ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/268047335

Arquitectura de un Sistema Multiagente para la Clasificación de Consultas con Inyección SQL

Article

CITATIONS
4

READS
546

2 authors:

Cristian Pinzon
Universidad Tecnológica de Panamá
35 PUBLICATIONS **206** CITATIONS

SEE PROFILE

Juan Manuel Corchado Rodríguez
Universidad de Salamanca
832 PUBLICATIONS **12,471** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

MOVIURBAN View project

DREAM-GO Project View project

All content following this page was uploaded by Juan Manuel Corchado Rodríguez on 19 March 2015.

The user has requested enhancement of the downloaded file.

Arquitectura de un Sistema Multiagente para la Clasificación de Consultas con Inyección SQL

Cristian I. Pinzón y Juan M. Corchado

Departamento de Informática y Automática,
Universidad de Salamanca, Salamanca, España
{cristian_ivanp, corchado}@usal.es

Resumen En este artículo se presenta una arquitectura multiagente para la clasificación de consultas con inyección SQL en aplicaciones Web. La arquitectura integra la clasificación de consultas SQL, la monitorización de las acciones de los usuarios y la extracción de conocimiento a partir de una bitácora de consultas. A través de estos mecanismos, se logrará un proceso de realimentación que permita perfeccionar la estrategia de clasificación y mejorar la seguridad de la aplicación Web. Esta arquitectura aportará un modelo flexible, robusto y dinámico para la solución ante un ataque basado en la inyección SQL.

1. Introducción

En los últimos años, se ha incrementado notablemente el número de ataques de que son objeto las aplicaciones Web [1]. *SecurityFocus* [2], ha identificado cinco de los ataques más comunes contra las aplicaciones Web, siendo la inyección SQL (*Structure Query Language injection*) uno de los más característicos.

Este tipo de ataque se manifiesta en los sistemas de base de datos introduciendo sentencias SQL [3], a través de entradas con niveles reducidos de validación y modificando la lógica de la consulta original.

En este artículo se presenta una arquitectura multiagente para la clasificación de consultas con inyección SQL. La arquitectura propuesta se enfoca no sólo en la clasificación de las consultas SQL, sino que considera monitorizar las acciones de los usuarios y la extracción de conocimiento a partir del historial de consultas ejecutadas sobre bases de datos. La arquitectura permitirá perfeccionar la técnica de clasificación ante nuevas estrategias de ataque y mejorar la seguridad de las aplicaciones Web.

Las características de los agentes y sistemas multiagente como la autonomía, robustez, adaptabilidad, aprendizaje, movilidad, etc. [4], hacen posible ofrecer una solución más flexible, robusta y dinámica que algunos de los principales enfoques propuestos [5] [6] [7] [8]. Un sistema multiagente está formado por un conjunto de múltiples entidades computacionales denominadas agentes, los cuales interactúan de forma autónoma y con cierto grado de inteligencia. Los agentes pueden funcionar en una diversidad de entornos y plataformas, intercambiando información con usuarios, otros agentes o su propio entorno. Pueden hacer uso

de mecanismos que les permitan resolver los problemas de mejor manera, uno de estos mecanismos es el razonamiento basados en casos (*CBR - Case-Based Reasoning*), el cual proporciona a los agentes una mayor capacidad para resolver problemas complejos basándose en la experiencia [9].

A continuación se presenta la problemática en torno a los ataques con inyección SQL. Posteriormente, se describe en detalle este tipo de ataque, continuando con un breve análisis sobre los agentes y sistemas multiagente. Después se presenta la arquitectura de un sistema multiagente para la detección y clasificación de inyecciones SQL, y se finaliza con los resultados y conclusiones obtenidos tras la implementación de un prototipo del sistema.

2. Descripción de la Problemática

Las bases de datos son el elemento clave de muchos sistemas de almacenamiento, búsqueda y manejo de información. Ellas son frecuentemente blanco de numerosos ataques debido al valor de la información que almacenan [10]. El éxito de un ataque puede ocasionar importantes daños que repercuten en el ámbito financiero, en la pérdida de credibilidad de los servicios Web y en la reputación de la organización que tiene a cargo su administración.

