Análisis Formal del Estándar NIST para Modelos RBAC

Carlos D. Luna[1] and Cristian D. Rosa[2]

[1] Instituto de Computación, Universidad de la República, Uruguay
cluna@fing.edu.uy
[2] FCEIA, Universidad Nacional de Rosario, Argentina
crosa@fceia.unr.edu.ar
Institución Actual: LORIA-INRIA, Equipe AlGorille, Batiment B, BP 239, 54506 Vandoeuvre-lès-Nancy, Cedex, France

Resumen *Role Based Access Control* (RBAC) es un enfoque basado en el conepto de rol para implementar políticas de control de acceso. Actualmente es considerado uno de los modelos más generales, debido a su neutralidad respecto a las políticas de control de acceso y a su flexibilidad. A causa de la existencia de múltiples variantes de modelos e implementaciones RBAC, el National Institute of Standards and Technology (NIST) realizó una propuesta de estandarización, definiendo un modelo de referencia. Si bien la especificación se desarrolla en un lenguaje de especificación formal (en lenguaje Z), la propuesta no incluye un análisis de su corrección, ni la verificación de propiedades de seguridad. Este artículo presenta un análisis formal de los modelos básico (*Core RBAC*) y jerárquico (*Hierarchical RBAC*) del estándar NIST, que pone en evidencia la existencia de imprecisiones, omisiones y errores en la especificación de referencia. A partir de este análisis el trabajo introduce una nueva especificación formal de *Core RBAC* y *Hierarchical RBAC* que subsana los problemas encontrados. En torno a la nueva formalización, desarrollada en el Cálculo de Construcciones Inductivas, se demuestran algunas propiedades relevantes de seguridad inherentes a los modelos RBAC, usando el asistente de pruebas Coq.

Palabras clave: Control de Acceso Basado en Roles, Especificación y Verificación Formal, Teoría de Tipos, Coq.

1. Introducción

El concepto del rol como entidad central en el control de acceso se origina con los sistemas multiusuarios, en la década del 70 [11]. La utilización de grupos de usuarios en UNIX, entre otros sistemas, y el agrupamiento por privilegios en sistemas de administración de bases de datos son las raíces de los sistemas actuales [2, 13, 14]. *Role Based Access Control* (RBAC) es un enfoque basado en el concepto de rol para implementar políticas de control de acceso. Actualmente es considerado uno de los modelos más generales, debido a su neutralidad respecto a las políticas de control de acceso y a su flexibilidad, que le permiten simular

enfoques alternativos como DAC y MAC. Los modelos RBAC han demostrado su efectividad en ámbitos muy variados, como en organizaciones gubernamentales [5] y en complejas aplicaciones web [6]. En RBAC los permisos son asociados con roles y los usuarios son asignados a roles apropiados, esto simplifica en gran medida la administración de los permisos. Los roles son creados según responsabilidades y niveles de autoridad en el contexto de una organización. Un rol es la construcción semántica sobre la cual se formulan las políticas de control de acceso. La colección de usuarios y permisos vinculados por un rol es transitoria.

Los modelos e implementaciones RBAC existentes son relativamente similares en los conceptos fundamentales, pero difieren, significativamente en algunos casos, en los detalles. Las diferencias no siempre resultan obvias, debido a que suelen utilizarse diferentes terminologías para describir los mismos conceptos. Para dar solución a problemas de alcance y terminología, el National Institute of Standards and Technology (NIST) realizó una propuesta de estandarización que unifica las implementaciones comerciales más relevantes y los principales modelos desarrollados en ámbitos académicos [4, 10]. El estándar referido, adoptado en 2004 como un estándar ANSI/INCITS, propone un modelo de referencia, que define la terminología general e introduce los elementos básicos de RBAC, y una especificación funcional en lenguaje Z, que describe los requerimientos de los comandos utilizados para interactuar con el sistema.

Este artículo presenta un análisis formal de los modelos básico (*Core RBAC*) y jerárquico (*Hierarchical RBAC*) del estándar NIST, que pone en evidencia la existencia de imprecisiones, omisiones y errores en la especificación propuesta, en mayor medida debidos a decisiones libradas a criterios de implementación. A partir de este análisis el trabajo introduce una nueva especificación formal de los modelos RBAC considerados que subsana los problemas encontrados. En torno a la nueva formalización, desarrollada en el Cálculo de Construcciones Inductivas (CCI), se demuestran importantes propiedades de seguridad inherentes a los modelos RBAC, usando el asistente de pruebas Coq [3, 12]. Si bien existen trabajos previos que analizan el estándar NIST referido, tales como [1, 7, 8], los mismos no plantean una nueva especificación ni verifican formalmente propiedades de seguridad del modelo. Asimismo, el presente trabajo, a diferencia de los anteriores, utiliza un lenguaje (una teoría de tipos) que permite derivar especificaciones funcionales correctas por construcción. En particular, como ventaja adicional de este enfoque, un modelo ejecutable podría utilizarse en una técnica complementaria de la verificación formal, como es el testing de caja negra, donde oficie de oráculo frente a casos de pruebas derivados de la especificación.

Una versión completa de este trabajo es el reporte [9]. La nueva especificación formal introducida en este artículo, incluyendo las propiedades probadas usando Coq, están disponibles en http://www.fceia.unr.edu.ar/∼crosa.

