ReCIBE. Revista electrónica de Computación, Informática Biomédica y Electrónica
E-ISSN: 2007-5448
recibe@cucei.udg.mx
Universidad de Guadalajara
México

Brito Abundis, Carlos Joaquín
Metodologías para desarrollar software seguro
ReCIBE. Revista electrónica de Computación, Informática Biomédica y Electrónica, núm. 3, diciembre, 2013
Universidad de Guadalajara
Guadalajara, México

ReCIBE, Año 2 No.3, Diciembre 2013

Metodologías para desarrollar software seguro

Carlos Joaquín Brito Abundis
Universidad Autónoma de Zacatecas
carlosbreeto@gmail.com

Resumen: La seguridad ha pasado de ser un requerimiento no funcional, que podía implementarse como parte de la calidad del software a un elemento primordial de cualquier aplicación. Los hackers y grupos criminales evolucionan día a día y se han convertido expertos en explotar las vulnerabilidades de las aplicaciones y sitios en internet. Para hacer frente a estas amenazas, es necesaria la implementación de metodologías que contemplen en su proceso de desarrollo de software la eliminación de vulnerabilidades y la inclusión de la seguridad como un elemento básico en la arquitectura de cualquier producto de software. Este trabajo revisa algunas de las metodologías que contemplan la seguridad en su proceso.

Palabras clave: metodologías, desarrollo de software, procesos ágiles, seguridad, vulnerabilidades.

Methodologies for software security development

Abstract: Security has changed from a non-functional requirement, which could be implemented as a part of software quality, to a key element in any software application. Hackers and criminal groups evolve every day and they have become expert in exploiting vulnerabilities in applications and websites. To address these threats, it is necessary that organizations implementing methodologies that include activities focused on eliminating vulnerabilities and

integrating security as a basic element in the software development process. This paper reviews some of the methodologies that provide security activities in the software development process.

Keywords: methodologies, software development, agile processes, security, vulnerabilities.

1. Introducción

La sociedad se encuentra vinculada innegablemente a la tecnología; sin embargo, su mismo uso tan amplío hace que existan grandes riesgos en cada una de las aplicaciones de la tecnología. Además, los riesgos evolucionan y aumentan con la tendencia en el uso de tecnologías como el cloud computing, Web 2.0 y aplicaciones para dispositivos móviles (Laskowski, 2011). En una organización se pierden aproximadamente $6.6 millones de dólares por cada brecha de seguridad de TI, estas actividades ilícitas tienen ganancias de hasta 114 billones de dólares anualmente y 431 millones de víctimas anuales, es decir, 14 víctimas de cibercrimen cada segundo (Norton, 2012). Un escenario mundial que no puede pasar desapercibido por la gente relacionada al desarrollo de software y empezar a optar por metodologías que garanticen el lugar que le corresponde a la seguridad.

El objetivo de este trabajo es presentar dos de las metodologías de desarrollo que permiten producir productos de software seguros. Después se presenta un análisis de las metodologías que permiten producir productos de software seguros, una comparación entre ellas y por último las conclusiones y los trabajos futuros que se desprenden de éste trabajo.

2. Metodologías de desarrollo tradicionales

La clave de un software seguro, es el proceso de desarrollo utilizado. En el proceso, es donde se produce el producto que pueda resistir o sostenerse ataques ya anticipados, y recuperarse rápidamente y mitigar el daño causado por los ataques que no pueden ser eliminados o resistidos. Muchos de los defectos relacionados con la seguridad en software se pueden evitar si los desarrolladores estuvieran mejor equipados para reconocer las implicaciones de su diseño y de las posibilidades de implementación.

La manera en que se desarrolla software ha evolucionado, desde la forma de “code and fix” (codificar y después arreglar), en la que un equipo de desarrollo tiene la idea general de lo que quiere desarrollar, pasando a metodologías formales (RUP, Proceso Unificado, PSP/TSP, entre otras) en las que existe una planeación detallada del desarrollo, desde la elicitación, documentación, requerimientos, diseño de alto nivel y la inspección (Vanfosson, 2006). Los métodos ágiles permiten tener un desarrollo iterativo, con ciclos de entrega continuos, un contacto con el cliente permanente; permitiendo que se incluyan como parte de los stakeholders a los administradores de riesgo de la empresa, certificadores y al personal responsable de las políticas de seguridad. Los métodos ágiles se rigen bajo el manifiesto de los procesos ágiles, que menciona que la prioridad más alta es la satisfacción del cliente a través de entregas tempranas y continúas de un producto con valor (Fowler & Highsmith, 2001).