Uno de los principales ataques a las aplicaciones Web son las inyecciones SQL, que se caracterizan por el número de variantes que se pueden generar [11]. Desafortunadamente, los mecanismos tradicionales de seguridad, como los cortafuegos, los IDS (*Intrusion Detection System*) o las conexiones seguras SSL (*Secure Socket Layer*) no están especializados en este tipo de ataque [12] [13].

Actualmente existen diversas propuestas para la detección de ataques basados en inyección SQL. *JDBC Checker* [5] es una herramienta basada en el análisis estático que se limita sólo a la comprobación de excepciones relacionadas a los tipos de datos. *SQLRand* [6], es una plataforma que permite crear consultas mediante instrucciones SQL aleatorias, garantizando que sólo las consultas con sus respectivos valores aleatorios sean ejecutadas. Sin embargo, su seguridad depende de la forma en la que se maneja la clave aleatoria, además de imponer una sobrecarga causada por la dependencia de un servidor *proxy*, para decodificar la cadena a su estado normal para su ejecución. Recientemente, Valeur, *et al.* [7] han desarrollado un estudio de un IDS basado en anomalías, compuesto por un aprendizaje automático que se entrena a partir de accesos a la base de datos de forma correcta, para luego monitorizar y clasificar accesos sospechosos. Su debilidad radica en la dependencia de la calidad del entrenamiento, debido a que si la calidad es pobre, se produciría una alta tasa de falsos positivos y falsos negativos. *AMNESIA* [8] es una herramienta que combina el análisis estático y dinámico para construir modelos sobre los posibles valores para los puntos de entrada, los cuales son confrontados con la consulta SQL a evaluar. La consulta que viole el modelo se considera sospechosa y se rechaza. Sin embargo, su eficacia depende de la precisión para generar los modelos, por lo tanto, algunas estrategias de ataque pueden ocasionar un número considerable de ejemplos mal clasificados.

Estos son sólo algunos de los desarrollos directamente relacionados con la inyección SQL que han intentado solucionar parte de la problemática, sin embargo es necesario continuar desarrollando soluciones más robustas que permitan detectar y prevenir el ataque de inyección SQL.

Los sistemas multiagente han sido exitosamente implementados en áreas como la industria, las telecomunicaciones, el comercio electrónico, el entretenimiento, la salud, etc. [9] [14]. En el campo de la seguridad de las redes y los servicios Web, algunos de los trabajos propuestos basados en la utilización de sistemas multiagente, se enfocan en la detección de intrusos [15], y el filtrado de correo no deseado [16]. En la siguiente sección se describe con detalle la inyección SQL, sus mecanismos de ataque y la clasificación de los tipos de inyecciones más conocidos.

3. Ataque de Inyección SQL

El lenguaje estructurado de consulta, es un tipo de lenguaje textual utilizado para la interacción con bases de datos relacionales [3]. El lenguaje incorpora sentencias DDL (*Data Definition Language Statements*), DCL (*Data Control Language Statement*) y sentencias DML (*Data Manipulation Language Statements*). La unidad de ejecución principal consiste en una consulta (*Query*) que devuelve un conjunto de resultados.

La inyección SQL se produce cuando un atacante logra cambiar la lógica semántica o sintáctica de un enunciado SQL legítimo, mediante la inserción de palabras reservadas del propio lenguaje u operadores especiales dentro de la consulta SQL original [3] [8], que se ejecutará sobre la base de datos. En la aplicación Web se concatena la cadena estática con las variables de entrada del usuario. En el caso de que no se aplique un mecanismo de validación apropiado, estas entradas se convierten en puntos vulnerables para la inyección SQL. El resultado de este ataque conlleva el acceso y la manipulación de los datos de forma no autorizada, a la recuperación de información confidencial y en el peor de los casos, compromete el servidor donde se aloja la aplicación Web [11]. Los sistemas administradores de base de datos como *Microsoft SQL Server*, *Oracle*, *MySQL*, *Informix*, *Sybase*, son blanco de inyecciones SQL [17].

Los mecanismos de ataques se fundamentan en inyección a través de: entradas de usuarios, *cookies*, las variables del servidor, o un tipo de inyección de segundo orden [11]. Una clasificación de los tipos de inyección SQL [11] se muestra en la Tabla 1.

