabecera contiene una hora diferente, mayor a diez minutos de la hora actual (según el servicio de autenticación), el servicio debe rechazar la petición. Finalmente, se debe verificar que la cabecera de fecha cuadre dentro de los períodos de validez del certificado.

Cuarto. El servicio de autenticación debe hacer la firma de identidad y agregar la cabecera de identidad a la petición que contiene la firma. Luego se agrega la cabecera *Identitiy-Info*, la cual contiene un URI donde se puede adquirir el certificado. Finalmente, el servicio de autenticación debe reenviar el mensaje normalmente.

2. Comportamiento de un verificador

Este puede ser instanciado por un AU o un *proxy*. Cuando un verificador recibe un mensaje SIP que contiene una cabecera de identidad, este debe revisar la firma y verificar la identidad del emisor del mensaje. Si dicha cabecera no está presente en una petición y se requiere, entonces se puede enviar una respuesta 428 (*Use Identity Header*). Para verificar la identidad el emisor del mensaje, un verificador deber seguir los siguientes pasos (Figura 3):

Primero. Adquirir los certificados del dominio firmante. Dado que el certificado del dominio usado para firmar el mensaje no es conocido previamente por el receptor, las entidades SIP deberían descubrir dicho certificado por medio de la cabecera *Identity Info*, a menos que se cuente con un servicio de búsqueda de certificados. Si el esquema del URI en la cabecera *Identity-Info* no puede referenciarse, entonces se puede enviar un mensaje 436 *Bad Identity-Info*. El cliente procesa este certificado de formas usuales, incluyendo el chequeo de que no haya expirado, de que la cadena valida hacia una entidad certificadora de confianza y que esta no aparece en listas de revocación (certificados no válidos). Una vez el certificado es adquirido, este deber ser validado usando los procedimientos definidos en el RFC3280 (Rosenberg, 2006). Si el certificado no puede ser validado (está autofirmado y no es confiable o está firmado por una entidad

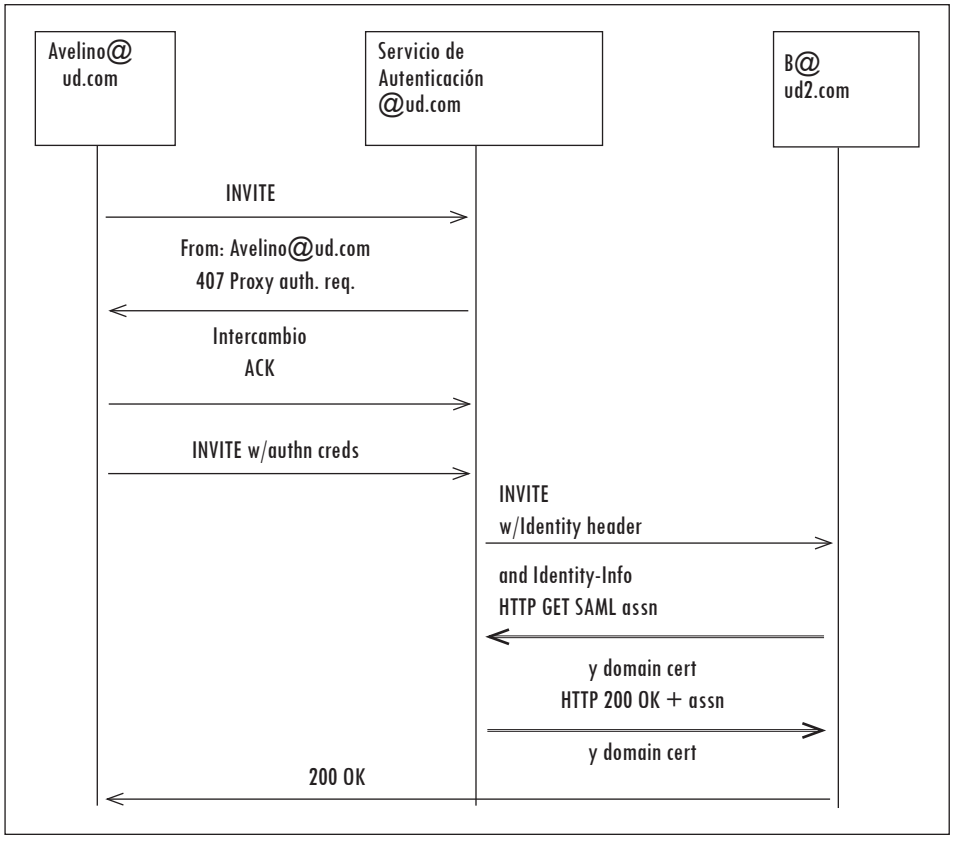
desconocida o no confiable, se venció o fue revocado), se debe enviar un mensaje 437 (*Unsupported Certificate*).

Segundo. El verificador debe determinar si el firmante está autorizado para el URI en el campo de la cabecera *From*.

Tercero. Se debe verificar la firma en el campo de la cabecera de identidad, siguiendo los procedimientos para generar una cadena *hash*. Si el verificador determina que la firma en el mensaje no corresponde con la reconstruida, entonces se envía un mensaje 438 (*Invalid Identity Header*).

Cuarto. Se deben validar las cabeceras de fecha, contacto e ID de la llamada (*date, contact* y *call-ID*). Así mismo, asegurar que el valor de la cabecera *date* encaje dentro del periodo de validez del certificado, cuya llave privada fue usada para firmar la cabecera de identidad.

