Tiempo, Listas negras y Seguridad en SPL

Hugo Andrés López Andrés Aristizábal Camilo Rueda*
Frank D. Valencia**

7 de febrero de 2006

Research Report [1]

Resumen

Diversos avances concernientes a la seguridad dentro de las comunicaciones han utilizado nociones que permiten denotar tanto caducidad en los mensajes para sortear de manera efectiva una amplia gama de ataques, como el manejo del concepto de listas negras utilizadas para ubicar aquellos agentes de reputación dudosa con los cuales no se debe establecer ningún contacto para evitar contratiempos. Debido a la relevancia intrínseca de la inclusión de esta clase de elementos frente a temas de seguridad, se introducirán nociones de este tipo dentro de un cálculo de procesos de seguridad bien establecido denominado SPL, sin realizar cambios intrusivos dentro del lenguaje. De esta forma se permitiría modelar nuevos protocolos de comunicación seguros, posibilitando el razonamiento de propiedades de seguridad a través de las técnicas inherentes a SPL. Por último, se presentarán modelos especificados en este lenguaje de dos conocidos protocolos que hagan uso de las citadas nociones, de forma que se pueda verificar la validez de la inclusión de estos nuevos componentes.

1. Introducción

El uso de los cálculos de procesos en seguridad ha representado poderosas ventajas en los procesos de análisis, diseño y verificación de propiedades de comunicación, encontrando fallas en sistemas globalmente conocidos por su aparente seguridad. Este enfoque presenta sus fortalezas en el uso de teorías lógicas y matemáticas en las cuales fundamenta sus técnicas de razonamiento. Diversos enfoques se han propuesto en la literatura, desde cálculos de proceso no-lineales donde primitivas de seguridad se plantean como abstracciones basadas en conceptos como *canales seguros* [MPW89], hasta enfoques que contemplan modelos lineales de comunicación y primitivas básicas de criptografía

*Pontificia Universidad Javeriana - Cali
**LIX Ecole Polythecnique

[1]Este reporte fue preparado en el contexto del proyecto de investigación "Modelamiento de Problemas en Ciencia y Tecnología Usando Cálculos de Procesos Concurrentes", financiado por la Pontificia Universidad Javeriana - Cali

[AG99, Cra03]. De estos, un enfoque particular presenta interesantes características. El cálculo de procesos SPL [Cra03], se erige como un enfoque diferente al contar con una semántica operacional dual basada en redes de petri [Jen94], la cual facilita el razonamiento sobre las interacciones de los agentes en un protocolo de comunicación. Adicionalmente, plantea un conjunto de principios fundamentales de prueba, proporcionando un conjunto de formalismos en donde diversas propiedades de seguridad han podido ser exitosamente verificadas [CW01, CCM02, ALR05, AL06].

Pese a su poderosa semántica operacional, solo un pequeño conjunto de construcciones en concurrencia han sido estudiadas en este lenguaje. Este reporte plantea un estudio minucioso respecto a un conjunto particular de nociones necesarias en seguridad. Especificamente, se estudian la inclusión de marcas temporales (timestamps) y codificaciones basadas en tests negativos dentro del modelo subyacente a SPL.

El documento se encuentra estructurado de la siguiente manera: En la siguiente sección presentamos las nociones básicas de SPL, de *timestamps* y de *listas negras*. En las secciones 3 y 4 extenderemos la sintaxis básica de SPL a través de un conjunto de *encodings* de tal forma que se puedan incluir elementos que permitan manejar nociones de temporalidad y de *tests negativos*. Por último se presentará una breve discusión acerca de lo realizado.

2. Preliminares

En esta sección se presentará un breve repaso del cálculo de procesos para seguridad desarrollado por Winskel y Crazzolara [CW01] además de introducir importantes nociones para seguridad, como lo son los *timestamps* o marcas temporales [NS93, DS81] y las *blacklists* o listas negras [Cha85, Mic96].

2.1. Breve descripción de SPL

SPL [Cra03] es un cálculo de procesos diseñado tanto para modelar protocolos de comunicación, como para probar las propiedades de seguridad que deben cumplir cada uno de éstos. La verificación dentro de este lenguaje es realizada a través de su semántica operacional dual, que incluye una semántica de transición con la cual se analiza la evolución de los procesos dentro del protocolo, y una semántica basada en eventos por la cual se especifica de manera acertada la dependencia que existe entre los eventos de un proceso. Lo anterior permite razonar de manera adecuada y relativamente sencilla acerca de importantes propiedades de seguridad tales como confidencialidad y autenticación.

