Patrón de seguridad para la autorización de bases de datos NoSQL (par clave-valor)

Julio Moreno[1], Eduardo B. Fernandez[2], Manuel A. Serrano[3],
Eduardo Fernández-Medina[1]

[1] Grupo de investigación GSyA, Universidad de Castilla la Mancha, Ciudad Real, España
[2] Dept. of Computer and Elec. Eng. and Computer Science, Florida Atlantic University, USA
[3] Grupo de investigación Alarcos, Universidad de Castilla la Mancha, Ciudad Real, España
{julio.moreno | manuel.serrano | eduardo.fdezmedina}@uclm.es
ed@cse.fau.edu

Resumen. Aunque las bases de datos de tipo NoSQL surgieron hace unos años, son cada vez más usadas en diversos contextos, debido al crecimiento de los sistemas web y su uso en sistemas analíticos como Big Data. Sin embargo, generalmente este tipo de sistemas de almacenamiento no ha sido concebido con la seguridad en mente, pues se centran en resolver otros problemas como la velocidad de acceso o su uso en entornos distribuidos. Uno de los principales problemas de seguridad que se pueden identificar en estos entornos es la falta de un mecanismo de control de acceso nativo. Aunque existen numerosas propuestas hechas por investigadores que solucionan esta problemática para una tecnología concreta, faltan propuestas con un mayor nivel de abstracción que propongan soluciones más generales. En este sentido, una solución para este tipo de problemas generales es la creación de un patrón de seguridad. En este artículo proponemos un patrón de seguridad específico para realizar el control de acceso de bases de datos NoSQL de tipo clave-valor y se definen los diferentes elementos que lo conforman.

Palabras clave: NoSQL, Seguridad de la información, Patrones de Seguridad.

1 Introducción

En los últimos años, los sistemas NoSQL están experimentando un incremento en popularidad. Sin embargo, hay muchos problemas de seguridad que pueden afectarlos. Los principales problemas se relacionan con la falta de encriptación de los datos, el débil sistema de autenticación entre los servidores y el cliente, la posibilidad de realizar ataques como inyección de código SQL o la denegación de servicio (DOS y DOSS) y la sencillez de los mecanismos de autorización, los cuales, no suelen soportar de forma nativa el control de acceso basado en roles o autorización de grano fino a los datos [1].

Se han propuesto una gran cantidad de modelos de autorización para bases de datos relacionales, normalmente estas soluciones se basan en el uso de vistas. Por otro lado, la mayoría de bases de datos NoSQL para aplicaciones Big Data usan un nuevo modelo más apropiado a sus capacidades de gestionar datos no estructurados, que pueden llegar

en tiempo real y en gran cantidad. Este modelo protege cada una de las celdas individuales en bases de datos de tipo clave-valor; cada registro es etiquetado con derechos de autorización siguiendo un modelo de control de acceso basado en roles o similar. Todos los ejemplos conocidos se derivan del modelo de seguridad de Apache Accumulo, el cual a su vez, lo recibió de un proyecto de la National Security Alliance (NSA). El trabajo que presentamos en este artículo busca abstraer mediante la creación de un patrón este modelo que se puede encontrar en numerosos entornos Big Data.

Antes de la aparición de estos modelos, las bases de datos NoSQL solo usaban un control de acceso basado en fichero, el cual, daba al usuario acceso a todo o a ninguno de los datos almacenados. Obviamente, esta aproximación es demasiado rudimentaria puesto que no permite combinar diferentes accesos en función de si el fichero al que se quiere acceder contiene o no información sensible [2]. Apache Accumulo se ha convertido en un estándar *de facto* y, como tal, puede ser la base sobre la cual definir un patrón para mejorar la seguridad de bases de datos de tipo clave-valor. Además, algunas bases de datos NoSQL carecen de soporte nativo para Apache Accumulo, lo cual, complica la posibilidad de obtener control de acceso a nivel de celda.