Tabla 1. Clasificación de los tipos de inyección SQL

Tipo de inyección	Descripción	Ejemplo
Tautología	La condición siempre resultará verdadera. El formulario de autenticación en una aplicación Web.	`SELECT * from TblUser where Username = '' or 1=1 -- and pass= ''`

Errores Lógicos-Consultas Ilegales	Aplicación de ingeniería inversa sobre los mensajes de error para obtener más información del esquema de la base de datos.	`SELECT * FROM TblSupplier WHERE NameSupplier = 'O'Reill'`
Unión de Consulta	A través de una unión, es posible que el atacante obtenga el control sobre las consultas logrando obtener los resultados de alguna de ellas.	`SELECT * FROM TblSupplier WHERE NameSupplier = '' UNION ALL SELECT * From TblConsumer WHERE 1 = 1`
Consulta piggy-backed	Se inyecta una nueva consulta sobre la original sin cambiar la lógica de la primera.	`SELECT * FROM TblUser WHERE UserName = ''; DROP TABLE TblUser -- 'AND pass =''`
Inyección basada en la inferencia	No hay mensajes de Error. El atacante envía consultas para inferir sobre las respuestas (cierto o falso). Otra variante, es realizar la inferencia en función del tiempo de respuesta.	`DECLARE @s varchar(8000); SELECT @s = db_name(); if (ascii(substring(@s, 1, 1)) {&} ( power(2, 0))) >0 waitfor delay '0:0:5'`
Procedimientos almacenados	Permiten una extensión de las funcionalidades de las bases de datos, inclusive, interactuar con el sistema operativo del servidor.	`Simplequoted.asp? city=seattle';EXEC master.dbo.xp_cmdshell 'cmd.exe dir c:`
Codificación alternativa	Permite enmascarar la inyección utilizando un tipo de codificación: codificación ASCII, formato hexadecimal.	`DECLARE @q varchar(8000); SELECT @q = 0x73656c65637420 404076657273696f6e; EXEC(@q). Resultado:'SELECT @@version'`

4. Tecnología de Agentes y Sistemas Multiagente

Un sistema multiagente es básicamente un conjunto de agentes autónomos capaces de trabajar de forma conjunta para resolver un problema [4]. Estos sistemas deben cumplir una serie de condiciones para que puedan ser catalogados como tal. Como mínimo, uno de los agentes debe ser autónomo y debe existir por lo menos una relación entre dos agentes, en donde uno de ellos satisfaga alguno de los objetivos del otro. Además, cualquiera de los agentes debe poseer una capacidad limitada para resolver el problema de forma independiente [18].

La forma como se descomponen los agentes en un conjunto de módulos y cómo estos módulos interactúan entre sí para alcanzar la funcionalidad requerida, viene dada por la arquitectura de agente seleccionada [4]. El modelo presentado en este artículo hace uso de la arquitectura deliberativa, la cual maneja una representación simbólica del mundo real, en donde las decisiones de un agente están fundamentadas en el razonamiento lógico basado en la concordancia de patrones y en la manipulación simbólica, partiendo de un estado inicial y un conjunto de planes con un objetivo a cumplir. Dentro de las arquitecturas deliberativas se encuentra la BDI (*Believe, Desire, Intention*), basada en aptitudes mentales, utilizando creencias, deseos e intenciones [19] [20].

Para definir y modelar adecuadamente un sistema multiagente es necesario emplear herramientas de ingeniería del software orientadas a agentes (*AOSE*), lo que conlleva a una serie de fases de análisis y diseño del sistema. Uno de los principales problemas con los que se encuentran los analistas de sistemas multiagente en la actualidad, consiste en que aun se carece de un estándar claro o una metodología completa que permita definir los pasos que involucra cada una de las etapas del modelado de un sistema multiagente [14]. Una posible solución a este problema, es la combinación de la metodología Gaia (*Methodology for Agent-Oriented Analysis and Design*) [21] y el lenguaje AUML (*Agent Unified Modeling Language*) [22], la cual ha sido aplicada en el análisis y diseño de diversos sistemas multiagente [9]. Gaia permite una visión abstracta del sistema en términos de organización, pero no proporciona detalles de implementación. Se basa fundamentalmente en la obtención de modelos de análisis, como son el modelo de roles y el modelo de interacción; en la obtención de modelos de diseño, como son el modelo de agentes, el modelo de conocidos y el modelo de servicios; y en la posibilidad de refinar de manera progresiva en cada una de las iteraciones que se den en la fase de análisis y diseño. Por su parte, AUML ofrece un nivel más detallado del sistema para su posterior implementación, pero sus modelos requieren un nivel de detalle alto para comenzar el estudio de la mayor parte del sistema multiagente. Los diagramas generados en AUML son: especificación de roles, diagramas de clases, diagramas de protocolos, diagramas de colaboración, diagramas de secuencia, diagramas de actividad y diagramas de estados. La combinación de ambas metodologías requiere establecer una etapa intermedia que permita utilizar los resultados obtenidos con Gaia en el inicio del diseño AUML. Es necesario adaptar ciertos criterios que varían ligeramente su significado en Gaia y AUML tales como rol, servicio y capacidad.