Debido a que SPL se encuentra basado en el modelo Dolev-Yao [DY81], incluye suposiciones importantes que permiten simplificar el razonamiento acerca de propiedades de seguridad dentro del protocolo a modelar. Dentro de éstas se pueden contar como las de mayor relevancia, aquellas que tratan acerca de la criptografía, en donde se indica que todo mensaje cifrado es inquebrantable si no se tiene la llave adecuada para desencriptarlo, y aquellas concernientes al atacante, en donde se indica a éste como un intruso activo que no sólo se dedica a filtrar información que fluye a través de la red, sino que puede ejercer diversas funciones con los datos que puede capturar en la red, como la replicación, supresión, etc.

Este cálculo de procesos concerniente a la seguridad, se encuentra definido operacionalmente en términos de configuraciones que contienen componentes de información (mensajes) que sólo pueden incrementarse durante su evolución, ya que éstos persisten por siempre, modelando de esta forma el hecho que en una red abierta un intruso puede ver y recordar cualquier mensaje que pudo estar en tránsito.

2.1.1. Sintaxis

Las entidades sintácticas presentes en SPL se describen de la siguiente manera:

- Un conjunto infinito de nombres N denotados por $n, m, ..., A, B, ...$ Nombres que oscilan entre *nonces* (valores únicos generados de manera randómica) [Per96]) y nombres sobre agentes.
- Tres tipos de variables sobre nombres (denotadas por $x, y, ..., X, Y, ...,$), sobre llaves ($\chi, \chi', \chi_1, ...,$) y sobre mensajes ($\psi, \psi', \psi_1, ...,$). Éstas variables igualmente pueden ser representadas como vectores $\vec{x}\vec{\chi}\vec{\psi}$ respectivamente.
- Un conjunto de procesos denotado por $P, Q, R, ...$

Variables sobre nombres	$x, y, ..., X, Y, ...,$
Variables sobre llaves	$\chi, \chi', \chi_1, ...,$
Variables sobre mensajes	ψ, ψ', ψ_1
Expresiones de nombres	$v ::= n, A, ... \mid x, X$
Expresiones de llaves	$k ::= Pub(v) \mid Priv(v) \mid Key(\vec{v}) \mid \chi, \chi', ...$
Mensajes	$M, M' ::= v \mid k \mid (M, M') \mid \{M\}_k \mid \psi, \psi', ...$
Procesos	$p ::= out\, new(\vec{x})\, M.p \mid in\, pat \vec{x}\vec{\chi}\vec{\psi}\, M.p \mid \parallel_{i \in I} P_i \mid !P$

Cuadro 1: Sintaxis de SPL

El resto de elementos incluidos dentro del conjunto sintáctico de SPL se encuentran definidos en la tabla 1, donde $Pub(v)$, $Priv(v)$ y $Key(\vec{v})$ denotan la generación de llaves públicas, privadas y compartidas respectivamente. Se utiliza la notación vectorial $\vec{s}$ para representar una lista de elementos, posiblemente vacía, $s_1, s_2, \ldots, s_n$.

Descripción intuitiva y Convenciones

De esta forma se presentan una serie de breves convenciones e intuiciones acerca de los procesos en SPL.

El proceso de salida $out\, new(\vec{x})\, M.p$ genera un conjunto de nuevos y distintos nombres (nonces) $\vec{n} = n_1, n_2, \ldots, n_m$ para las variables $\vec{x} = x_1, x_2 \ldots x_m$. Luego emite un mensaje de salida $M[\vec{n}/\vec{x}]$ (ej., M donde x_i es remplazado por n_i) hacia el *store* de mensajes y termina como el proceso $p[\vec{n}/\vec{x}]$. Este proceso de salida, liga las ocurrencias de las variables $\vec{x}$ en M y en p. Como un ejemplo de un típico proceso de salida se puede ver lo siguiente, $p_A = out\, new(\vec{x})\, \{x, A\}_{Pub(B)}.p$ donde A representa un agente que simplemente genera un mensaje con un *nonce* n y su propio identificador A encriptado con la llave pública del agente B. Se puede escribir $out\, new(\vec{x})\, M.p$ simplemente como $out\, M.p$ si el vector $\vec{x}$ está vacío.