El resto del trabajo se estructura de la siguiente forma: primero, una sección de trabajo relacionado, en la cual, se explican los conceptos de patrón de seguridad y algunos de los patrones que se pueden encontrar en la literatura y se encuentran relacionados con el nuestro. A continuación, se define nuestro patrón para la autorización de bases de datos NoSQL incluyendo su objetivo, su contexto, la problemática que resuelve, las restricciones que tiene y ayudas para su implementación. Finalmente, se incluye una sección con las conclusiones y trabajo futuro.

2 Trabajo relacionado

En esta sección se explica el concepto de bases de datos NoSQL, su importancia y sus principales problemas de seguridad. Por otro lado, se define qué es un patrón de seguridad, así como su uso y relevancia.

Las bases de datos relacionales que siguen un formato entidad-relación eran suficientes para gestionar los datos generados por los sistemas de información tradicionales, pero en la actualidad, existe una enorme cantidad de datos provenientes de sistemas multimedia y de redes sociales. Esta serie de características hace que sea complicado almacenar dichos datos de forma eficiente, como solución surgieron las bases de datos NoSQL. En resumen, se puede decir que las bases de datos NoSQL sirven para almacenar datos en situaciones en las cuales las bases de datos presentan problemas de escalabilidad y rendimiento [3]. Existen diferentes formas de clasificar las bases de datos NoSQL, pero la más aceptada define cuatro categorías: i) almacenamiento clave-valor (los datos se almacenan en una tabla hash con formato clave-valor, se pueden lanzar consultas rápidamente utilizando la clave), ii) almacenamiento documental (se encuentran diseñadas para almacenar los datos con un formato similar a documentos como

JSON), iii) columnares (los datos se almacenan en filas y columnas, las cuales, se pueden agrupar en familias columnares. Es una forma de distribuir y almacenar grandes cantidades de datos) y iv) basadas en grafos (se utilizan para almacenar formación que tiene un formato similar a un grafo, como las relaciones entre amigos en una red social) [4].

Los mecanismos y técnicas para mejorar la seguridad de las bases de datos NoSQL se encuentran todavía en desarrollo. Cada tipo de base de datos NoSQL fue diseñada para solucionar un problema analítico distinto, motivo por el cual, la seguridad no fue considerada durante su etapa de diseño. Como forma de mejorar la seguridad, algunos desarrolladores incluyen mecanismo en el middleware que lo gestiona. Los principales problemas de seguridad que se pueden identificar en una base de datos NoSQL son la privacidad de los datos que almacenan y cómo realizar el control de acceso [5]. En cuanto a la privacidad, normalmente este problema se soluciona utilizando mecanismos de encriptación [6]. Por otro lado, para el control de acceso, los investigadores han propuesto diversas soluciones: desde considerar el control de acceso como un servicio aparte de la base de datos NoSQL [7], pasando por aplicar un esquema de control de accesos basado en roles (RBAC) encriptado que se encuentra adaptado a las necesidades de este tipo de sistemas [8] o mejorar el esquema RBAC para permitir un acceso granular a tecnologías específicas como MongoDB [9]. El problema surge cuando se observa que la mayoría de las propuestas se encuentran enfocadas en solucionar un problema de seguridad de una tecnología concreta, no suelen encontrarse soluciones con un mayor nivel de abstracción que puedan usarse en cualquier base de datos NoSQL. Esto complica mucho el trabajo de los desarrolladores. Una forma de solucionar este problema es mediante el uso de patrones de seguridad.

Los patrones son mecanismos que permiten solucionar problemas recurrentes haciendo uso de la abstracción de sus causas principales. Así, un patrón de seguridad permite a los desarrolladores aplicar medidas de seguridad en sus sistemas, sin necesidad de que sean expertos en la materia [10]. Desde su creación, los patrones de diseño se han convertido en un mecanismo muy popular en el área de ingeniería de software a la hora de proteger sistemas contra amenazas y situaciones de mal uso. Cuando se define un patrón es común utilizar la siguiente estructura: intención (el problema que se desea resolver con dicho patrón), contexto (la situación general, en la cual, se aplica el patrón), problema (pequeño enunciado en el que se expresa el problema a resolver), fuerzas (las dificultades o preocupaciones a considerarse a la hora de definir el patrón), solución (la forma recomendada de resolver el problema en un contexto dado), implementación (pistas o ayudas para implementar el patrón), usos conocidos (mínimo tres usos en sistemas reales), consecuencias (ventajas y desventajas de la solución) y patrones relacionados (patrones que resuelven un problema similar o son complementarios al patrón definido) [11].