Sin embargo; las metodologías mencionadas, tienden enfocarse en mejorar la calidad en el software, reducir el número de defectos y cumplir con la funcionalidad especificada (Davis, 2005); pero en la actualidad, también es necesario entregar un producto que garantice tener cierto nivel de seguridad. Hay metodologías que contemplan durante su proceso, un conjunto de actividades específicas para remover vulnerabilidades detectadas en el diseño

o en el código, la aplicación de pruebas que aportan datos para la evaluación del estado de seguridad, entre otras actividades relacionadas para mejorar la seguridad del software.

3. Metodologías enfocadas al desarrollo de software seguro

Existen varias metodologías que establecen una serie de pasos en búsqueda de un software más seguro y capaz de resistir ataques. Entre ellas se encuentran Correctness by Construction (CbyC), Security Development Lifecycle (SDL), Cigital Touchpoints, Common Criteria, Comprehensive, Lightweight Application Security Process (CLASP), TSP-Secure. El presente artículo estudiará las características de las dos primeras, detallando las fases que las conforman, destacando sus particularidades y al final se realiza una comparativa de ambas.

3.1 Correctness by Construction (CbyC)

Es un método efectivo para desarrollar software que demanda un nivel de seguridad crítico y que además sea demostrable. La empresa Praxis ha utilizado CbyC desde el año 2001 y ha producido software industrial con taza de defectos por debajo de los 0.05 defectos por cada 1000 líneas de código, y con una productividad de 30 líneas de código por persona al día.

Las metas principales de ésta metodología son obtener una taza de defectos al mínimo y un alta resilencia al cambio; los cuales se logran debido a dos principios fundamentales: que sea muy difícil introducir errores y asegurarse que los errores sean removidos tan pronto hayan sido inyectados. CbyC busca producir un producto que desde el inicio sea correcto, con requerimientos

rigurosos de seguridad, con definición muy detallada del comportamiento del sistema y un diseño sólido y verificable (Croxford & Chapman, 2005).

3.1.1 Fases de la Metodología CbyC

CbyC combina los métodos formales con el desarrollo ágil; utiliza notaciones precisas y un desarrollo incremental que permite mostrar avances para recibir retroalimentación y valoración del producto. La Figura 1, muestra las fases propuestas por ésta metodología para el desarrollo de software.