El proceso de entrada $in\, pat\, \vec{x}\vec{\chi}\vec{\psi}\, M.p$ es el otro proceso que permite ligar variables en SPL, en donde se ligan las ocurrencias $\vec{x}\vec{\chi}\vec{\psi}$ en M ejecutando p. Como ejemplo de un típico proceso de entrada está, $p_B = in\, pat\, x, Z\, \{x, Z\}_{Pub(B)}.p$ en el cual se puede ver al agente B esperando por un mensaje de la forma $\{x, Z\}$ encriptado con su llave pública $pub(B)$. Si el mensaje p_A, citado anteriormente, se encuentra en el *store* de mensajes, se genera la siguiente instanciación de la siguiente manera $\{n/x, A/Z\}$, donde n se liga a x y A hace lo propio con Z. Si no se puede presentar

confusión alguna, el proceso $in\, pat\, \vec{x}\vec{\chi}\vec{\psi}\, M.p$ se puede abreviar como $in\, M.p$.

Finalmente, $\|_{i\in I}\, P_i$ denota la composición paralela de todos los P_i. Por ejemplo en $\|_{i\in\{A,B\}}\, P_i$ los procesos P_A y P_B, citados en los párrafos anteriores, corren en paralelo, de tal manera que se puedan comunicar. Se utilizará $!P = \|_{i\in\omega}\, P$ para representar un infinito número de copias de P ejecutándose en paralelo. En ocasiones se escribe $\|_{i\in\{1,2,\ldots n\}}\, P_i$. para decir $P_1 \parallel P_2 \parallel \ldots \parallel P_n$

Las nociones sintácticas concernientes a las variables libres y a los procesos/mensajes cerrados, se definen de la manera usual. Una variable es *libre* en un proceso/mensaje si no tiene una ocurrencia ligada dentro de ese proceso/mensaje. Se dice que un proceso/mensaje se considera cerrado si no posee variables libres.

Para mayor información relacionada con este lenguaje de verificación de protocolos de seguridad, se puede remitir a [Cra03, LARV06, AL06].

2.2. Timestamps

Los *timestamps* o marcas temporales, son elementos utilizados esencialmente en protocolos de comunicación para garantizar la seguridad en el flujo de la información. Éstas esencialmente representan la fecha y el tiempo actual, de forma que se pueda llegar a razonar acerca de la caducidad de los datos que se transmiten, asimismo como del orden en que se puedan presentar o generar varios mensajes. Esto resulta verdaderamente importante ya que se pueden llegar a prevenir nuevas variedades de ataques que no podrían ser detectados sin incluirse estas nociones de temporalidad, como lo son los ataques de replicación. Este tipo específico de ataques ocurre cuando un intruso filtra algún mensaje en la red y lo reenvía después, de forma que pueda alterar el correcto desempeño del protocolo. Claramente se puede ver que el uso de *nonces* para identificar los mensajes, no serviría de nada para impedir este tipo de ataques, ya que el receptor de mensajes dentro del protocolo no podría identificar si el mensaje está siendo enviado en el momento adecuado o no. En cambio, al hacer uso de *timestamps*, se puede modelar la caducidad de los mensajes y de esta forma se puede garantizar que el primer mensaje envíado dentro de su tiempo de vida puede ser recibido, en cambio el mensaje enviado a través de un ataque de replicación, ya habrá caducado y el receptor no lo recibirá. [NS93, DS81]

Para manejar estos elementos, primero se debe definir un ente que permita controlar el tiempo. Esto se puede realizar a través de diversas formas, bien sea a través de un reloj central que controle cada uno de los procesos presentes dentro del protocolo o a través de relojes ubicados en cada proceso, pero que se encuentren sincronizados entre sí de tal forma que cada uno de estos puedan tener acceso al mismo tiempo del sistema, bien sea para introducir un *timestamp* en un mensaje que se ha de enviar, como para poder verificar la veracidad de la marca de tiempo y por tal la caducidad del mensaje o el orden de ciertos mensajes.

2.3. Certificados de Autorización, Listas negras y ataques de replicación

En seguridad pocos estamentos se mantienen inmutables en el tiempo, y mantener la seguridad a medida que dichos sistemas cambian se convierte en una actividad de mayor trascendencia y de mayor complejidad debido a los múltiples cambios que se originan día a día. Uno de los mecanismos ideados para garantizar la seguridad en un sistema se conoce como *Listas de revocación*. Son comúnmente utilizadas en organizaciones que expiden certificados de autorización, los cuales son usados conjuntamente con timestamps para la correcta autenticación en el sistema. Sin embargo, si

un agente autenticado pierde sus privilegios antes que su certificado caduque (ej. es expulsado de la compañía, o su llave ha sido comprometida ante entes externos), el sistema debe estar en capacidad de revocar los permisos a los cuales puede acceder dicho agente. Esto se logra incluyendo dentro de los mecanismos de verificación las listas de revocación, en donde se ubican todos los certificados validos por cierto tiempo pero que hayan perdido validez.