3 Patrón autorizador de bases de datos NoSQL

El patrón autorizador pretende solucionar los problemas de control de acceso típicos de las bases de datos clave-valor. En esta sección, se va a definir los diferentes elementos que lo forman, incluyendo la solución, para ello se seguirá la estructura típica de definición de patrones.

3.1 Intención

El patrón busca describir quién puede acceder a cada una de las celdas de una base de datos NoSQL de tipo clave-valor. Cada una de estas celdas puede contener elementos con información sensible cuyo acceso debe ser controlado. Es probable que junto a estos datos sensibles se encuentre información no sensible, es importante recalcar en este punto que este tipo de sistemas puede implementarse dentro de un contexto Big Data, en el cual, se puede inferir información sensible a partir de este tipo de datos. Por lo cual, en este tipo de ecosistemas, es interesante añadir una serie de restricciones de acceso.

3.2 Contexto

El contexto objetivo en el que puede ser aplicado este patrón es el de bases de datos muy grandes (VLDB) en las que la información es almacenada utilizando la técnica de pares clave - valor y, que por sus características inherentes, la información sensible puede estar entremezclada con la información ordinaria y no sensible.

3.3 Problema

Las bases de datos relacionales suelen proteger los datos utilizando vistas que restringen el acceso a los distintos usuarios o roles. Estas vistas se definen utilizando sentencias SQL. Este mecanismo no es possible usarlo en bases de datos NoSQL donde no existe el concepto de vista. La falta de este mecanismo provoca que se suelan implementar controles de acceso basados en ficheros con permisos, lo cual, tiene como consecuencia un acceso de grano grueso a los datos.

3.4 Fuerzas

Esta solución está restringida por las siguientes fuerzas:

- *Granularidad del control de acceso.* La intención del patrón es conseguir controlar cada celda de datos (cada par clave, valor).
- *Flexibilidad.* Diferentes tipos de usuarios pueden requerir diferentes modelos de autorización.
- *Velocidad.* Una de las principales razones para la utilización de bases de datos es la velocidad, en este sentido, la comprobación de la autorización debe ser rápida.

- *Registro (log) y Auditoria.* Debido a la sensibilidad del acceso a los datos, el acceso a estos debe ser registrado y auditado posteriormente para poder asegurar el cumplimiento de las leyes y la prevención de usos inadecuados.

3.5 Solución

La solución propuesta es etiquetar cada celda de valor (fila) con información de autorización. La figura 1 muestra una celda de datos típica, siguiendo el modelo de Accumulo, el cual, es considerado como un estándar de facto para la autorización de celdas [12]. En Accumulo, los datos se almacenan en una correspondencia (map) distribuida y ordenada, en la que las clases se dividen lógicamente en diferentes partes:

- *RowID*: es un identificador único de la fila
- *Column*: que, a su vez, puede ser dividida en 3 partes:
    - Family es la agrupación lógica de la clave,
    - Qualifier es utilizado para indicar un atributo más específico de la clave
    - Visibility almacena una combinación lógica de etiquetas de seguridad que deben ser satisfechas en el momento de la búsqueda. Esto es necesario para que la clave y el valor sean devueltos como parte de la respuesta al usuario, permitiendo que se puedan almacenar diferentes requisitos de seguridad en la misma tabla. Solo permite el acceso al usuario de las claves y valores a los que tiene permiso.
- *Timestamp*: es una marca de tiempo que se genera automáticamente y es utilizada para el control de versiones.