Figura 3. Flujo de la identidad



Fuente: presentación propia de los autores

3. Java SIP-JAIN API

Actualmente, en el mercado se encuentran varias API para SIP versión 2 (Franks, 2009),la API JAIN SIP, que soporta las funcionalidades del RFC 3261 y las siguientes extensiones; el método INFO (RFC 2976), fiabilidad de las respuestas provisionales (RFC 3262); el Framework para la notificación de eventos (RFC 3265); el método *Update* (RFC 3311), el encabezado *Reason* (RFC 3326); el método *Message* (RFC 3428), definido para la mensajería instantánea, y el método REFER (RFC 3515) (Rosenberg, 2006).

Este paquete contiene las principales interfaces que modelan la arquitectura JAIN SIP desde la vista del desarrollador de aplicaciones y del vendedor:

Vista del desarrollador. Se trata de la implementación de la interfaz *SipListener*. Esta define los métodos requeridos por las aplicaciones para recibir y procesar mensajes de los vendedores SIP. Un *SipProvider* recibe mensajes de la red SIP, que encapsula dichos mensajes como eventos y los pasa a su *SipListener* registrado. Una aplicación debe registrarse con *SipProvider* para escuchar eventos dada la implementación de la interfaz *SipListener*. Una sola interfaz *SipListener* es obligatoria dentro de la arquitectura JAIN SIP.

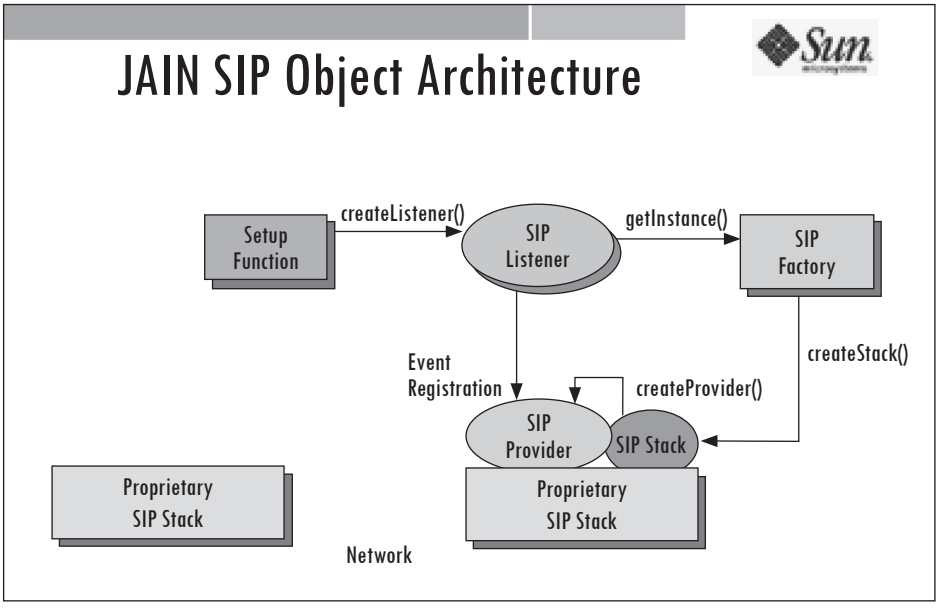
Vista de los vendedores. El vendedor implementa todas las interfaces en la especificación JAIN SIP, excluyendo la interfaz *SipListener*; sin embargo, las dos interfaces más importantes desde el punto de vista arquitectural son la *SipStack* y la *SipProvider*.

SipStack. Esta interfaz puede verse como una de gestión de la arquitectura JAIN SIP y solamente una puede existir por dirección IP. Esta interfaz encapsula las características de administración de SIP, como *ListeningPoints*, puntos de escucha que encapsulan el puerto y el transporte.

SipProvider. Esta interfaz puede verse como la de mensajería de la arquitectura JAIN SIP. Multiples *SipProviders* son permitidos dentro de la arquitectura. Esta interfaz define los métodos con los cuales se implementa una aplicación para que el *SipListener* se registre con el *SipProvider* y recibir peticiones entrantes y responderlas. Los métodos definidos para enviar mensajes SIP son también definidos dentro de la interfaz *SipProvider* (Saverio, 2008; Dean y Keith, 2009).

La implementación más popular que se tiene de esta API se puede encontrar en la misma página de JAIN. El proyecto se llama *sip-comunicator* (Helim y Ranga, 2002). Igualmente, en la página del National Institute of Standards and Technology (NIST) se puede encontrar el proyecto NIST-SIP, con bastante documentación y algunas herramientas (Figura 4).

Figura 4. Arquitectura JAIN SIP



Fuente: presentación propia de los autores.