Otro de los grandes problemas encontrados en relación a los protocolos de seguridad tiene que ver con la aparente estabilidad de algunos protocolos seguros para un número fijo de interacciones. Considérese el siguiente protocolo:

$$
\begin{aligned}
Bob &\longrightarrow Charles : \langle Bob, Pub(Bob) \rangle \\
Charles &\longrightarrow Bob : \{Charles, Pub(Charles)\}_{Pub(Bob)} \\
Bob &\longrightarrow Charles : \{Withdraw_i, B\}_{Pub\{Charles\}} \\
Charles &\longrightarrow Bob : \{Accept, Withdraw_i, B\}_{Pub\{Bob\}}
\end{aligned}
$$

Aparentemente, el uso de llaves públicas garantiza que la transacción es realizada por agentes fiables, por lo cual Charles actúa debitando una suma de la cuenta de Bob; sin embargo, considere que el primer mensaje es captado por un agente malicioso David, el cual se comunica con Charles, debitando diversas cantidades sin la previa autorización de Bob, como en el siguiente ejemplo:

$$
\begin{aligned}
Bob &\longrightarrow David : \langle Bob, Pub(Bob) \rangle \\
David &\longrightarrow Charles : \langle Bob, Pub(Bob) \rangle \\
Charles &\longrightarrow David : \{Charles, Pub(Charles)\}_{Pub(Bob)} \\
Bob &\longrightarrow David : \{Withdraw_i, B\}_{Pub\{Charles\}} \\
David &\longrightarrow Charles : \{Withdraw_i, B\}_{Pub\{Charles\}} \\
Charles &\longrightarrow David : \{Accept, Witdraw_i, B\}_{Pub\{Bob\}} \\
&\vdots \\
David &\longrightarrow Charles : \{Withdraw_i, B\}_{Pub\{Charles\}} \\
Charles &\longrightarrow David : \{Accept, Witdraw_i, B\}_{Pub\{Bob\}}
\end{aligned}
$$

Este ataque, conocido como *ataque de replicación*, es particularmente peligroso en términoss de seguridad. Dado su poder, diversas técnicass se han puesto en practica mejorando protocolos con técnicas de autenticación y confianza más robustas, o adicionando comportamientos en el protocolo, de manera que mediante un chequeo dentro del servidor se verifique que el mensaje *no* ha sido enviado previamente [Cha85].

3. Elementos que permitan expresar caducidad en los mensajes en SPL

Como se ha citado anteriormente, debido a la importancia que pueden llegar a tener la inclusión de nociones que permitan representar caducidad y orden en cuanto a los mensajes en un protocolo

de comunicación, para garantizar diversas propiedades de seguridad y para prevenir otros tipos de ataques, se realizarán una serie de inclusiones sutiles sobre el lenguaje de forma que no sólo se puedan prevenir una mayor gama de ataques a la seguridad de los sistemas, sino que no se pierda la esencia de este lenguaje y se permita de esta forma verificar las propiedades de seguridad sobre estos nuevos protocolos, haciendo uso de las flexibles y generales técnicas de prueba del cálculo de procesos en mención.

Ya que en SPL no existe una noción de temporalidad, se define en primera instancia, la forma en la cual se habrá de manejar este tipo de elementos dentro de este lenguaje sin llegar a afectar su semántica operacional. Este complejo concepto de tiempo, se tratará de manera relativamente sencilla, asumiéndose un ente presente dentro del protocolo ha modelarse, el cual llevará un conteo sincronizado de ticks de manera que se pueda llegar a asumir como un reloj. De esta manera cuando algún proceso en ejecución dentro del protocolo requiriera una marca de tiempo, simplemente pueda llegar a solicitárselo a este componente, el cual sin ningún problema podrá suministrárselo de manera acertada. Este elemento se denominará secuenciador y será explicado de manera más amplia y explícita en la sección 3.1.3.

3.1. Nuevos Elementos

Antes de adentrarse en el tema del manejo del tiempo en SPL, y sobretodo para poder incluir nuevos conceptos de temporalidad en un cálculo que en principio no contiene los elementos idóneos para manejar estas características, se deben establecer varias definiciones sobre variables, componentes y algunas codificaciones que permitan representar conceptos relacionados al tiempo:

3.1.1. Timestamps:

Definiendo el tiempo como un instante único e irrepetible, consideraremos el uso de nonces para modelar *ticks* de reloj los cuales permiten expresar la linealidad e irrepetibilidad sin incluir elementos externos. A partir de esta definición, se define un *timestamp* como la concatenación de los ticks generados desde un momento inicial hasta el presente, el cual es generado y administrado por un proceso secuenciador hacia cualquier ente presente en la red. Esto de tal manera que cualquier marca temporal tenga conocimiento sobre los instantes previos de tiempo permitiéndole verificar tiempos inexistentes o caducos. De manera formal, se define un elemento T de tipo *timestamp*, como una extensión de los nombres utilizados comúnmente dentro de la sintáxis de SPL, en donde la única diferencia frente a su base, es su significado subyacente, que implica la la representación de un estado en el tiempo mediante un conjunto de nonces.

$$T \equiv (t_1, t_2, ..., t_i) \tag{1}$$

Figura 1: Timestamp generado en el momento definido en el instante t_i

3.1.2. Tamaño de Variables:

Al realizar un evento de entrada en SPL $(in\, X)$, la variable X puede ligarse tanto a una tupla (a, b) como a un simple nombre a presente dentro del store. Esto se debe a que la semántica de SPL no logra resolver este tipo de elecciones de una manera determinística frente al reconocimiento de

patrones sobre nombres concatenados. El hecho de no tener variables con un tamaño específico, es uno de los mayores problemas para garantizar una selección particular. Por tal razón y para garantizar que una variable se ligue a una cantidad específica de nombres, se define la siguiente noción sobre variables: Decimos que $X_{\{1..n\}}$ implica que la variable X puede ligarse o bien a un solo elemento, como hasta tuplas de n componentes. Esto resulta sumamente importante, pues permite verificar la validez de un vector ordenado de ticks representando un *timestamp*, ya que se podrá garantizar la posición en donde ha de estar el último tick al momento de recibir un mensaje, verificando así los datos que se envían adjunto a esta marca de tiempo han caducado o no. (Esto se explicará de manera más profunda cuando se explique el proceso $Check$. en la sección 3.1.5)

3.1.3. Secuenciador de Timestamps

El proceso Sec representa un agente, que internamente va generando ticks de reloj t_i cada determinado intervalo i y lo va añadiendo a un vector $\vec{Ts}$ de tipo *timestamp*, de forma que se lleve un orden entre los componentes y así la temporalidad se vea reflejada. De esta forma el último tick generado, es aquel que representa el momento actual, siendo su posición dentro del vector la que da a entender el tiempo que ha pasado desde el comienzo hasta el evento actual. De esta forma cada proceso que solicite un timestamp a este secuenciador, recibirá el vector $\vec{Ts}$ de tipo *timestamp* que se lleva hasta el momento del requerimiento y que tiene un tamaño n y está compuesto por ticks de reloj, en donde los $n-1$ primeros valores son los ticks pasados y el n es el tick actual.

Este vector $\vec{Ts}$ es esencial cuando algún proceso que desee conocer si un mensaje que contiene un timestamp T y una duración n, ha caducado o no. Este proceso resulta bastante simple, ya que aquel que recibe el mensaje solamente debe verificar a través de un chequeo en la entrada del timestamp que pide al secuenciador, que el último tick presente en el *timestamp* del mensaje se encuentre a una distancia n del actual tick.

Sec se representa de la siguiente manera:

$$Sec(Pub(x)) \equiv out\, new(T)\, \{\vec{Ts}, T\}_{Pub(x)} \tag{2}$$

3.1.4. TimedOut

Se define un *encoding* que permite representar el envío de un mensaje junto con el timestamp actual proveniente del secuenciador.

$$TimedOut(T, \psi) \equiv out\, new(x)\, x\, .\, Sec(Pub(x))\, .\, in\{\vec{T'}, T_{\{1\}}\}_{Pub(x)}\, .out\, (Ts, \psi) \tag{3}$$

Donde Ts tiene la forma $\vec{T'}, T_{\{1\}}$. Este proceso recibe el *timestamp* previo, ejecuta un llamado al secuenciador actualizando los valores del timestamp y enviando el mensaje ψ con el tiempo actualizado.

3.1.5. Check

Utilizamos esta codificación para verificar la caducidad de un mensaje dado comprobando si la distancia en tiempo desde el último tick t presente en el timestamp T hasta el último tick t_1 presente en T_1, es menor a un tiempo de expiración L, permitiendo su evolución si el tiempo se encuentra dentro del rango válido o abortándolo de lo contrario. Esto se puede realizar verificando que entre la

ubicación de T y T_1 en el vector $\vec{T}s$, existan una cantidad de elementos menor o igual a L, lo cual es formalizado a continuación:

$$Check(T, T_1, L) \equiv out\, new(x)\, x\, .\, Sec(Pub(x))\, .\, in\{\vec{T'}, T_{\{1\}}, T''_{\{1..L-1\}}, T_{1\,\{1\}}, T'''_{\{1..\infty\}}\}_{Pub(x)} \quad (4)$$