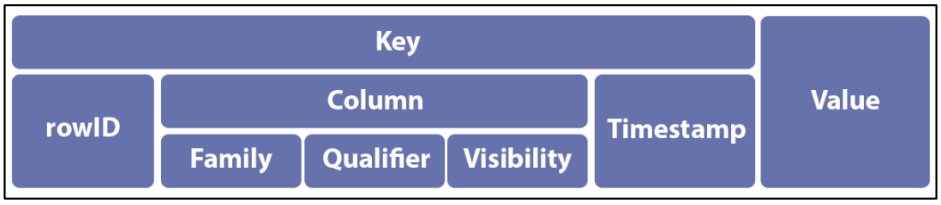

Figura 1. Estructura de una celda de datos

3.5.1 Estructura

Para la estructura del patrón se ha realizado una ampliación del estándar autorizador de [10], para ello, se ha añadido un “Sujeto” que representa una entidad activa (un usuario o un sistema) que se identifica con el servidor web utilizando un “Autenticador”. Una vez el sujeto se encuentra autenticado este ofrece un catálogo de datos utilizando “Sentencias Cliente”. Estos datos a los que se quiere acceder son autorizados por el componente “Autorizador” (proveniente del estándar Autorizador), el cual, devuelve

los permisos que se disponen para acceder a diferentes partes de la base de datos. La estructura del patrón se puede observan en el diagrama de clases de la Figura 2.

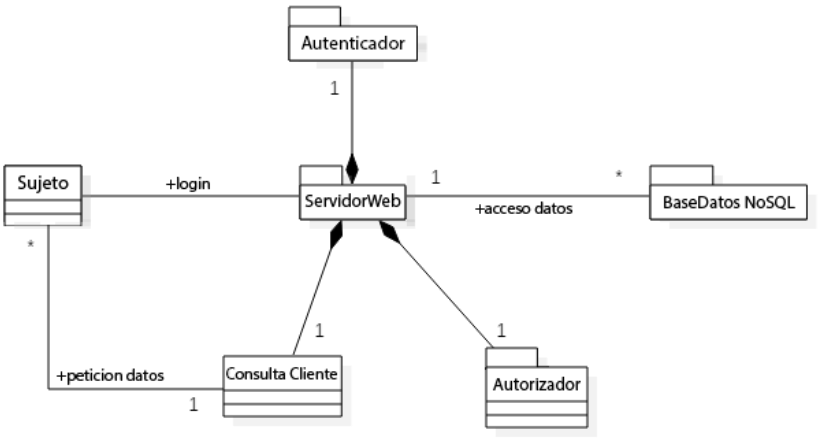

Figura 2. Diagrama de clases para el patrón autorizador de bases de datos NoSQL.

3.5.2 Dinámicas

Para mejorar la comprensión del patrón de seguridad, se incluye esta subsección, en la cual, se explica un escenario normal de ejecución del mismo. La Figura 3 muestra el caso de uso "Acceder a un elemento de dato". Este caso de uso sigue la siguiente secuencia:

1. Autenticar usuario.
2. Enviar el rango de claves requerido.
3. Pasar la prueba de autenticación al servicio de Autorización.
4. Devolver el uso de derechos.
5. Consultar el rango de claves a la base de datos.
6. Recibir los pares clave-valor solicitados del rango.
7. Devolver los valores al usuario.

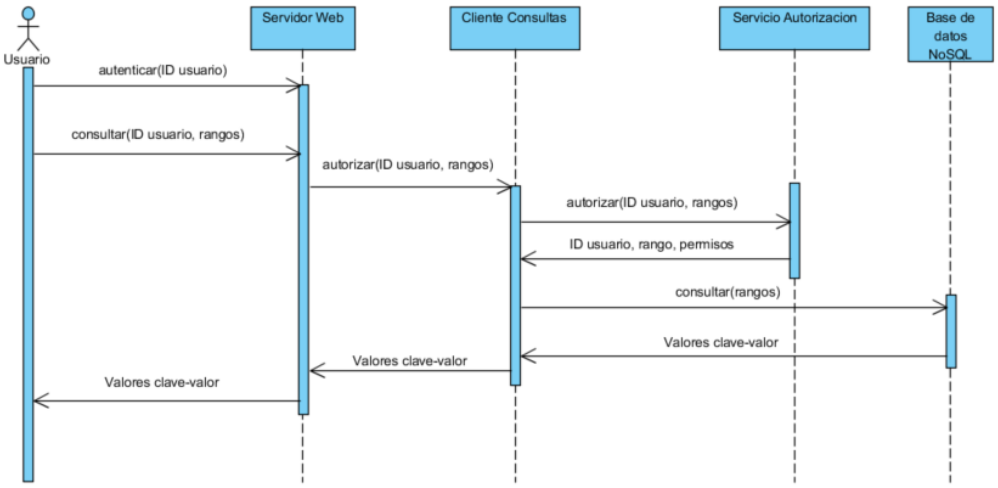

Figura 3. Diagrama de secuencia para el caso de uso "Acceder al elemento del dato"