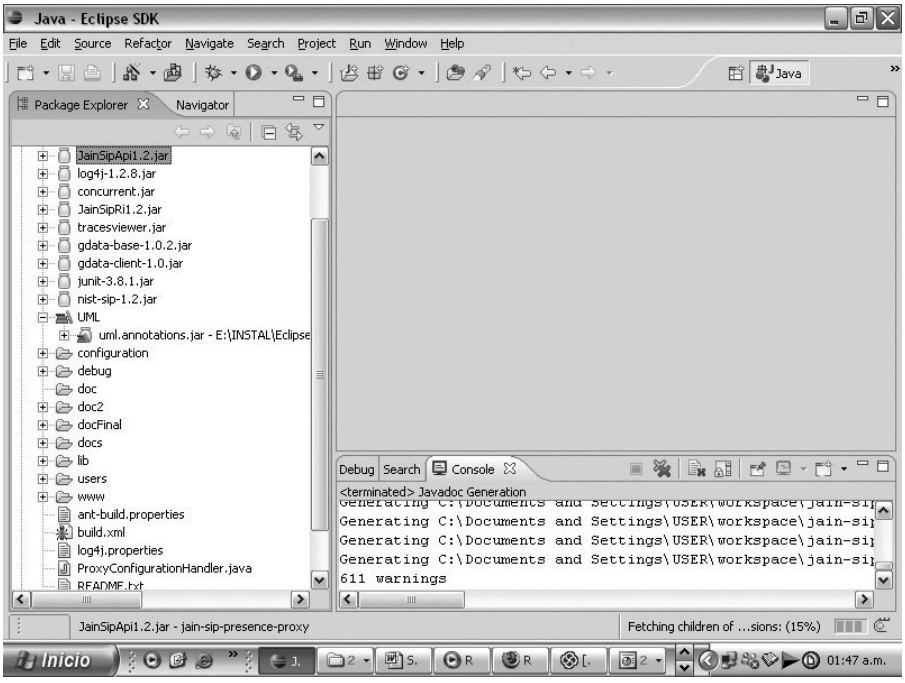
Básicamente, la JainSipAPI se encargaría de dos tareas: recibir peticiones y responder a ellas. Los dos comportamientos llevados a cabo por un UAS y un UAC, respectivamente. Con ello se dependería de los siguientes puntos:

- Crear la nueva cabecera.
- Crear los nuevos campos para la cabecera.
- Definir en qué puntos sería analizada e implementar el análisis.
- Implementar la firma de las cabeceras y su resumen dentro de la nueva cabecera.
- Implementar la verificación de las firmas.
- Crear los atributos de la llamada.
- Enviarlos dentro de la cabecera.
- Analizar los datos de la llamada.
- Aceptar o rechazar la llamada.

4. Desarrollo

Una vez configurado el entorno de programación, ya se puede iniciar el desarrollo (Figura 5).

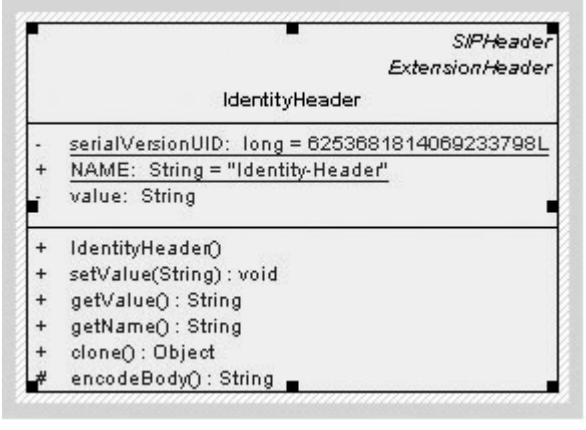
Figura 5. Desarrollando en eclipse



Fuente: presentación propia de los autores.

- Crear la nueva cabecera.
- Crear los nuevos campos para la cabecera (Figura 6).

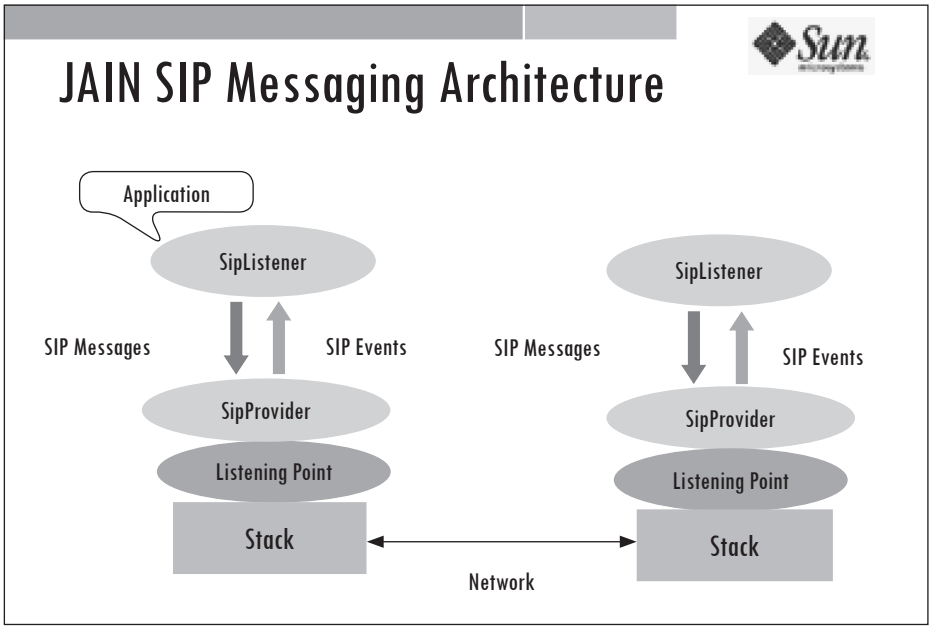
Figura 6. Clase *IdentityHeader*



Fuente: presentación propia de los autores.

- Definir en qué puntos sería analizada e implementar el análisis (Figura 7).

Figura 7. Arquitectura de los mensajes para JAIN SIP



Fuente: presentación propia de los autores.