Mediante el ejemplo de la figura 2 se puede clarificar el razonamiento. En este caso el agente A envía un mensaje a B con un timestamp T_A y un tiempo de expiración n. En el momento en que B recibe el mensaje debe chequear si este no ha caducado, teniendo en cuenta el timestamp del mensaje T_A y su vigencia n de tal forma que pueda continuar la comunicación con A.

$$\begin{aligned} A &\equiv TimedOut\,\{T_A, \psi, n\}_{Key(A,B)} \\ B &\equiv in\,\{T, \psi, n\}_{Key(A,B)}\, .\, TimedOut\,\{T_B, \psi, n\}_{Pub(B)}\, .Check(T, T_B, n)\, .\, Success \end{aligned}$$

Figura 2: Autenticación basada en timestamps

3.2. Protocolo de Autenticación Kerberos

Uno de los casos más famosos en la literatura de seguridad en donde se hace uso de marcas temporales para garantizar correctitud es el protocolo de autenticación Kerberos [SNS88], una extensión del protocolo Needham-Schröeder [NS78] robusta ante ataques de replicación.

Kerberos hace uso de un tercer ente denominado como **Key Distribution Center** (KDC), que se divide en dos diferentes partes lógicas: un servidor de autenticación (AS) y un servidor concesor de tiquetes (TGS). Esencialmente Kerberos hace uso de tiquetes que incluyen *timestamps* y tiempos de vida de mensajes, de tal manera que actuén como indicadores de caducidad del mensaje y por ende del canal de comunicación. En concreto Kerberos actúa como el administrador veraz que sirve como medio intercesor para la autenticación de un cliente y un servidor, dando fe de la autenticidad del cliente respecto al servidor. Esto se realiza primero a través de una autenticación y luego enviando un tiquete que incluye el tiempo actual, la llave por la cual se comunicará con el servidor con el que se desea autenticar y el tiempo de vigencia que posee para establecer dicha autenticación entre el cliente y el servidor.

3.2.1. Modelo Dolev-Yao

Haciendo uso del modelo Dolev-Yao se presenta la especificación del protocolo, donde *Alice* (A), actuando como cliente , se autentica con *Bob* (B), a través del servidor *Kerberos* (S). De esta forma utilizamos la notación presente dentro de este modelo de la siguiente forma $X \longrightarrow Y : M$, donde el agente X envía un mensaje M, bien sea encriptado o no, al agente Y.

$$\begin{array}{lcl} A \rightarrow S & : & A, B \\ S \rightarrow A & : & \{T_S, L, K_{AB}, B, \{T_S, L, K_{AB}, A\}_{K_{BS}}\}_{K_{AS}} \\ A \rightarrow B & : & \{T_S, L.K_{AB}, A\}_{K_{BS}}, \{A, T_A\}_{K_{AB}} \\ B \rightarrow A & : & \{T_A + 1\} \end{array}$$

Figura 3: Modelo Dolev-Yao de Kerberos

Como primer paso, A se comunica con el servidor S presentando las identificaciones del originador y del receptor. Al recibir estos datos S genera una llave de sesión K_{AB} por la cual le permitirá comunicarse al cliente A con B, además de crear un tiquete que contiene el nombre del cliente A, el tiempo actual T_S, el tiempo de vigencia para la autenticación L, la llave de sesión K_{AB} y el nombre del cliente A encriptado con una llave compartida entre S y B, de tal forma que el cliente A no pueda cambiar el tiempo de vigencia del mensaje. Por último S envía a A el tiquete encriptado junto con el tiempo actual T_S, la llave de sesión K_{AB}, el tiempo de vigencia L y el nombre del servidor B, en un mensaje encriptado con la llave que comparten S y A.

En el momento en que A recibe el mensaje de S, construye un mensaje donde incluye su identificación y el tiempo presente T_A encriptado con la llave de sesión K_{AB}. El mensaje es enviado hacia B junto con el tiquete proveniente de S. En el momento en que B recibe el mensaje, desencripta el tiquete y obtiene la llave de sesión la que desencripta el mensaje de A, de forma que luego pueda comparar que el espacio de tiempo entre el tiempo T_S generado por el servidor S y el tiempo T_A en el que A se autenticó, no sobrepasa el límite de vigencia L para el proceso de autenticación entre A y B. Luego de aclarar que A si ha enviado un mensaje no caduco, el servidor B responde confirmando la autenticación enviando el *timestamp* $T_A + 1$.