5. Arquitectura propuesta

En este trabajo se presenta una arquitectura multiagente orientada a la detección de ataques basados en inyección SQL. Esta arquitectura permite aplicar las propiedades y características de los sistemas multiagente, para aportar nuevas soluciones a un problema bien conocido. El sistema que se desarrolla sobre esta arquitectura clasifica las consultas SQL con una baja tasa de falsos positivos y falsos negativos; monitoriza el comportamiento de los usuarios considerados de confianza, pero que puedan representar una amenaza para la aplicación; y por último, extrae conocimiento del registro de las consultas. Todo ello de forma dinámica y distribuida, con la finalidad de proporcionar información que permita perfeccionar las técnicas de clasificación de consultas SQL y mejorar la seguridad de la aplicación Web.

Este sistema, además de incorporar un mecanismo de clasificación, plantea un proceso de realimentación dinámico incorporando técnicas de razonamiento basado en casos (*CBR*) [9]. Cada consulta SQL es un caso a evaluar a partir del cual es posible aprender, no solamente desde el punto de vista de su fiabilidad, sino que aporta información que luego es reutilizada por el propio sistema para

perfeccionar la técnica de clasificación. También es posible extraer conocimiento para identificar puntos vulnerables en la aplicación Web, objetos de la base de datos susceptibles a intentos de ataque, sentencias del lenguaje más usuales en los ataques, la generación de perfiles de usuarios clasificados como peligrosos, etc.

El sistema está compuesto por un conjunto de agentes, con roles bien definidos que comparten información y servicios. La clasificación de cada consulta SQL implica un proceso complejo donde cada agente ejecuta su tarea con la información disponible en cada momento. En la Figura 1, se muestra un esquema de la arquitectura del sistema multiagente para la clasificación de las consultas SQL. El usuario envía la consulta a través de la página Web de la aplicación; antes de que la consulta alcance el punto de ejecución sobre la base de datos, el sistema multiagente captura la consulta y la clasifica dependiendo de su fiabilidad. Si la consulta es fiable, se ejecuta y devuelve los resultados al usuario. En caso contrario, la consulta se rechaza.