En este punto, ya se sabe dónde se va a tocar el API para que también analice la nueva extensión. En el proveedor SIP, en el evento de registro. Justo entre el *listener* y el proveedor. Por consiguiente, lo que hace falta es ubicar esto dentro del código *wrapper* implementado.

- Implementar la firma de las cabeceras y su resumen dentro de la nueva cabecera. En este caso se trata del evento de inicio (*sign in*). Se firman las cabeceras del registro y se envían:

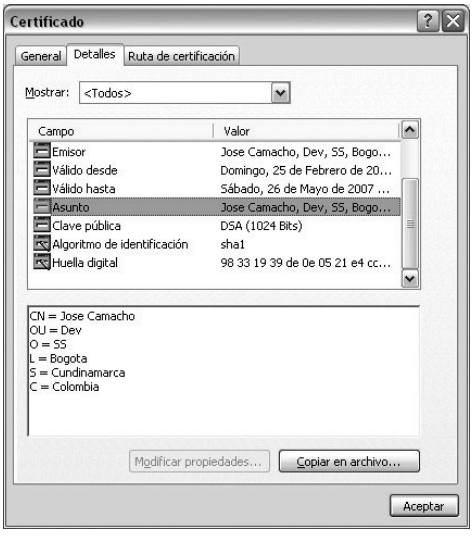
```
IdentityHeader identityHeaderL = new IdentityHeader();
identityHeaderL.setValue(signedHeaders);
SIPHeader identityHeader = (SIPHeader)headerFactory.createHeader("Identity",
signedHeaders);
request.setHeader(identityHeader);
SIPHeader identityHeader2 = (SIPHeader)request.getHeader("Identity");
ClientTransaction clientTransaction = sipProvider.getNewClientTransaction(request);
clientTransaction.sendRequest();
```

Se creó un certificado digital de ejemplo llamado *test.cer*. Este fue generado usando la herramienta *KeyTool*. Para autenticar a los usuario del dominio (figuras 8 a 11).

Figura 8. Certificado digital de prueba: pestaña 1

Fuente: presentación propia de los autores.

Figura 9. Certificado digital de prueba: pestaña 2

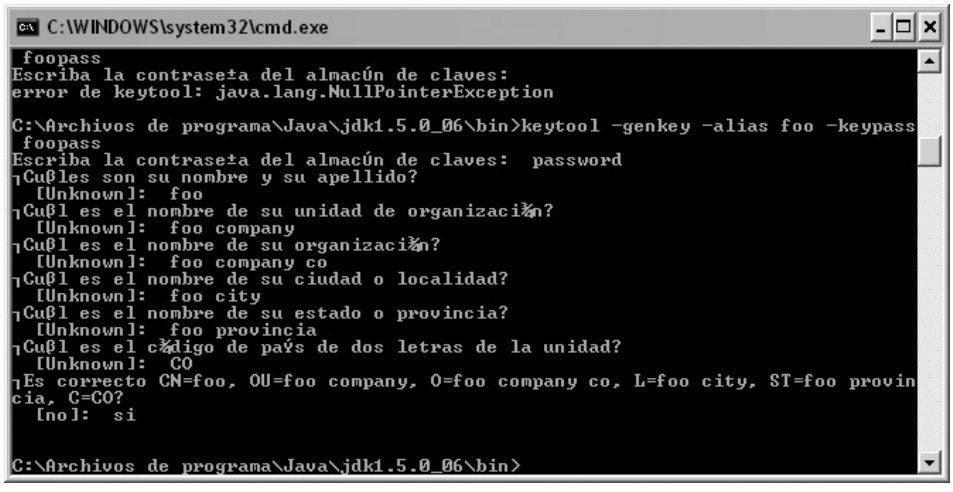


Fuente: presentación propia de los autores.

Figura 10. Certificado digital de prueba: pestaña 3

Fuente: presentación propia de los autores.

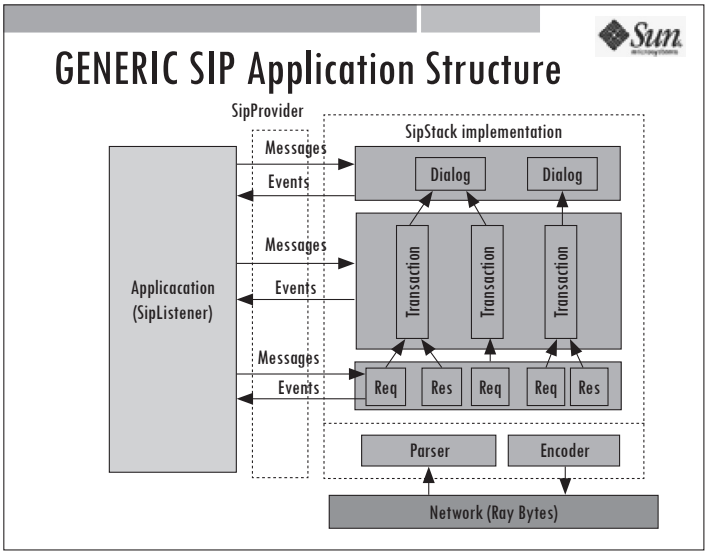
Figura 11. Creando un certificado digital con *KeyTool*



Fuente: presentación propia de los autores.

- Implementar la verificación de las firmas (Figura 12).