3.2.2. Modelo en SPL

Utilizando el lenguaje SPL junto las modificaciones planteadas en un comienzo, el protocolo se presenta en la figura 4:

$$A \equiv out\,(A,B)\,.\,in\,\{T_S, L, Key(A,B), B, W\}_{Key(A,S)}\,.\,out\,W\,.\,TimedOut\,\{A, T_A\}_{Key(A,B)}\,.\,in\{T_A\}_{Key(A,B)}$$

$$S \equiv in\,(A,B)\,.\,TimedOut\,\{T_S, L, Key(A,B), B, \{T_S, L, Key(A,B)\}_{Key(B,S)}\}_{Key(A,S)}$$

$$B \equiv in\,\{T_S, L, Key(A,B), A\}_{Key(B,S)}\,.\,in\,\{A, T_A\}_{Key(A,B)}\,.Check(T_S, T_A, L)\,.\,out\,\{T_A\}_{Key(A,B)}$$

Figura 4: Modelo de Kerberos en SPL

Como se puede observar, este modelo resulta bastante intuitivo y comprensible, además de no presentar mayores modificaciones frente a la sintaxis del lenguaje lo cual permitiría verificar sin lugar a dudas mediante a las técnicas de prueba inherentes a SPL, las propiedades de seguridad que dice cumplir este protocolo

4. Test Negativos

SPL presenta un poderoso chequeo de patrones que le permiten a un proceso específico evolucionar a una interacción si existe un mensaje en la red que cumpla con un patrón dado. Sin embargo, el lenguaje se queda corto cuando intentamos razonar sobre información complementaria en un conjunto determinado [Zav98]. A partir de este hecho, se incluye un nuevo elemento dentro del patrón de SPL que permita denotar este tipo de información.

$$in\neg Bl[M].p \equiv \|_{i \in x \wedge i \notin Bl}\, in\, M_i.p \,\|_{j \in Bl}\, in M_j.Fail \quad (5)$$

Donde Bl es un conjunto previamente determinado y $Fail$ es un proceso que bloquea una posterior ejecución. Básicamente este proceso se comporta como una composición paralela en donde un proceso recibe todos los mensajes que pertenezcan al conjunto Bl a los cuales bloquea, y permite el paso de los mensajes que se encuentren determinados y no pertenezcan a Bl. Es de aclarar que el patrón no puede lidiar con ausencia de información, ya que una evolución del sistema parcial o totalmente indeterminado presentaría fallos de consistencia respecto a posibles mensajes adicionados al almacén de mensajes en el futuro.

Un ejemplo del uso de estos chequeos esta presente en el *Online Certificate Status Protocol* (OCSP) [MAM+99], el cual incluye listas de revocación (CRLs [Mic96, Sta04]). El funcionamiento de dicho protocolo se asemeja mucho al manejo de listas negras explicado anteriormente, con los siguientes pasos:

1. Alice y Bob presentan certificados generados por Ivan, el servidor de certificados.
2. Alice, al intentar establecer comunicación con Bob, le envía su llave pública, y su certificado.
3. Bob, preocupado por la validez de la información, pregunta a Ivan si el certificado ha sido comprometido, por un canal de comunicación compartido.
4. Si Ivan encuentra que el certificado de Alice no ha sido comprometido, envía un mensaje de aceptación a Bob.
5. Bob recibe la información de Ivan y autentica la validez del mensaje, usando el *signature* del servidor.
6. Habiendo solucionado las dudas, Bob intercambia las llaves con Alice y continúan su proceso autenticados.

Como puede verse modelado en SPL a continuación:

$$\begin{aligned}
A \equiv\ & out\, new\, C_a\{C_a\}_{Pub(I)}\, .\, out\, \{Pub(A), C_a\}_{PubB}.\, in\{C_a, Ack, Key(A,B)\}_{Pub(A)}. \\
& outnew(MSG)\{MSG\}_{key(A,B)} \\
B \equiv\ & out\, new\, C_b\{C_b\}_{Pub(I)}.\, out\, new R\, \{R, Pub(A), C_a, B\}_{PubI}.\, in\, \{\langle\{Ok, C_a\}_{Priv(I)}\rangle\}_{Pub(B)} \\
& .out\, new\, (Ack, Key(A,B))\, \{C_a, Ack, Key(A,B)\}_{Pub(A)}.\, in\{MSG\}_{key(A,B)} \\
I \equiv\ & !(in\, \{C_x\}_{Pub(I)})\, \|!in\, \{X, Pub(X), C_x, Y\}_{Pub(I)}.\, in\neg Bl[C_x].\, out\{\langle\{Ok, C_x\}_{Priv(I)}\rangle\}_{Pub(Y)}
\end{aligned}$$