El resto de este artículo está organizado como sigue. La sección 2 introduce el modelo RBAC propuesto por NIST. En la sección 3 se presenta un análisis de las decisiones de diseño libradas a criterios de implementación por parte del estándar y se justifican las elecciones tomadas en este trabajo, que dan lugar a una nueva especificación formal de *Core RBAC* y *Hierarchical RBAC*. La sección 4 describe

algunos errores encontrados en la especificación del estándar NIST. Luego, en la sección 5 se introducen aspectos relevantes de la nueva formalización desarrollada en el CCI y se mencionan algunas propiedades verificadas. Finalmente, la sección 6 exhibe las conclusiones y los trabajos futuros.

2. Modelo RBAC

Esta sección presenta el modelo de referencia RBAC propuesto por NIST [4]. Primero se introducen generalidades de RBAC, su organización modular y las funcionalidades provistas por cada componente. Posteriormente se describen, con mayor detalle, los componentes que serán analizados en este trabajo.

2.1. Descripción General del Modelo

El modelo de referencia NIST y su especificación funcional están divididos en cuatro componentes o modelos: *Core RBAC*, *Hierarchical RBAC*, *Static Separation of Duty Relations* (SSD) y *Dynamic Separation of Duty Relations* (DSD). Cada uno de estos componentes extiende la funcionalidad del componente anterior.

Core RBAC abarca los aspectos esenciales de RBAC. Dentro de los conceptos básicos de RBAC tenemos que: los usuarios son asignados a roles, los permisos son asociados a roles y los usuarios adquieren permisos, siendo miembros de roles. Las asignaciones usuario-rol y permiso-rol permiten que un usuario pertenezca a varios roles simultáneamente y que un rol posea múltiples usuarios. Asimismo, un permiso puede ser asociado a varios roles y un rol puede tener asociados múltiples permisos. *Core RBAC* incluye también el concepto de sesión, que permite la activación y desactivación selectiva de roles, con el objetivo de minimizar el número de permisos ejercidos simultáneamente, limitándose a los necesarios para cada operación. Las decisiones de acceso se realizan sobre el conjunto de permisos asociados con los roles activos de cada sesión.

Hierarchical RBAC agrega al modelo anterior los elementos necesarios para el soporte de jerarquías de roles. Una jerarquía es un orden parcial en el conjunto de roles que define una relación de herencia entre éstos. Los roles ascendientes en la jerarquía adquieren los permisos de sus descendientes y los roles descendientes adquieren los usuarios pertenecientes a sus ascendientes.

Los componentes SSD y DSD definen relaciones que se utlizan para hacer cumplir políticas de conflicto de intereses. Un conflicto de interés puede darse cuando un usuario obtiene autorización para ejercer permisos asociados con roles conflictivos. Por ejemplo, si un usuario tiene asignados un rol que le permite gestionar una compra y un rol que lo autoriza a aprobar dicha operación. Ambos componentes permiten especificar políticas de conflicto de intereses de manera centralizada y luego imponerlas uniformemente en el sistema. La diferencia entre SSD y DSD reside en el contexto en el cual la política es impuesta. Mientras que SSD restringe las asignaciones usuario-rol, DSD limita la disponibilidad de permisos, imponiendo restricciones en los roles que pueden ser activados simultáneamente en las sesiones.

2.2. Core RBAC

Como se mencionó previamente, el modelo RBAC está esencialmente definido en términos de usuarios individuales asignados a roles y de permisos asociados a roles. Se puede considerar al rol como la manera de describir relaciones "muchos a muchos" entre usuarios individuales y permisos. Por razones de simplicidad, un usuario está definido, en general, como un ser humano. Un rol es un trabajo o función, en el contexto de una organización, con alguna semántica relacionada a la autoridad y responsabilidad conferida a los usuarios asignados a dicho rol. Un permiso es una autorización para realizar una operación sobre un objeto determinado. Una operación es una acción que, al ser invocada, ejecuta una tarea para un usuario. En tanto que un objeto es una identidad que contiene o recibe información.

Core RBAC define cinco conjuntos de elementos de datos básicos: usuarios (USERS), roles (ROLES), objetos (OBS), operaciones (OPS) y permisos (PRMS). La figura 1 esquematiza la relación que existe entre los elementos que componen el modelo (la relación *Role Hierarchy* se describe en la sección 2.3).

Para que un usuario pueda ejercer los permisos para los cuales está autorizado, es necesario que éste active los roles que tienen asociados tales permisos. Para tal fin, *Core RBAC* define un conjunto de sesiones (SESSIONS), donde cada sesión es un mapeo entre un usuario y un subconjunto de roles activos. Cuando una sesión es creada debe proveerse un conjuto inicial de roles activos (posiblemente vacío), que posteriormente puede ser actualizado por el sistema. La utilización de sesiones incrementa la granularidad del control de acceso a los recursos y fortalece la aplicación del *principio del menor privilegio*. Las decisiones relacionadas a control de acceso son tomadas por usuario y sesión. Es decir, un usuario está autorizado para realizar una determinada operación sobre cierto objeto en una sesión, si existe algún rol activo en dicha sesión que tiene asignado el permiso correspondiente.

Core RBAC especifica los requerimientos funcionales de los comandos utilizados para administrar los elementos básicos del modelo. Estos comandos contemplan la creación y eliminación de usuarios, roles y permisos, la asignación de usuarios a roles y la asignación de permisos a roles. Asimismo, el modelo básico incluye comandos para instrumentar el control de acceso, realizar la gestión de sesiones e inspeccionar el estado del sistema.