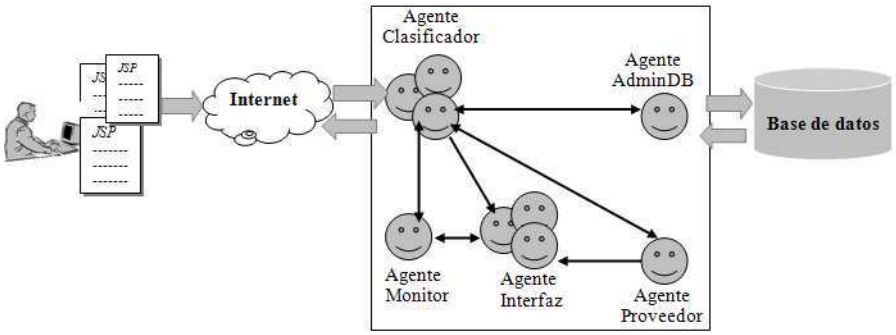

Fig. 1. Esquema de la arquitectura del sistema multiagente

El sistema multiagente trabaja de forma distribuida y utiliza agentes deliberativos BDI para resolver diversos problemas relacionados con la inyección SQL. Uno de los principales problemas a resolver es identificar el número de variantes en los ataques que se pueden producir como resultado de la combinación de diferentes estrategias. Resulta complejo determinar con precisión la fiabilidad de una consulta SQL sólo desde el punto de vista sintáctico, pero es posible aplicar un proceso de razonamiento que evalúe diferentes factores, como el perfil del usuario que realiza la consulta, los resultados de comparación de modelos predefinidos con la nueva consulta, evaluación de resultados obtenidos de la base de datos en consultas anteriores y soluciones anteriores de clasificación para consultas similares. La utilización de agentes deliberativos BDI y mecanismos de razonamiento basados en casos son una solución adecuada para afrontar este tipo de problemas, caracterizados por una escasa información y un alto grado de incertidumbre [23] [9].

El funcionamiento del sistema multiagente conlleva la coordinación de agentes que cooperan y comparten entre sí la información necesaria durante el proceso de clasificación. Partiendo de los requisitos del problema, se ha utilizado la metodología Gaia para la identificación de los roles y sus distintos puntos de interacción. Al descomponer el proceso de clasificación, las tareas generadas se distribuyen entre cada uno de los roles del sistema que son los siguientes:

- Clasificador: captura y clasifica las consultas SQL. Incluye la generación de modelos para cada consulta, realiza un análisis sintáctico, compara modelos (modelos estáticos, esquema de la base de datos y memoria de casos).
- Proveedor: extrae casos de la memoria, patrones de ataques conocidos y el esquema de la base de datos.
- AdminDB: ejecuta las consultas sobre la base de datos corporativa y aplica un análisis post-ejecución de los resultados comparándolos con resultados anteriores de consultas similares.
- Monitor: procesa y monitoriza la bitácora de las acciones de los usuarios. Aplica procesos de extracción de conocimiento sobre la bitácora de las consultas realizadas.
- Interfaz: punto de interacción con el administrador Web, activando alertas, procesando y presentando informes según la necesidad del administrador.

La Figura 2 muestra el modelo de agentes para este sistema. Cada agente tiene asignado un rol específico. Será necesario que exista al menos una instancia tanto del rol clasificador como del rol interfaz.

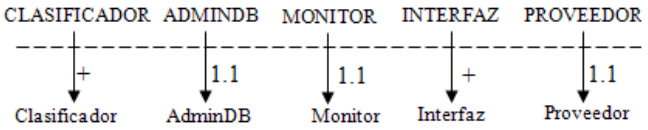

Fig. 2. Modelo de Agentes

Otro modelo obtenido aplicando la metodología Gaia, es el modelo de conocidos, que se muestra en la Figura 3(a). Este modelo representa la comunicación entre los agentes y permite visualizar el flujo de comunicación entre ellos, además de detectar posibles puntos de congestión antes de la implementación.Como resultado de la interacción entre los roles, se definen los protocolos necesarios, así como los roles que toman parte en dicha interacción. Una vez generados los modelos con Gaia, es necesario adaptarlos y detallarlos con AUML para obtener un nivel de detalle más elevado y por lo tanto, un modelo más cercano a la implementación. En la Figura 3(b) se representa la interacción entre el agente clasificador y el agente proveedor, utilizando el diagrama de protocolos de AUML. El rol clasificador solicita al rol proveedor la librería de patrones SQL conocidos, disponibles en ese momento. El rol proveedor recibe la solicitud y envía al clasificador la respuesta de la solicitud.

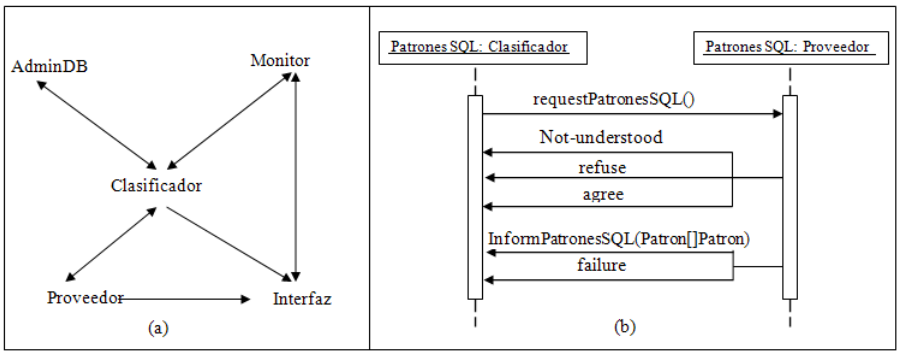

Fig. 3. (a)Modelo de conocidos del sistema multiagente (b)Diagrama de interacción entre el agente clasificador y el agente proveedor