5. Discusión

Este documento presenta dos ideas importantes que cabe destacar: La primera se basa en la inclusión de elementos que permiten modelar caducidad en los mensajes, de tal forma que nuevos protocolos que incluyen mecanismos de este tipo con el fin de contrarrestar ataques de intrusos que pretenden desestabilizar el buen desempeño del protocolo, puedan ser modelados a través de la sintaxis intuitiva que presenta SPL, permitiendo de igual forma la posibilidad de realizar verificaciones veraces sobre propiedades de seguridad frente a esta clase de ataques, a través de las flexibles técnicas de pruebas presentes en este lenguaje. Este modelo puede extender la sintaxis original de

SPL, redefiniendo el proceso de salida $out\, new(\vec{x})\, M.p$ por un $timedOut(x, M).p$ realizando cambios menores en el encoding anteriormente presentado de manera que sea posible generar nuevos nombres paramétricamente. Lo anterior permitiría usar SPL de una manera completamente temporal. Esta extensión permite inferir la posibilidad de incluir al modelo razonamientos basados en lógicas temporales lineales (LTL) sin modificar su semántica operacional, abriendo nuevas posibilidades en la verificación de nuevas propiedades en seguridad.

Como segunda contribución destacamos igualmente el hecho de introducir dentro de SPL, nociones que permitieran razonar acerca de la reputación de los agentes con los que se pretende entablar una interacción, previa a la realización del acto de comunicación en sí.

Referencias

[AG99] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Inf. Comput.*, 148(1):1–70, 1999.

[AL06] Andrés A. Aristizábal and Hugo A. López. Using process calculi to model and verify security properties in real life communication protocols. Trabajo de grado, Pontificia Universidad Javeriana, 2006.

[ALR05] Andrés Aristizábal, Hugo A. López, and Camilo Rueda. Using a declarative process language for P2P protocols. *The Association for Logic Programming Newsletter Digest*, 18(4), November 2005.

[CCM02] Mario José Cáccamo, Federico Crazzolara, and Giuseppe Milicia. The ISO 5-pass authentication in χ-Spaces. In Youngsong Mun and Hamid R. Arabnia, editors, *Proceedings of the Security and Management Conference (SAM'02)*, pages 490–495, Las Vegas, Nevada, USA, June 2002. CSREA Press.

[Cha85] David Chaum. Security without identification: transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.

[Cra03] Federico Crazzolara. Language, semantics, and methods for security protocols. Doctoral Dissertation DS-03-4, brics, daimi, May 2003. PhD thesis. xii+160.

[CW01] Federico Crazzolara and Glynn Winskel. Events in security protocols. In *ACM Conference on Computer and Communications Security*, pages 96–105, 2001.

[DS81] D. Denning and G. Sacco. Timestamps in key distributed protocols. *Communication of the ACM*, 24(8):533–535, 1981.

[DY81] Danny Dolev and Andrew C. Yao. On the security of public key protocols. Technical report, Dept. of Computer Science, Stanford University, Stanford, CA, USA, 1981.

[Jen94] Kurt Jensen. An introduction to the theoretical aspects of coloured petri nets. In *A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium*, pages 230–272, London, UK, 1994. Springer-Verlag.

[LARV06] Hugo A. López, Andrés A. Aristizábal, Camilo Rueda, and Frank D. Valencia. Process calculi for the verification of security properties of communication protocols for peer-to-peer systems. Unpublished, January 2006.

[MAM+99] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. RFC 2560: X.509 Internet Public Key InfrastructOnline Certificate Status Protocol - OCSP. IETF RFC Publication, June 1999.

[Mic96] S. Micali. Efficient certificate revocation. Technical Report MIT/LCS/TM-542b, 1996.

[MPW89] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, parts I and II. Technical Report -86, 1989.

[NS78] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.

[NS93] B. Clifford Neuman and Stuart G. Stubblebine. A note on the use of timestamps as nonces. *Operating Systems Review*, 27(2):10–14, 1993.

[Per96] Charles Perkins. IP Mobility Support - RFC2002. IETF RFC Publication, 1996.

[SNS88] Jennifer G. Steiner, B. Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of the USENIX Winter 1988 Technical Conference*, pages 191–202, Berkeley, CA, 1988. USENIX Association.

[Sta04] William Stallings. *Fundamentos de Seguridad en Redes. Aplicaciones y Estándares*. Pearson Educación, Madrid,, 2004.

[Zav98] G. Zavattaro. Towards a hierarchy of negative test operators for generative communication, 1998.