2.3. Hierarchical RBAC

Las jerarquías de roles son una manera natural de estructurar los roles con el objetivo de reflejar las líneas de autoridad y las responsabilidades en una organización. El componente *Hierarchical RBAC* incopora al modelo básico el soporte para instrumentar jerarquías de roles. Para esto se agrega a *Core RBAC* una relación RH (*Role Hierarchy*) entre roles, que en la figura 1 se denota con el uso de líneas de puntos.

La relación de herencia se define en término de permisos y usuarios. Intuitivamente, que r_1 "herede de" r_2 tiene como consecuencia que todos los permisos

de r_2 son también permisos de r_1 y que todos los usuarios de r_1 son asimismo usuarios de r_2. Formalmente, $RH \subseteq ROLES \times ROLES$ es un orden parcial sobre el conjunto $ROLES$, llamado relación de herencia, que se denota con el símbolo $\succeq$. Luego, se cumple que: $r_1 \succeq r_2 \Rightarrow perms(r_2) \subseteq perms(r_1)$ y $users(r_1) \subseteq users(r_2)$, donde $perms$ y $users$ tienen una semántica formal que resulta intuitiva en el contexto previo. *Hierarchical RBAC* define los requerimientos funcionales de los comandos de gestión e inspección de la jerarquía, y especifica cambios en los requerimientos de algunos comandos de *Core RBAC*, que deben contemplar la existencia de una jerarquía de roles.

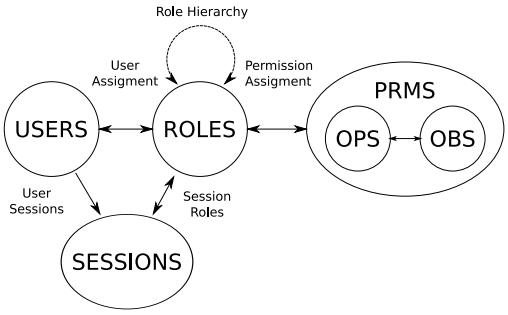

Figura 1. Elementos del Modelo RBAC

3. Análisis de las Decisiones de Diseño

El modelo RBAC propuesto por NIST, introducido en la sección 2, propone en múltiples partes a lo largo de su especificación decisiones de diseño que quedan libradas a criterios de implementación. A continuación incluimos un análisis de alternativas válidas para estas decisiones, según el estándar, que en ciertos casos dan lugar a potenciales fallas de seguridad, si no se toman recaudos apropiados. En cada caso planteamos restricciones a las decisiones de diseño, que serán consideradas luego en la formulación de una nueva especificación en la sección 5.

3.1. Operaciones sobre la Jerarquía de Roles

La jerarquía de roles está definida como un orden parcial sobre el conjunto de los roles. Esto es así ya que la transitividad y la antisimetría modelan de manera precisa el comportamiento de una relación de herencia.

Eliminación de Herencias Indirectas. Es una decisión de implementación permitir, o no, la eliminación de herencias indirectas, también llamadas implícitas o transitivas.

La figura 2 muestra una jerarquía de roles, donde $A \succeq B$, $B \succeq C$ y $B \succeq D$ se relacionan y representan de manera directa (explícita), y $A \succeq C$ y $A \succeq D$ se vinculan y representan indirectamente. Si se permitiese eliminar, por ejemplo,

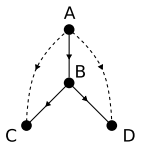

Figura 2. Jerarquía de Ejemplo

la herencia indirecta $A \succeq C$ se podría proceder de dos formas:

1. Eliminar $A \succeq C$, pero entonces se tendría que: $A \succeq B$ y $B \succeq C \not\Rightarrow A \succeq C$. Por lo tanto, $\succeq$ resultaría no transitiva y no sería un orden parcial. De esta manera se perderían las propiedades deseables de la jerarquía.
2. Para preservar el orden parcial se podría eliminar $B \succeq C$ o $A \succeq B$. Esto por un lado representa una elección no determinista y por el otro, la decisión que se tome podría llegar a afectar a roles intermedios, generándose de esta manera resultados impredecibles.

A partir del análisis previo, en este trabajo se considera que la eliminación de herencias debería restringirse únicamente a las que son de relación directa, es decir, de herencia inmediata.

Conservación de Herencias Indirectas. El estándar NIST considera que es una decisión de implementación la conservación o no de herencias indirectas inducidas por una herencia inmediata que se elimina. Por ejemplo, dada la jerarquía descripta en la figura 2, si se elimina $A \succeq B$, la conservación de $A \succeq C$ y $A \succeq D$ es opcional.

Existen dos alternativas para implementar esta funcionalidad, conservando las propiedades de orden parcial de $\succeq$. La primera consiste en exigir que todas las relaciones entre roles sean inmediatas. Para lograr esto, cada vez que se vinculan dos roles resulta necesario agregar de manera explícita todas las nuevas relaciones transitivas inducidas por dicho vínculo. Este enfoque subutiliza el hecho de que $\succeq$ es un orden parcial, ya que no se hace uso de la transitividad. La segunda alternativa es transformar en herencias inmediatas todas las relaciones inducidas por transitividad a partir de la relación eliminada. Este comportamiento complica el control de la jerarquía. Por ejemplo, si elimina una relación e inmediatamente se la vuelve a incorporar, no necesariamente la jerarquía resultante es igual a la inicial.