3.6 Implementación

De cara a tener una implementación real del patrón, se propone una estructura tal y como se muestra en el diagrama de clases de la Figura 4. En este diagrama de clases, se puede observar que un Servidor Web dedicado al acceso a los datos de una Base de Datos NoSQL, necesitará un Cliente de Consultas, en el que se delegará la autorización de cada usuario para cada rango de datos, en una tercera clase que indicará al Cliente de Consultas que rangos de tuplas clave-valor pueden ser devueltos al usuario y en base a esta autorización se realizará la consulta a la Base de Datos NoSQL y se devolverán los datos permitidos al usuario.

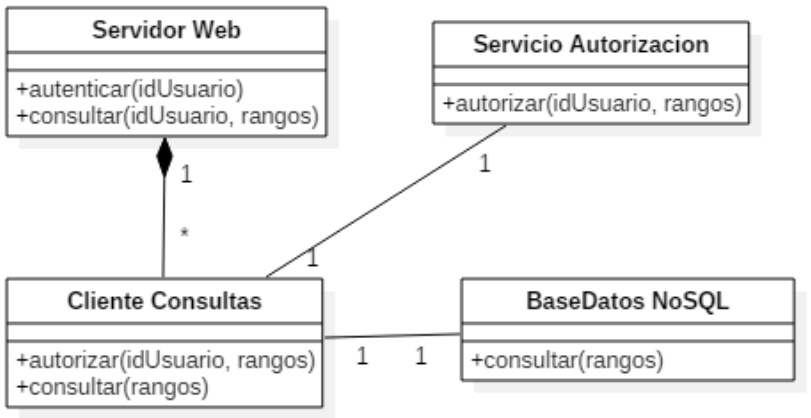

Figura 4. Diagrama de clases propuesta para la implementación

3.7 Usos conocidos

En esta subsección se describen algunos casos de uso que implementan una solución tecnológica similar a nuestra propuesta de patrón de seguridad:

- *Apache Accumulo [13]*. Accumulo es un sistema de almacenamiento de tipo clave-valor basado en el diseño de Big Table de Google. Accumulo almacena los datos en el HDFS de Hadoop y utiliza Apache Zookeeper para crear consenso. Mientras que los usuarios pueden interactuar directamente con Accumulo, numerosas bases de datos usan Accumulo como una capa extra inferior. Cada par clave-valor de Accumulo tiene sus propias etiquetas de seguridad que obtienen los resultados de sus consultas dependiendo de los permisos de autorización. Apache Accumulo extiende el modelo de datos de Big Table de Google, añadiendo un nuevo elemento a la clave denominado visibilidad de la columna. Este elemento guarda una combinación lógica de etiquetas de seguridad que deben ser satisfechas en tiempo de consulta para devolver el par clave-valor solicitado como parte de la respuesta al usuario. Esto permite tener datos con diferentes requisitos de seguridad almacenados en la misma tabla, y además permite a los usuarios acceder solo a aquellas claves y valores que tienen permiso.
- *Koverse [12]*. Realiza una implementación del modelo de seguridad de Apache Accumulo.
- *SQRRL [14]*. Al igual que Koverse, realiza una implementación del modelo de seguridad de Apache Accumulo.
- *Blue Talon [15]*. Blue Talon afirma que provee seguridad para cualquier tipo de base de datos, incluyendo las NoSQL, pero no muestran qué modelo de seguridad utilizan.