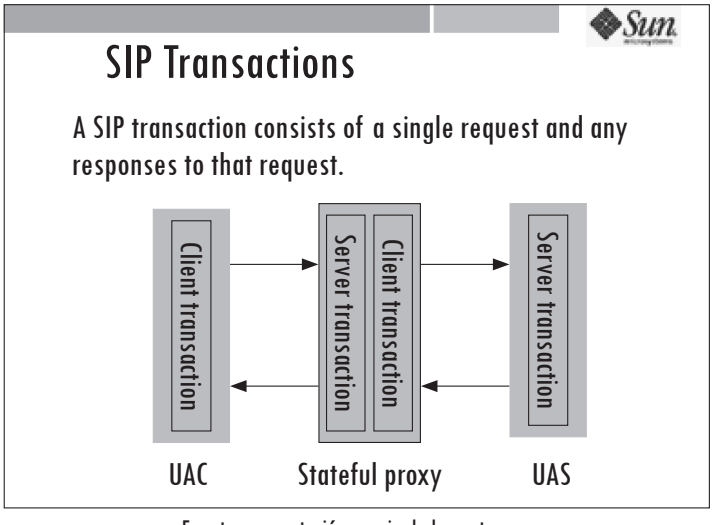
Figura 12. Estructura de una aplicación genérica SIP



Fuente: presentación propia de los autores.

La verificación se realiza en el *proxy* cuando procesa una respuesta. Al inicio puede verificar la nueva extensión y no aceptar la llamada si encuentra que las firmas no coinciden (Figura 13).

Figura 13. Transacciones SIP

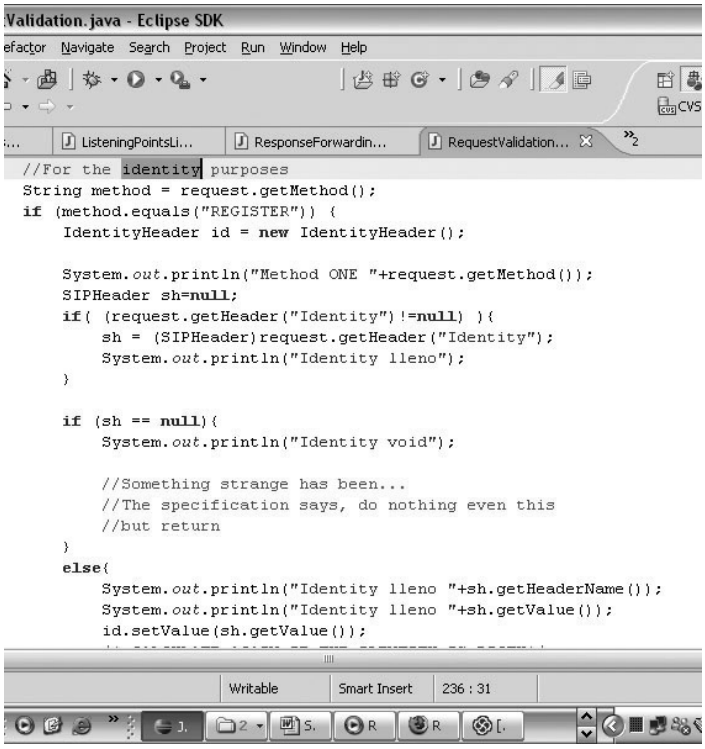


Fuente: presentación propia de los autores.

Con la verificación (figuras 14 y 15) se acabaría la transacción, y al agente le tocaría volver a intentar:

- Crear los atributos de la llamada.
- Enviarlos dentro de la cabecera.
- Analizar los datos de la llamada.
- Aceptar o rechazar la llamada (Figura 16).

Figura 14. Validación de cabeceras



Fuente: presentación propia de los autores.

Figura 15. Se verifican las firmas y se toma la decisión

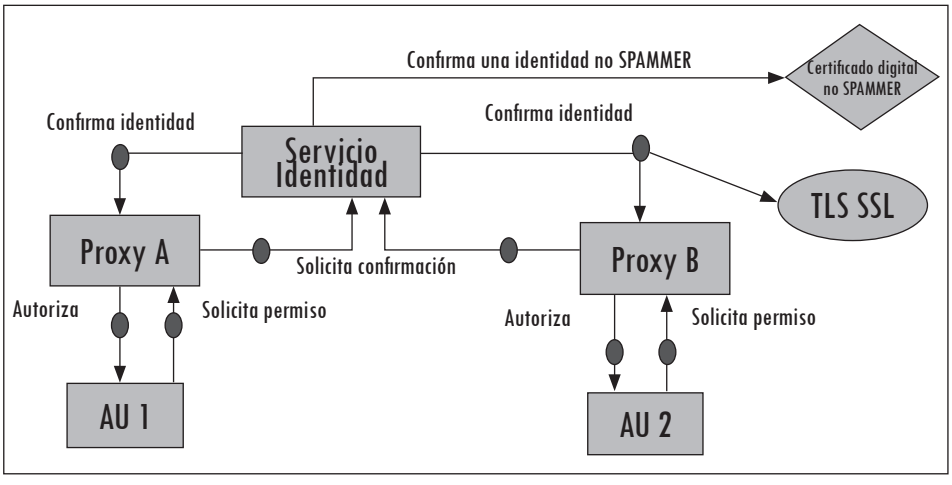
ListeningPointsLi... | ResponseForwardin... | RequestValidation...

```
GestorSeguridad gs = new GestorSeguridad();
DocumentoConcreto headers = new DocumentoConcreto();
headers.setContenido(headersToSign.toString());
System.out.println(headersToSign.toString());
String signedHeaders = null;
signedHeaders = gs.hacerSHA1(headers);
System.out.println("The signature = "+signedHeaders);

/* */
    //Finally, we evaluate the signature
    if ( signedHeaders.equals(sh.getValue()) ){
        System.out.println("OK identidad acertada");
        System.out.println("We get "+signedHeaders);
        System.out.println("We receive "+sh.getValue());
    }
    else{
        //we return errors.
        Response response = proxy.getMessageFactory().createResponse(
                Response.BAD_REQUEST, request);
        if (serverTransaction != null)
            serverTransaction.sendResponse(response);
```

Fuente: presentación propia de los autores.