Como conclusión, en este trabajo se considera que las herencias indirectas relacionadas a través de un rol eliminado no deberían ser conservadas.

3.2. Manejo de Sesiones

Activación Explícita de Roles. Otra decisión librada a criterio de implementación se relaciona con la manera de proceder ante la creación de una sesión o la activación de un rol en una sesión existente, cuando se cuenta con una jerarquía de roles. El sistema puede activar sólo el rol o los roles indicados, o bien puede además activar automáticamente todos los roles heredados a través de éstos.

Si se tiene como objetivo preservar el *principio del menor privilegio*, es entonces muy importante que la implementación asegure que toda activación de roles sea explícita, ya que de esta manera se reduce el número de roles activos en todo momento.

Terminación de Sesiones al Eliminar un Usuario. Es una decisión de diseño establecer qué sucede con las sesiones de usuarios que son eliminados. El sistema puede terminar las sesiones inmediatamente o esperar a que éstas finalicen normalmente.

Si las sesiones de un usuario no se terminan al ser éste eliminado y no se modifica la semántica dada a algunos comandos, el sistema podría evolucionar a un estado vulnerable a una potencial falla de seguridad. Por ejemplo, considérese un estado del sistema donde se elimina cierto usuario y se preservan sus sesiones activas. Si posteriormente se crea el mismo usuario y se le asigna un conjunto de roles diferente, dicho usuario podría tener ahora sesiones con roles activos para los cuales posiblemente ya no esté autorizado. Esta consituye una grave falla de seguridad.

En este trabajo se considera que es necesario culminar de manera inmediata todas las sesiones de un usuario cuando éste es eliminado. De esta manera se evita cualquier inconsistencia entre el conjunto de sesiones y el conjunto de usuarios.

Terminación de Sesiones al Eliminar o Desasignar un Rol. Al igual que en el caso anterior, es también una decisión de implementación determinar qué sucede con las sesiones que tienen activo un rol, cuando éste es eliminado o desasignado a un usuario. Las opciones son: esperar hasta que las sesiones terminen normalmente, eliminar el rol de la lista de roles activos de las sesiones, o terminar inmediatamente dichas sesiones.

En el primer caso, el estado del sistema luego de desasignar un rol a un usuario no satisface la condición de integridad que debe existir entre los conjuntos de roles activos de las sesiones pertenecientes al usuario y el conjunto de los roles asignados al mismo. Esta falta de integridad implica una falla de seguridad, debido a que el usuario puede seguir ejerciendo permisos asociados a un rol para el cual ya no está autorizado. Adicionalmente, si se agregaran luego nuevos permisos a dicho rol, los mismos podrían ser ejercidos igualmente por el usuario.

Para este trabajo resultan válidas las dos últimas opciones, aunque en la propuesta de una nueva especificación se establece que las sesiones involucradas deberían ser finalizadas inmediatamante, en concordancia con la restricción formulada para el caso anterior.

4. Algunos Problemas de la Especificación

El estándar propuesto por NIST utiliza el lenguaje de especificación Z para formalizar los requerimientos funcionales de los comandos de interacción con el sistema. Si bien este lenguaje dota a la especificación de un grado de precisión y de claridad notable, un análisis riguroso de los requerimientos, resultado de aplicar herramientas de verificación, puso de manifiesto algunas impresiciones e incluso contradicciones en la especificación de la semántica de varios comandos. A continuación se describen algunos de los problemas encontrados. Un análisis completo está disponible en [9].

Los inconvenientes detectados en la especificación están relacionados con el componente *Hierarchical RBAC*, debido especialmente a la forma en que está definida la relación de herencia entre roles. En la sección "A.2 Requirements for Hierarchical RBAC" del estándar se define la relación de herencia a través de dos relaciones, una de herencia inmediata entre roles ($\gg$) y otra que corresponde a la clausura reflexo-transitiva de la anterior ($\geq$).

Un problema reside en los esquemas Z de los comandos *AddInheritance* y *DeleteInheritance*, que asumen un vínculo entre las relaciones $\gg$ y $\geq$ que no está especificado [4]. *AddInheritance* está definido de la siguiente manera:

"...This command establishes a new immediate inheritance relationship $r_{asc} \gg r_{desc}$ between existing roles r_{asc}, r_{desc}..."

$$AddInheritance(r\_asc, r\_desc : NAME) \lhd$$
$$r\_asc \in ROLES; r\_desc \in ROLES; \neg(r\_asc \gg r\_desc); \neg(r\_desc \geq r\_asc)$$
$$\geq' = \geq \cup \{r, q : ROLES \mid r \geq r\_asc \wedge r\_desc \geq q \bullet r \mapsto q\} \rhd$$

Claramente se observa que la poscondición del esquema no actualiza la relación $\gg$. Esto es, la especificación formal no coincide con la descripción que acompaña al esquema. Algo similar sucede con el comando *DeleteInheritance*, que tiene por definición y esquema:

"...This command deletes an immediate inheritance relationship $r_{asc} \gg r_{desc}$ ... in this definition, implied relationships are preserved after deletion..."