3.8 Consecuencias

En esta sección se describen las diferentes ventajas y desventajas derivadas de la implementación de nuestro patrón. Así, en nuestro patrón se pueden identificar las siguientes ventajas:

- *Granularidad en el control de acceso,* Gracias al uso de nuestro patrón es posible controlar el acceso a cada celda de datos (par clave-valor). Esto es equivalente al control de acceso dependiente del contexto.
- *Flexibilidad.* Las etiquetas pueden definir las diferentes reglas para acceder a matriz, el control de acceso basado en roles o modelos multinivel.
- *Velocidad.* Comprobar los permisos de autorización requiere una simple operación de coincidencia entre registros, lo cual, se puede conseguir de una forma eficiente.

- *Auditoría y registro.* Las etiquetas pueden ser usadas para controlar cuánta información se obtiene en cada acceso.

Por otro lado, de la implementación de nuestro patrón también se pueden derivar una serie de desventajas que incluyen:

- *Trabajo extra.* Para aprovechar los beneficios potenciales de la aplicación del patrón, los desarrolladores deben relacionar las etiquetas con las celdas de datos para protegerlos en función de sus necesidades.
- *Velocidad.* Si bien se ha indicado que nuestra propuesta no supone un incremento sustancial en la carga de procesamiento, también es cierto que la sobrecarga que se provoca el comprobar los permisos en cada consulta puede provocar un descenso en la velocidad de la aplicación si dichas consultas se realizan frecuentemente.
- *Inferencia de datos.* Un aspecto que no contempla este patrón es la posible inferencia de datos que se puede realizar en contextos Big Data. Debido a las capacidades analíticas de sus algoritmos es posible extraer información sensible a partir de datos no sensibles.

3.9 Patrones relacionados

Finalmente, en esta subsección se definen diferentes patrones de seguridad que cubren aspectos similares al nuestro o que se encuentran incorporados de alguna forma en nuestra propuesta:

- *Autenticador [10]*. Cuando una entidad activa como un usuario o un sistema (sujeto) se identifica con el sistema, ¿cómo se verifica que el sujeto que intenta acceder al sistema es quien dice ser? Presentando información que es reconocida por el sistema e identifica al sujeto. Después de ser reconocido, el sujeto recibe algún tipo de prueba de que ha sido autenticado.
- *Autorizador [10]*. Describe quien está autorizado para acceder a recursos específicos de un sistema en un entorno, en el cual, se tienen recursos cuyo acceso debe ser controlado. Indica para cada entidad activa, a qué recursos puede acceder y qué puede hacer con ellos.
- *Monitor de referencia [10]*. Este patrón se usa en entornos computacionales, en los cuales, los usuarios o procesos hacen peticiones por datos o recursos. Así, este patrón mejora las restricciones de acceso cuando una entidad realiza este tipo de peticiones. Para ello, describe un proceso abstracto que intercepta todas las peticiones de recurso y comprueba que se cumplen las autorizaciones.
- *Control de acceso dependiente del contenido [16]*. Este patrón responde a la pregunta de cómo restringir el acceso (lectura, escritura o borrado) de los diferentes sujetos (usuarios, sistemas u otras partes involucradas) a solo datos con valores específicos. Para ello, filtra los valores obtenidos tras aplicar las reglas de autorización a los datos de acuerdo al predicado (condición) que filtra los valores específicos.

- *Registro de seguridad/Auditor [10]*. Este patrón busca llevar una traza de las acciones del usuario para determinar quién hizo qué y cuándo lo hizo. Registrar todas las acciones de seguridad, que pueden incurrir en incumplimiento de reglas para datos sensitivos, ayuda a realizar un mejor control del acceso a los recursos que pueden ser útiles para procesos de auditoría.

4 Conclusiones y trabajo futuro

En este artículo presentamos nuestra propuesta de patrón para realizar el control de acceso en bases de datos NoSQL de tipo clave-valor. Este patrón es una especialización de un patrón abstracto de autorización. En un entorno de tipo Big Data, las bases de datos relacionales coexisten con bases de datos no estructuradas y NoSQL. Como cada base de datos proviene de un proveedor diferente, sus modelos de autorización no se encuentran coordinados. Esto es crítico en este tipo de contextos, puesto que los mismos datos (o relacionados) pueden aparecer en distintas formas en diferentes tipos de bases de datos.