Figura 16. Resumen de la propuesta



Fuente: presentación propia de los autores.

5. Resultados

Para llevar a cabo el desarrollo, se utilizó un equipo (Equipo 1) con las siguientes características: un procesador AMD 1800 + 1,6 GHZ 800 FSB; una memoria de 1 GB; red de 100 Mbps. Además, se conectó al Equipo 2 por un cable directo.

Para llevar a cabo las pruebas, el equipo anterior estuvo a cargo de un *proxy* y un agente; adicionalmente, se utilizó otro equipo (Equipo 2), encargado de uno de los agentes, con las siguientes características: procesador AMD K II 500 MHZ 100 FSB; memoria de 256 MB; red de 100 Mbps. Se conectó al Equipo 1 por un cable directo.

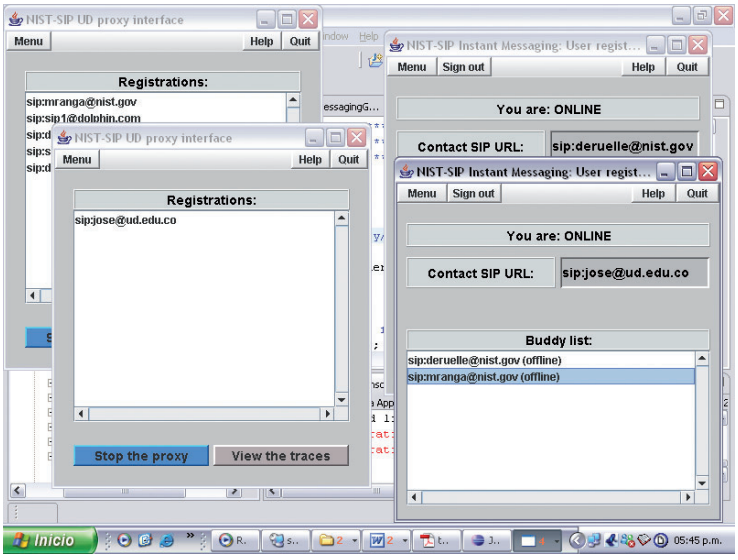
Finalmente, para realizar pruebas de carga, se utilizó un tercer equipo (Equipo 3) con otro *proxy* y un agente: procesador Intel Pentium 4 HT 3,2 GHZ-3,2 GHZ (que emula dos procesadores); memoria de 2 GB; red de 100 Mbps y 1 Gbps. Se conectó al Equipo 1 a través de internet (1Mbps Equipo 3 *vs.* 400 Kbps al Equipo 1).

Se observó que cada agente consume en promedio 28 MB de memoria RAM. El *proxy* consume la misma memoria RAM de un agente (estos datos fueron obtenidos usando la sentencia tasklist /FI "IMAGENAME eq java*, que retorna características como el consumo de memoria y CPU de cierto proceso en el SO WIN XP).

El gasto de procesador para los agentes es mínimo (10% a 15%). Incluso el *proxy* fue capaz de mantener hasta 100 llamadas sin pérdida de rendimiento. El mayor gasto se presentó en la red. Es interesante que con solo IM el gasto de la red aumentara en un 20%.

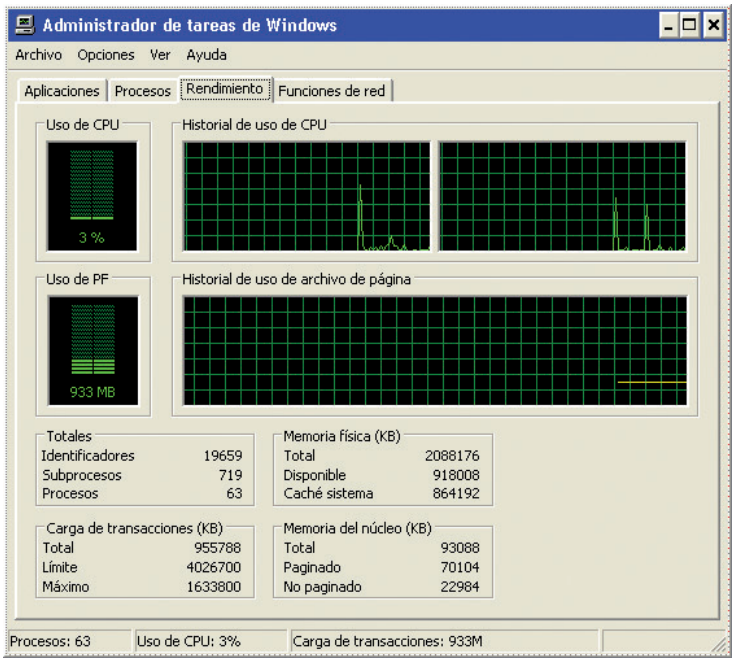
El método de prueba fue básicamente en los mensajes. El experimento consistió en un agente que disparó (inició) llamadas con diversas cuentas origen ficticias, durante doce horas, hacia un conjunto de direcciones bajo la administración de un *proxy* (Figura 17). Este último se mantuvo estable. Como se puede observar el consumo fue más de memoria que de CPU. Esto quizás a la cantidad de transacciones que tuvo que mantener, al igual que los registros, pues, como se sabe, no se usó ninguna base de datos (figuras 18 y 19).