$$DeleteInheritance(r\_asc, r\_desc : NAME) \lhd$$
$$r\_asc \in ROLES; r\_desc \in ROLES; r\_asc \ll r\_desc$$
$$\geq' = (\gg \setminus \{r\_asc \mapsto r\_desc\})^* \rhd$$

Nuevamente la poscondición no especifica que sucede con la relación $\gg$. En el esquema anterior existe también otro problema. La descripción informal del comando aclara que las relaciones implícitas (transitivas) son preservadas, pero la semántica de *DeleteInheritance* no expresa esto. A fin de ejemplificar esta última observación, considérense las siguientes relaciones:

$$\gg = \{A \mapsto r\_asc, r\_asc \mapsto r\_desc, r\_desc \mapsto B\}$$
$$\geq = \ \gg \cup \{A \mapsto r\_desc, r\_asc \mapsto B, A \mapsto B\}$$

Si se ejecuta el comando $DeleteInheritance\, r\_asc\, r\_desc$, según su especificación, se tiene:

$$\begin{aligned} \geq' &= (\gg \setminus \{r\_asc \mapsto r\_desc\})^* \\ &= (\{A \mapsto r\_asc, r\_desc \mapsto B\})^* \\ &= \{A \mapsto r\_asc, r\_desc \mapsto B\} \end{aligned}$$

donde en la nueva relación $\geq'$ se perdieron las herencias implícitas.

Continuando con los comandos relacionados con la jerarquía de roles, los esquemas que especifican la semántica de *AddAscendant* y *AddDescendant* (que permiten agregar herencias inmediatas creando uno de los roles involucrados) poseen una contradicción en sus precondiciones. *AddAscendant* está especificado como sigue,

$$\begin{aligned} &AddAscendant(r\_asc, r\_desc : NAME) \lhd \\ &\quad AddRole(r\_asc) \\ &\quad AddInheritance(r\_asc, r\_desc) \rhd \end{aligned}$$

done *AddRole* permite agregar un rol en el sistema. Si se expanden los esquemas incluidos en la especificación, se obtiene como parte de la precondición la siguiente contradicción:

$$\begin{aligned} &AddAscendant(r\_asc, r\_desc : NAME) \lhd \\ &\quad r\_asc \notin ROLES; \\ &\quad r\_asc \in ROLES; \ldots \rhd \end{aligned}$$

Todo parece indicar que el origen de dicha contradicción proviene de asumir un concepto de *secuencialidad* entre los estados de los esquemas incluidos, que es inexistente en Z.

Por último y también relacionando con una funcionalidad incluida en *Hierarchical RBAC*, el estándar especifica que al eliminar un rol la jerarquía de roles no se modifica. Si bien esto no es afirmado explícitamente, al comienzo de la sección A.2a.1 se expresa que: "All functions of Section A.1.1 remain valid. In addition, this section defines the following new specific functions". La sección A.1.1 incluye la definición de *DeleteRole*, que por lo tanto no cambia en presencia de la jerarquía de roles. No obstante, esto se puede transformar en una potencial falla de seguridad, ya que el sistema podría transicionar a un estado en donde no haya integridad entre los roles en la jerarquía y el conjunto de roles del sistema. Por ejemplo, se pueden activar todos los roles heredados a través de un rol eliminado, a pesar de que éste ya no exista. Peor aún, si este rol es nuevamente creado en el sistema, estaría relacionado con sus antiguos herederos, agregando un comportamiento no especificado sobre el comando *AddRole*.

En este trabajo se considera que deberían eliminarse todas las relaciones de la jerarquía de roles en las cuales un rol eliminado esté involucrado.

5. Formalización en el CCI

Esta sección introduce una nueva especificación de los modelos *Core RBAC* y *Hierarchical RBAC*. La formalización, desarrollada en el CCI, respeta el estándar NIST bajo las consideraciones planteadas en las secciones 3 y 4. Por razones de espacio muchos detalles serán omitidos. El lector interesado puede referirse a [9] por más información.

5.1. Notación

Para los conectivos lógicos y la igualdad se utiliza la notación estándar $(\wedge, \vee, \rightarrow, \neg, \exists, \forall, =)$. Un tipo registro se define como sigue,

$$R \stackrel{\text{def}}{=} [\![\ field_0 : A_0, \ldots, field_n : A_n\ ]\!]$$

donde R es un tipo inductivo no recursivo con un único constructor y n funciones de proyección $field_i : R \rightarrow A_i$.

La notación para conjuntos de tipo A es *set* A. El tipo inductivo *set* codifica a los conjuntos como listas. Para los constructores de tipos inductivos de la forma

$$I : t_0 \ldots t_n \stackrel{\text{def}}{=} C_i\ (a_0 : k_0) \ldots (a_j : k_j) : I\ (b_0 : t_0) \ldots (b_n : t_n) \qquad \forall\ i, j, n \in \mathbb{N}_0$$

se usa alternativamente la notación:

$$C_i\ \frac{a_0 \ldots a_j}{I\ b_0 \ldots b_n}$$

donde las variables libres se consideran cuantificadas universalmente y sus tipos definidos por el contexto.

5.2. Formalización de Componentes

Preludio. Primero se introducen algunos tipos que serán utilizados en el resto de la especificación.

$User$, $Role$, Op, Ob y $Session$ representan los tipos de los usuarios, los roles, las operaciones, los objetos y las sesiones, respectivamente. Estos tipos constituyen elementos básicos del modelo RBAC.