6. Resultados y Conclusiones

En este artículo se ha presentado la arquitectura de un sistema multiagente para la clasificación de las consultas SQL, con el fin de prevenir inyecciones SQL en la base de datos de las aplicaciones Web. A través de la metodología Gaia y el lenguaje AUML se diseñó un modelo de la arquitectura del sistema y con base a éste, se desarrolló un prototipo funcional.

La arquitectura de un sistema multiagente para la clasificación de consultas SQL aprovecha las características particulares de los agentes para proveer un mecanismo robusto, flexible y dinámico con el fin de obtener clasificaciones de consultas SQL eficientes. Para ello se incorporan nuevos mecanismos tales como, razonamiento basado en casos, la monitorización de las acciones de usuarios y la implementación de técnicas de minería de datos sobre las consultas globales, que permitan la realimentación del propio sistema. Con esto, se pretende mejorar la técnica de clasificación así como la seguridad del sistema reduciendo al máximo posible los falsos positivos.

Los resultados obtenidos tras la implementación del prototipo del sistema, demuestran que el sistema multiagente es eficiente para la clasificación de consultas SQL. Como se aprecia en la Figura 5, en la primera iteración se clasificaron 27 consultas con un error del 54.05 %; sin embargo, a medida que se incrementó el número de iteraciones y consultas, el sistema multiagente alcanzó un cierto nivel de aprendizaje, mejorando notablemente sus resultados. En la iteración 11, con 100 consultas se obtuvo sólo un margen del 2.12 % de error en la clasificación.

Como trabajo futuro, se continuará con el desarrollo hasta obtener un prototipo más completo que integre los procesos de monitorización y la extracción de información aplicando técnicas de minería de datos, además de someter el sistema a pruebas en entornos reales, que permitan evaluar y demostrar la eficiencia de la solución propuesta para el ataque basado en la inyección SQL.