Figura 17. Proxy y dos agentes



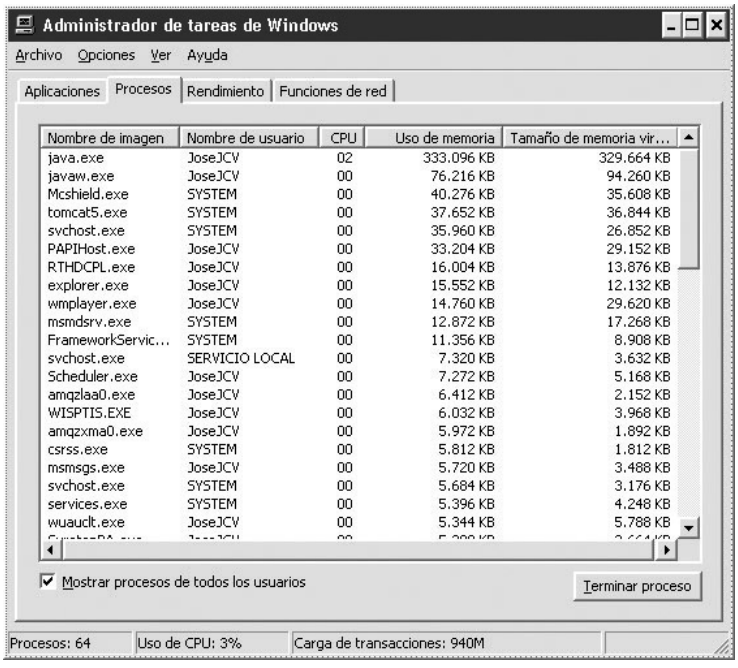
Fuente: presentación propia de los autores.

Figura 18. Gasto de memoria en 12 días acumulado



Fuente: presentación propia de los autores.

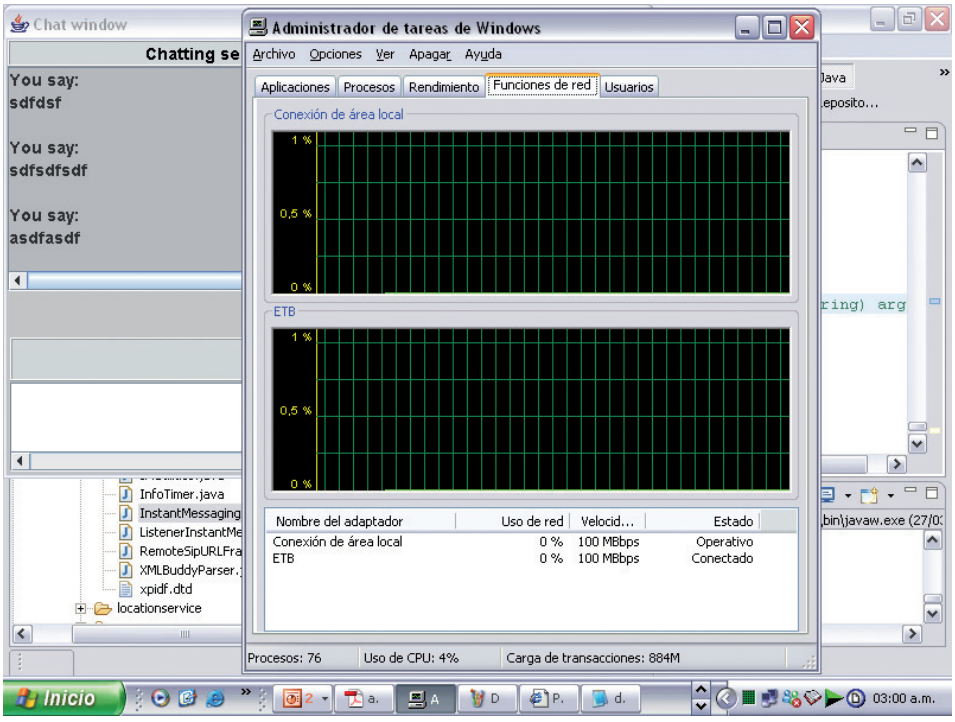
Figura 19. Gasto de memoria por parte de JVM



Fuente: presentación propia de los autores.

El consumo de ancho de banda es muy positivo: no consume un gran ancho. En promedio, envía paquetes de 4 KB. Lo anterior se puede verificar viendo el Log del *proxy*, pues este registra todos los paquetes y eventos. En las dos horas de funcionamiento continuo llegó a 8 MB (Figura 20).

Figura 20. Consumo de ancho de banda: se mantiene constante



Fuente: presentación propia de los autores.