Un permiso describe la posibilidad de realizar una operación sobre un objeto determinado. Formalmente es un elemento del siguiente tipo registro:

$$Perm \stackrel{\text{def}}{=} [\![\ op_0 : Op,\ ob_0 : Ob\ ]\!]$$

donde op_0 representa una operación sobre el objeto ob_0.

Los permisos pueden estar asociados a múltiples roles. Estas asociaciones están definidas por el tipo:

$$PA_{elem} \stackrel{\text{def}}{=} [\![\ r_1 : Role,\ p_1 : Perm\ ]\!]$$

De manera análoga, los usuarios pueden estar asignados a uno o más roles. El siguiente tipo registro formaliza esta relación:

$$UA_{elem} \stackrel{\text{def}}{=} [\![\ u_0 : User,\ r_0 : Role\ ]\!]$$

Jerarquía de Roles. La relación de jerarquía entre roles es un orden parcial sobre el conjunto de roles del sistema. Para simplificar la manipulación de esta jerarquía se consideran dos relaciones: una de herencia inmediata o directa,

$$(\ll) : Role \to Role \to Prop$$

y su respectiva clausura reflexo-transitiva, definida inductivamente como sigue:

$$(\preceq) : Role \to Role \to Prop$$

$$\text{rt\_refl}\ \frac{}{x \preceq x} \qquad \text{rt\_step}\ \frac{x \ll y}{x \preceq y} \qquad \text{rt\_trans}\ \frac{x \preceq y \quad y \preceq z}{x \preceq z}$$

La jerarquía de roles está vinculada con los usuarios del sistema a través de la relación $authorizedUser$. Un usuario está autorizado para un rol si éste está asignado directamente a él o bien si está asignado a un rol que hereda del mismo. Formalmente,

$$authorizedUser\ (u{:}User)\ (r{:}Role) \stackrel{\text{def}}{=} \exists\ r'{:}Role,\ role\_of\_user\ r'\ u \wedge r \preceq r'$$

donde $role\_of\_user$ es el predicado característico para la asignación de roles a usuarios.

Sesiones. De acuerdo a la descripción dada en la sección 2.3, una sesión es un mapeo entre los usuarios y sus roles activos. Un usuario puede tener más de una sesión establecida de manera simultánea. El estado de las sesiones del sistema está dividido en dos partes. La primera es un conjunto de identificadores válidos de sesión de tipo $Session$. La segunda parte es una función que mapea cada sesión con el usuario dueño de la misma y con el conjunto de sus roles activos, esto es, $se : Session \to SE_{elem}$, donde SE_{elem} es el tipo:

$$SE_{elem} \stackrel{\text{def}}{=} [\![\ usr{:}User,\ rl{:}set\ Role\ ]\!]$$

siendo el campo usr el usuario dueño de la sesión y rl el conjunto de roles activos de la misma.

Estado. El estado del sistema se describe a través del siguiente tipo registro:

$$\begin{aligned} State \stackrel{\text{def}}{=} [\![\ & ob{:}set\ Ob,\ op{:}set\ Op,\ p{:}set\ Perm,\ u{:}set\ User,\ r{:}set\ Role, \\ & ua{:}set\ UA_{elem},\ pa{:}set\ PA_{elem},\ (\ll){:}Role \to Role \to Prop, \\ & seID{:}set\ Session,\ se{:}Session \to SE_{elem}\ ]\!] \end{aligned}$$

donde ob es el conjunto de objetos sobre el cual se pueden realizar operaciones; op es el conjunto de estas operaciones; p es el conjunto de permisos; u es el conjunto de usuarios válidos; r es el conjunto de roles; ua es el conjunto de asignaciones usuario-rol; pa es el conjunto de asignaciones permiso-rol; $\ll$ es la relación de herencia inmediata entre roles; $seID$ es el conjunto de identificadores de sesión de la sesiones establecidas en el sistema; y se es la función, descripta previamente, que mapea cada identificador de sesión con su dueño y roles activos.

Propiedades de Validez de los Estados. No todo elemento de tipo *State* es un estado válido, para ello es necesario que sus componentes verifiquen ciertas condiciones. Para caracterizarlas formalmente se definen propiedades de validez, que todo estado RBAC $s : State$ debe cumplir. A continuación se detallan sólo algunas de las más relevantes.

"Toda sesión tiene un dueño, que además es miembro del conjunto de usuarios"

$$\begin{aligned} existsSessionOwner\ s \stackrel{\text{def}}{=} \forall\ sid{:}Session, \\ isSession\ s\ sid \rightarrow \exists\ u{:}User,\ isUser\ s\ u \wedge session\_owner\ s\ u\ sid \end{aligned}$$

"El usuario dueño de una sesión es único"

$$\begin{aligned} uniqueSessionOwner\ s \stackrel{\text{def}}{=} \forall\ (sid{:}Session)(u_1\ u_2{:}User), isSession\ s\ sid \\ \wedge session\_owner\ s\ u_1\ sid \wedge session\_owner\ s\ u_2\ sid \rightarrow u_1 = u_2 \end{aligned}$$

"Los roles activos de una sesión son un subconjunto del conjunto de roles para los que el usuario dueño de la misma está autorizado"

$$\begin{aligned} activeSessionRoles\ s\ \stackrel{\text{def}}{=} \forall\ (sid{:}Session)(u{:}User)(r{:}Role),\ isSession\ s\ sid \\ \wedge session\_owner\ s\ u\ sid \wedge role\_of\_session\ s\ r\ sid \rightarrow authorized\_user\ s\ u\ r \end{aligned}$$