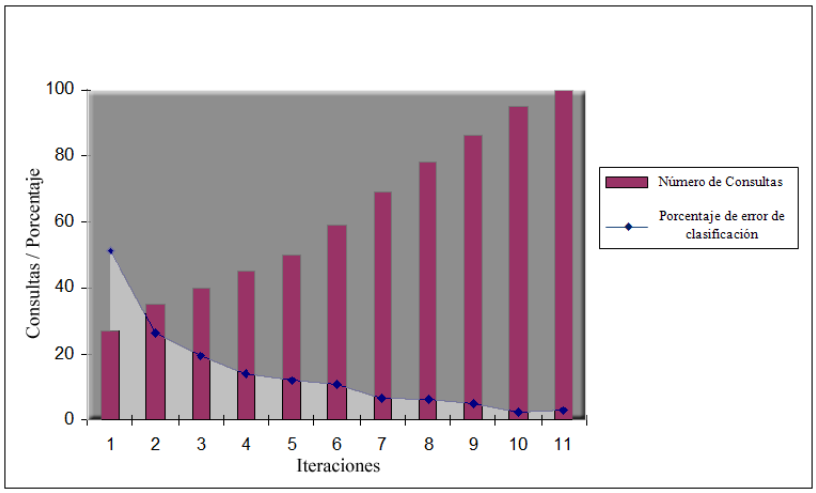

Fig. 4. Resultados del clasificador durante su fase de prueba

Referencias

1. CSO Magazine, United Service Secret Service, CERTő: 2005 e-crime watch survey. http://www.cert.org/archive/pdf/ecrimesummary05.pdf (2005) Consultado el 03/10/2007.
2. Siddharth, S., GCIA, Doshi, P.: Five common web application vulnerabilities. SecurityFocus. http://www.securityfocus.com/infocus/1864 (2006) Consultado el 03/10/2007.
3. Anley, C.: Advanced sql injection in sql server applications. http://www.nextgenss.com/papers/advanced_sql_injection.pdf (2002) Consultado el 03/10/2007.
4. MAS, A.: Agentes Software y Sistemas Multiagente: conceptos, arquitectura y aplicaciones. Pearson-Prentice Hall (2005)
5. Gould, C., Su, Z., Devanbu, P.: JDBC Checker: A static analysis tool for sql/jdbc applications. In: ICSE '04: Proceedings of the 26th International Conference on Software Engineering, Washington, DC, USA, IEEE Computer Society (2004) 697–698
6. Boyd, S.W., Keromytis, A.D.: SQLrand: Preventing sql injection attacks. In: Applied Cryptography and Network Security. Volume 3089. (2004) 292–302
7. Valeur, F., Mutz, D., Vigna, G.: A learning-based approach to the detection of sql attacks. In: Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA), Vienna, Austria (2005) 123–140
8. Halfond, W.G.J., Orso, A.: AMNESIA: analysis and monitoring for neutralizing sql-injection attacks. In: ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, New York, NY, USA, ACM (2005) 174–183
9. Corchado, J.M., Bajo, J., de Paz, Y., Tapia, D.: Intelligent environment for monitoring alzheimer patients, agent technology for health care. Decision Support Systems **34**(2) (2008) 382–396

10. SANS Institute: Sans top-20 internet security attack targets - 2006 annual update. https://www2.sans.org/top20/ (2006) Consultado el 03/11/2007.
11. Halfond, W.G., Viegas, J., Orso, A.: A classification of sql-injection attacks and countermeasures. In: Proceedings of the IEEE International Symposium on Secure Software Engineering, Arlington, VA, USA (2006)
12. Acunetix Software Ltd.: Is your website hackable? http://www.acunetix.com/vulnerability-scanner/wvsbrochure.pdf (2007) Consultado el 05/10/2007.
13. WhiteHat Security: Web application security 101 real-world examples, tools and techniques for securing websites. http://www.whitehatsec.com/articles/webappsec101.pdf (2005) Consultado el 05/10/2007.
14. Bajo, J., de Luis, A., Gonzalez, A., Saavedra, A., Corchado, J.M.: A shopping mall multiagent system: Ambient intelligence in practice. In Bravo, J., ed.: 2nd International Workshop on Ubiquitous Computing & Ambient Intelligence. (2006) 115–125
15. Lazzari, L., Mari, M., Poggi, A.: A collaborative and multi-agent approach to e-mail filtering. In: IAT '05: Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology, Washington, DC, USA, IEEE Computer Society (2005) 238–241
16. Herrero, A., Corchado, E., Pellicer, M.A., Abraham, A.: Hybrid multi agent-neural network intrusion detection with mobile visualization. In Heidelberg, S.V.B., ed.: Innovations in Hybrid Intelligent Systems. Volume 44 of Advances in Soft Computing. (2007) 320–328
17. Litchfield, D.: Data mining with sql injection and inference. NGS Software. http://www.ngssoftware.com/research/papers/sqlinference.pdf (2005) Consultado el 15/10/2007.
18. Woolridge, M., Wooldridge, M.J.: Introduction to Multiagent Systems. John Wiley & Sons, Inc., New York, NY, USA (2002)
19. Bratman, M.E.: Intention, Plans, and Practical Reason. Harvard University Press, Cambridge, MA (1987)
20. Rao, A.S., Georgeff, M.P.: Modeling rational agents within a BDI-architecture. In Allen, J., Fikes, R., Sandewall, E., eds.: Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91), Morgan Kaufmann publishers Inc.: San Mateo, CA, USA (1991) 473–484
21. Wooldridge, M., Jennings, N.R., Kinny, D.: The Gaia methodology for agent-oriented analysis and design. Autonomous Agents and Multi-Agent Systems **3**(3) (2000) 285–312
22. Bauer, B., Müller, J.P., Odell, J.: Agent UML: a formalism for specifying multi-agent software systems. In: First international workshop, AOSE 2000 on Agent-oriented software engineering, Secaucus, NJ, USA, Springer-Verlag New York, Inc. (2001) 91–103
23. Corchado, J.M., Laza, R.: Constructing deliberative agents with case-based reasoning technology. International Journal of Intelligent Systems **18**(12) (2003) 1227–1241

View publication stats