Finalmente, en cuanto a SPIT, se generó un ciclo en el agente para que dispararan llamadas con bastante frecuencia, usando diferentes atributos, válidos e inválidos. El *proxy* fue muy rápido en responder, teniendo en cuenta que se usó un tan solo el 50% de CPU de un procesador de 7000 millones de instrucciones por segundo. Los autores se atreven a decir que para un sistema monodominio el uso de *SIP Identity* realmente es muy bueno, porque el *proxy* conoce a sus usuarios, conoce su propio certificado y, por ende, *SIP Identity* es fuerte autenticando usuarios en un solo dominio administrativo (referencia *strong authentication*).

Falta esperar los resultados del estudio llevado a cabo en http://www.spitprevention.net/ (Ivov, 2009), que busca implementar un filtro para el SPIT teniendo en cuenta un patrón para el comportamiento del SPIT, la identidad de los usuarios, retroalimentación y uso de SAML. Kayote Networks, quien entre el apoyo de varios investigadores, cuenta con Hannes Tschofenig (pionero en tratar temas sociales y de seguridad para SIP, en casos de emergencia, SPIT, entre otros) y Henning Schulzrinne (creador de SIP).

6. Conclusiones

Este artículo presenta una extensión al API JAIN SIP para que este tenga capacidad de utilizar la identidad SIP como mecanismo anti-SPIT. Se muestra la facilidad tecnológica para poder enviar SPIT; sin embargo, también la facilidad con la cual se podría evitar al implementar la solución propuesta.

La solución propuesta muestra cómo de manera sistemática usando *software* existente es posible crear herramientas anti-SPIT con una simple, mas no sencilla, extensión llamada identidad SIP. Este trabajo puede ser considerado uno de los primeros en el país, pues si bien hay VoIP, no se puede medir el potencial de SPIT en el país —que lidera la penetración de internet en América Latina (Falomi, Garroppo y Niccolini, 2007), lo cual lo hace atractivo para generar SPIT al aumentar los usuarios de internet— es importante generar herramientas como la producida con el presente trabajo.

Como trabajo futuro está la integración con otras técnicas anti-SPIT,y su perspectiva en Colombia.

Referencias

DEAN, W. y KEITH, D. *Capítulo de SIP en la IETF* [documento en línea] <http://www.ietf.org/html.charters/sip-charter.html> [Consulta: 10-07-2009].

FALOMI, M.; GARROPPO, R. y NICCOLINI, S. Simulation and optimization of SPIT detection frameworks. *Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE.* 26-30 Nov 2007, pp. 2156-216. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4411322&isnumber=4410910>.

FRANKS, J.; HALLAM-BAKER, P.; HOSTETLER, J.; LAWRENCE, S. y LEACH, P. *HTTP authentication. Basic and digest access authentication, RFC 2617* [documento en línea]. 1999. <http://www.ietf.org/rfc/rfc2617.txt> [Consulta: 09-07-2009].

HELIM, P. y RANGA, M. *Especificación JSR-000032 JAIN SIP API* [documento en línea]. 2002. Java Community Process. <http://www.jcp.org/aboutJava/communityprocess/maintenance/jsr032/> [Consulta: 08-03-2009].

IVOV, E. *Proyecto cliente VoIP para SIP, "Sip-communicator"* [documento en línea]. <http://www.jitsi.org/> [Consulta: 07-04-2009].

KAYOTE NETWORKS. *Empresa pionera en desarrollo de soluciones contra el SPIT*. [web en línea]. <http://www.kayote.com/web/About/Research.htm> [Consulta: 05-03-2008].

PETERSON, J. y JENNINGS, C. *Enhancements for authenticated identity management in the Session Initiation Protocol (SIP), RFC 4474* [documento en línea]. 2006. IETF. <http://tools.ietf.org/html/rfc4474> [Consulta: 09-07-2009].

RANGA, M. y HELIM M. *Proyecto JAIN SIP. Proyecto JAIN SIP* [web en línea] <http://jain-sip.dev.java.net/> [Consulta: 08-07-2009].

RENATA: Conectividad en Latinoamérica, Colombia lidera penetración en Internet [web en línea]. <http://200.31.94.218/index.php/noticias/5-noticias/1485-conectividad-en-latinoamerica-colombia-lidera-penetracion-en-internet.html> [Consulta: 23-01-2011].

ROSENBERG, J. *A Framework for Consent-Based Communications in the Session Initiation Protocol (SIP) draft-ietf-sipping-consent-framework-05* [documento en línea]. 2006. IETF. <http://www.tools.ietf.org/html/draft-ietf-sipping-consent-framework> [Consulta: 10-07-2009].

ROSENBERG, J. y JENNINGS, C. *The Session Initiation Protocol (SIP) and Spam.* [documento en línea]. 2007. IETF. <http://tools.ietf.org/html/draft-ietf-sipping-spam-05> [Consulta: 07-10-2009].

SAVERIO, N. SPIT prevention state of the art and research challenges [documento en línea]. 2008. *Third Annual VoIP Security Workshop*. <http://old.iptel.org/voipsecurity/doc/07%20-%20Niccolini%20-%20SPIT%20prevention%20state%20of%20the%20art%20and%20research%20challenges.pdf> [Consulta: 10-07-2009].

STEWART, L. *HTTP Authentication: Basic and Digest Access Authentication, RFC 2617*. <http://www.ietf.org/rfc/rfc2617.txt>.

UTILIDAD PARA CERTIFICADOS DIGITALES Y JAVA [web en línea]. <http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html> [Consulta: 06-05-2009].