"La relación entre roles ($\preceq$) que modela la jerarquía de roles es un orden parcial"

$$\begin{aligned} isOrder\ (\preceq) \stackrel{\text{def}}{=} \forall\ x,\ x \preceq x \\ \wedge \forall\ x\ y,\ x \preceq y \wedge y \preceq x \rightarrow x = y \\ \wedge \forall\ x\ y\ z,\ x \preceq y \wedge y \preceq z \rightarrow x \preceq z \end{aligned}$$

"Los roles pertenecientes a la jerarquía de roles son un subconjunto del conjunto de roles del sistema"

$$H\_integrity\ s \stackrel{\text{def}}{=} \forall\ r_1\ r_2{:}Role,\ r_1 \ll r_2 \rightarrow isRole\ s\ r_1 \wedge isRole\ s\ r_2$$

Finalmente, un estado s es válido si cumple la condición $Valid$ [9]:

$$\begin{aligned} Valid\ s \stackrel{\text{def}}{=} existsSessionOwner\ s \wedge uniqueSessionOwner\ s \\ \wedge activeSessionRoles\ s \wedge isOrder\ (\preceq)\ s \wedge UA\_integrity\ s \\ \wedge Perm\_integrity\ s \wedge PA\_integrity\ s \wedge H\_integrity\ s \end{aligned}$$

5.3. Comandos

El estado del sistema evoluciona mediante la ejecución de comandos, cuya sintaxis está definida por el tipo inductivo no recursivo $Funcs$, y cuya su semántica está especificada mediante *pre* y *pos* condiciones. Para cada comando (más de 20 fueron considerados), su precondición es un predicado sobre el estado, y su

postcondición es un predicado que relaciona al estado previo y al posterior a la ejecución del mismo, con una posible respuesta del sistema (de tipo *Answer*). Ambas relaciones estan definidas inductivamente, utilizando un constructor para cada comando. Por razones de espacio sólo se indican sus tipos a continuación:

$$Pre : State \to Funcs \to Prop$$

$$Pos : State \to State \to Answer \to Funcs \to Prop$$

Errores. Cuando se intenta ejecutar un comando cuya precondición no se satisface, el sistema responde con un código de error de tipo *ErrorCode*. Para cada comando f se define la relación $ErrorMsg\ f$, que vincula un estado sobre el cual no se verifica la precondición de f con un código de error.

Ejecución de los Comandos. La ejecución de un comando f en un estado s produce un nuevo estado s' y una respuesta ans, donde la relación entre el estado previo, el nuevo y la respuesta está determinada por la postcondición de f. Se define el tipo $exec : State \to Funcs \to Answer \to State \to Prop$, donde:

$$\text{exec\_pre}\ \frac{Pre\ s\ f \qquad Pos\ s\ s'\ ans\ f}{exec\ s\ f\ ans\ s'}$$

$$\text{exec\_npre}\ \frac{\neg\ (Pre\ s\ f) \qquad \exists\ ec,\ s = s' \wedge ErrorMsg\ s\ f\ ec \wedge ans = error\ ec}{exec\ s\ f\ ans\ s'}$$

Notar que si $Pre\ s\ f$ no se cumple entonces el estado s no cambia y la respuesta del sistema es el mensaje de error determinado por la relación $ErrorMsgf$.

5.4. Verificación

La verificación de la especificación se realizó en dos etapas. Primero se probó la invarianza de la validez de los estados respecto a la ejecución de los comandos, esto es, que la semántica de los comandos preserva la propiedad $Valid$ (teorema 1). Luego se demostraron propiedades relevantes para los modelos formalizados. A modo de ejemplo se incluyen 2 lemas. Para ver todas las propiedades analizadas, junto con sus pruebas, referirse a [9].

Teorema 1 $\forall(s{:}State)(f{:}Func), Valid\ s \wedge Pre\ s\ f \wedge Pos\ s\ s'\ ok\ f \to Valid\ s'$

La invarianza de $Valid$ garantiza que al ejecutarse cualquier comando se obtiene un estado en el cual se cumple que cada una de las sesiones establecidas poseen un único usuario válido dueño de las mismas. Es importante además demostrar que este usuario es el mismo que en el estado previo a la ejecución del comando, siempre y cuando éste no termine la sesión. La propiedad de intransferibilidad de las sesiones se demuestra en el siguiente lema:

Lema 1 $\forall(u{:}User)(sid{:}Session)(f{:}Funcs), isSession\ s\ sid \wedge$
$session\_owner\ s\ u\ sid \wedge exec\ s\ f\ ok\ s' \wedge isSession\ s'\ sid \to session\_owner\ s'\ u\ sid$

Una propiedad fundamental que debe cumplir la formalización respecto a la jerarquía de roles es que ésta modele correctamente el concepto de herencia. El segundo lema establece que, no puede suceder que un usuario asignado a roles de mayor jerarquía que otro no esté autorizado para un rol cuando el segundo si lo está.