Además de este tipo de bases de datos NoSQL existen otros tipos como las basadas en grafos, en columnas y en documentos. Como trabajo futuro nos planteamos la posibilidad de diseñar diferentes patrones de seguridad para el control de acceso a cada uno de estos sistemas, como objetivo a largo plazo se tiene la integración de un patrón global que permita el control de acceso a cualquier base de datos NoSQL. Por otro lado, nos planteamos la opción de tratar los inconvenientes de esta propuesta, como la inferencia de datos, para ello, hemos planteado la posibilidad de integrar este patrón con el patrón "Registro de seguridad/auditor". También consideramos importante definir en mayor profundidad los detalles de implementación de este patrón de forma que se relacione con el contexto de una arquitectura de referencia para Big Data [17].

Agradecimientos

Este trabajo ha sido financiado por el proyecto SEQUOIA (Ministerio de Economía y Competitividad y Fondo Europeo de Desarrollo Regional FEDER, TIN2015-63502-C3-1-R) y el proyecto GENESIS (Consejería de Educación, Cultura y Deportes de la Dirección General de Universidades, Investigación e Innovación de la JCCM).

Referencias

1. Okman, L., Gal-Oz, N., Gonen, Y., Gudes, E., Abramov, J.: Security issues in nosql databases. In: Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on. pp. 541–547. IEEE (2011).
2. Colombo, P., Ferrari, E.: Fine-Grained Access Control Within NoSQL Document-Oriented Datastores. Data Sci. Eng. 1, 127–138 (2016).

3. Acens: Bases de datos NoSQL. Qué son y tipos que nos podemos encontrar. (2014).
4. Tang, E., Fan, Y.: Performance Comparison between Five NoSQL Databases. In: 2016 7th International Conference on Cloud Computing and Big Data (CCBD). pp. 105–109 (2016).
5. Cloud Security Alliance (CSA), B.D.W.G.: Expanded Top Ten Big Data Security and Privacy, https://downloads.cloudsecurityalliance.org/initiatives/bdwg/Expanded_Top_Ten_Big_Data_Security_and_Privacy_Challenges.pdf.
6. Macedo, R., Paulo, J., Pontes, R., Portela, B., Oliveira, T., Matos, M., Oliveira, R.: A Practical Framework for Privacy-Preserving NoSQL Databases. In: 2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS). pp. 11–20 (2017).
7. Habiba, M., Islam, M.R., Ali, A.B.M.S.: Access control management as a service for NoSQL big data. Presented at the 2015 2nd Asia-Pacific World Congress on Computer Science and Engineering, APWC on CSE 2015 (2016).
8. Shalabi, Y., Gudes, E.: Cryptographically enforced role-based access control for NoSQL distributed databases. Lect. Notes Comput. Sci. Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinforma. 10359 LNCS, 3–19 (2017).
9. Colombo, P., Ferrari, E.: Enhancing MongoDB with Purpose-Based Access Control. IEEE Trans. Dependable Secure Comput. 14, 591–604 (2017).
10. Fernandez, E.B.: Security patterns in practice: designing secure architectures using software patterns. John Wiley & Sons (2013).
11. Bunke, M.: Software-security Patterns: Degree of Maturity. In: Proceedings of the 20th European Conference on Pattern Languages of Programs. pp. 42:1–42:17. ACM, New York, NY, USA (2015).
12. Cordova, A., Rinaldi, B., Wall, M.: Accumulo: Application Development, Table Design, and Best Practices. O'Reilly Media, Inc. (2015).
13. Apache Accumulo, http://accumulo.apache.org/.
14. SQRRL: Big Data and Data Centric Security. (2014).
15. BlueTalon: BlueTalon Data-Centric Security Platform: Bringing Order to Data Security Chaos. (2016).
16. Fernández, E.B., Monge, A.R., Carvajal, R., Encina, O., Hernández, J., Silva, P.: Patterns for content-dependent and context-enhanced authorization. In: Proceedings of the 19th European Conference on Pattern Languages of Programs. p. 32. ACM (2014).
17. Moreno, J., Serrano, M.A., Fernandez-Medina, E., Fernandez, E.B.: Towards a security reference architecture for big data. Presented at the CEUR Workshop Proceedings (2018).