Lema 2 $\forall(u_1\ u_2{:}User)(r\ r_2{:}Role)$,
$(\forall(r_1{:}Role), role\_of\_user\ s\ r_1\ u_1 \rightarrow r_1 \preceq r_2) \wedge authorized\_user\ s\ u_2\ r_2 \rightarrow$
$\neg(authorized\_user\ s\ u_1\ r \wedge \neg\ authorized\_user\ s\ u_2\ r)$

6. Conclusiones y Trabajos Futuros

El análisis realizado de los modelos RBAC considerados en el estándar NIST puso en evidencia la existencia de problemas en torno a la especificación propuesta, que por otra parte no incluye un análisis de su corrección, ni la verificación de propiedades de seguridad.

Este trabajo introdujo una nueva especificación formal de *Core RBAC* y *Hierarchical RBAC*, desarrollada en el CCI, que subsana los problemas encontrados. En particular, esta formalización posibilitó el estudio en profundidad de alternativas libradas a decisiones de implementación por parte del estándar. Se mostró que para administrar la jerarquía de roles, cualquier intento de manipular las herencias indirectas de manera explícita provoca, en algunos casos, la pérdida de la propiedad de orden parcial de la relación que modela la jerarquía y, en otros casos, conduce a la subutilización de la transitividad del orden. Asimismo, en este artículo se pusieron de manifiesto ambigüedades, imprecisiones y omisiones en los requerimientos funcionales de algunos comandos incluidos en el estándar NIST. Por ejemplo, en la especificación del comando *DeleteRole* se omite la presencia de una jerarquía de roles, hecho que permite al sistema transicionar a un estado inválido, donde existan relaciones de herencia entre roles que no pertenecen al conjunto de roles del sistema.

Contar con un especificación en el CCI permitió razonar de manera abstracta sobre la corrección de los modelos RBAC formalizados, usando Coq. Por un lado, se verificó que la semántica de los comandos conserva invariante la validez de los estados, propiedad que garantiza que el sistema no puede transicionar a un estado que no represente un posible estado RBAC. Por otro lado, se demostraron dos propiedades, a modo de ejemplo, relevenates para todo modelo RBAC: la intransferibilidad de las sesiones y el correcto comportamiento de la jerarquía de roles. Debido a limitaciones de espacio se omitieron muchos detalles sobre la nueva especificación introducida en este artículo, como así también propiedades verificadas, junto con sus pruebas. La formalización completa consta de 14 módulos (archivos) Coq con 1525 líneas en total, que incluyen 87 definiciones, 27 lemas y 8 teoremas. El lector interesado puede referirse a [9] por más información.

Si bien éste no es el primer artículo que analiza el estándar NIST referido, se diferencia de los anteriores en que plantea una nueva formalización, en una

teoría de tipos, que permite verificar formalmente propiedades de seguridad relevantes del modelo, usando un asistente de pruebas como Coq. Dado que el formalismo utilizado permite obtener especificaciones *certificadas* de sistemas, se plantea como trabajo futuro la extracción de un prototipo funcional correcto por construcción de la formalización de los modelos RBAC, que sirva como implementación de referencia, u oráculo, para desarrolladores e investigadores. Previo a esto, se propone desarrollar una extensión de la especificación en el CCI para incorporar los modelos RBAC SSD y DSD del estándar NIST. Finalmente, resulta también de interés extender la formalización para permitir razonar sobre secuencias de estados, contemplando, en particular, el análisis de propiedades temporales.

Referencias

[1] A. E. Abdallah and E. J. Khayat. Formal z specifications of several flat role-based access control models. *Software Engineering Workshop, Annual IEEE/NASA Goddard*, 0:282–292, 2006.

[2] R. Baldwin. Naming and grouping privileges to simplify security management in large databases. In *IEEE Symposium on Security and Privacy*, pages 116–132, 1990.

[3] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development. CoqÁrt: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer-Verlag, 2004.

[4] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed nist standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, 2001.

[5] J. Joshi, A. Ghafoor, W. G. Aref, and E. H. Spafford. Digital government security infrastructure design challenges. *Computer*, 34(2):66–72, 2001.

[6] J. B. D. Joshi, W. G. Aref, A. Ghafoor, and E. H. Spafford. Security models for web-based applications. *Commun. ACM*, 44(2):38–44, 2001.

[7] N. Li, J.-W. Byun, and E. Bertino. A critique of the ansi standard on role-based access control. *IEEE Security and Privacy*, 5(6):41–49, 2007.

[8] N. Li and M. V. Tripunitara. Security analysis in role-based access control. *ACM Trans. Inf. Syst. Secur.*, 9(4):391–420, 2006.

[9] C. D. Rosa. Especificación formal del modelo RBAC en el cálculo de construcciones inductivas. Master's thesis, U.N.Rosario, Argentina, 2008.

[10] R. Sandhu, D. Ferraiolo, and R. Kuhn. The nist model for role-based access control: Towards a unified standard. In *In Proceedings of the fifth ACM workshop on Role-based access control*, pages 47–63. ACM Press, 2000.

[11] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[12] The Coq Development Team. *The Coq Proof Assistant Reference Manual – Version V8.1*, 2006.

[13] D. J. Thomsen. Role-based application design and enforcement. *Database Security, IV: Status and Prospects*, 1990.

[14] T. C. Ting, S. A. Demurjian, and M.-Y. Hu. Requirements, capabilities, and functionalities of user-role based security for an object-oriented design model. In *Results of the IFIP WG 11.3 Workshop on Database Security V*, pages 275–296, Amsterdam, The Netherlands, 1992. North-Holland Publishing Co.