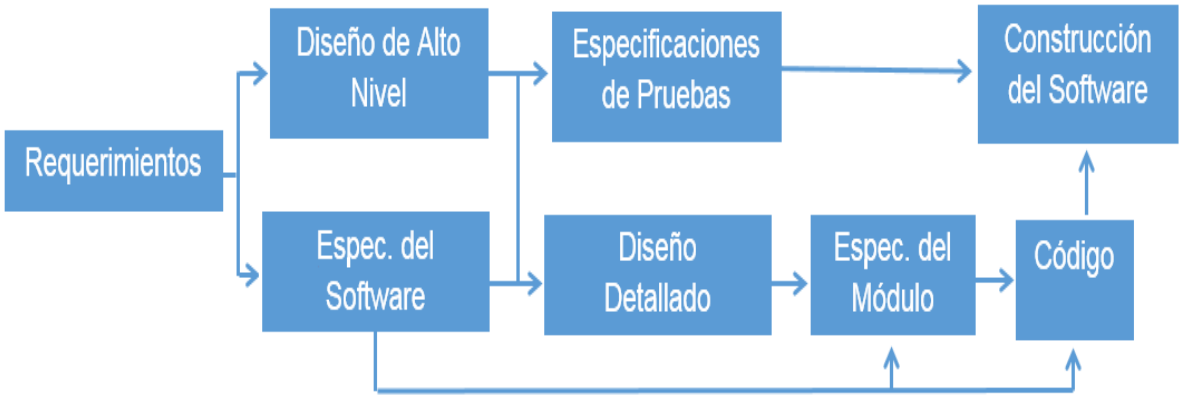

Figura 1.*Proceso de desarrollo propuesto por el CbyC (Amey, 2006).*

3.1.1.1 Fase de Requerimientos

En la fase de requerimientos se especifica el propósito, las funciones y los requerimientos no funcionales. Se escriben los requerimientos de usuario con sus respectivos diagramas contextuales, diagramas de clase y definiciones operativas. Cada requerimiento debe pasar por un proceso de trazabilidad, así como requerimiento de seguridad a la amenaza correspondiente.

3.1.1.2 Fase de Diseño de Alto Nivel

Se describe la estructura interna del sistema, (distribución de la funcionalidad, estructura de las bases de datos, mecanismos para las transacciones y comunicaciones) la manera en que los componentes colaboran y especifican con mayor énfasis los requerimientos no funcionales de protección y seguridad. Se utilizan los métodos formales para definir el diseño de alto nivel y obtener una descripción, un comportamiento y un modelo preciso. Los métodos formales han probado ser exitosos para especificar y probar el nivel de 'correctness' y en la consistencia interna de las especificaciones de funciones de seguridad (Jarzombek & Goertzel, 2006).

3.1.1.3 Fase de Especificación del Software

En ésta fase se documentan las especificaciones de la interfaz de usuario (se define el "look and feel" del sistema), las especificaciones formales de los niveles superiores y se desarrolla un prototipo para su validación.

3.1.1.4 Fase de Diseño Detallado

Define el conjunto de módulos y procesos y la funcionalidad respectiva. Se utiliza notación formal para eliminar ambigüedad en la interpretación del diseño.

3.1.1.5 Fase de Especificación de los Módulos

Se define el estado y el comportamiento encapsulado en cada módulo del software tomando en cuenta el flujo de información. Los módulos deberán de

tener el enfoque de bajo acoplamiento y alta cohesión; así los efectos serían mínimos en caso de producirse una falla en el flujo de información.

3.1.1.6 Fase Codificación

El concepto de evitar cualquier ambigüedad posible, también se aplica en el código; por ello se sugiere la utilización de SPARK (lenguaje formalmente definido y matemáticamente comprobable, basado en el lenguaje de programación Ada, utilizado en sistema de aeronáutica, sistemas médicos y control de procesos en plantas nucleares) en las partes críticas del sistema por estar diseñado para tener una semántica libre de ambigüedades, permitir realizar análisis estático y ser sujeto de una verificación formal (Brito, 2010). Se conducen pruebas de análisis estático al código para eliminar algunos tipos de errores y en caso de ser necesario, se realizará una revisión al código.

3.1.1.7 Fase de las Especificaciones de las Pruebas

Para obtener las especificaciones de las pruebas, se toman en cuenta las especificaciones del software, los requerimientos y el diseño de alto nivel. Se efectúan prueban de valores límites, pruebas de comportamiento y pruebas para los requerimientos no funcionales; pero CbyC no usa pruebas de caja blanca ni pruebas de unidad; todas las pruebas son a nivel de sistema y orientado a las especificaciones.

3.1.1.8 Fase de Construcción del Software

CbyC utiliza el desarrollo de tipo ágil; en la primera entrega, se tiene el esqueleto completo del sistema con todas las interfaces y mecanismos de

comunicación; con una funcionalidad muy limitada que se irá incrementado en cada iteración del ciclo.

CbyC es compatible con los principios de los procesos de Personal Software Process/Team Software Process (PSP/TSP) y al combinar resulta en una taza reducción en la taza de defectos. El enfoque técnico de CbyC complementa a PSP para aumentar la capacidad de remoción de defectos en etapas tempranas del desarrollo. (Croxford & Chapman, 2005) . La Figura 2, muestra las tasas de defectos en los cinco niveles del CMM y la tasa cercana cero perteneciente a la metodología CbyC, dejando en claro el beneficio de la implementación de ésta un proceso de desarrollo.

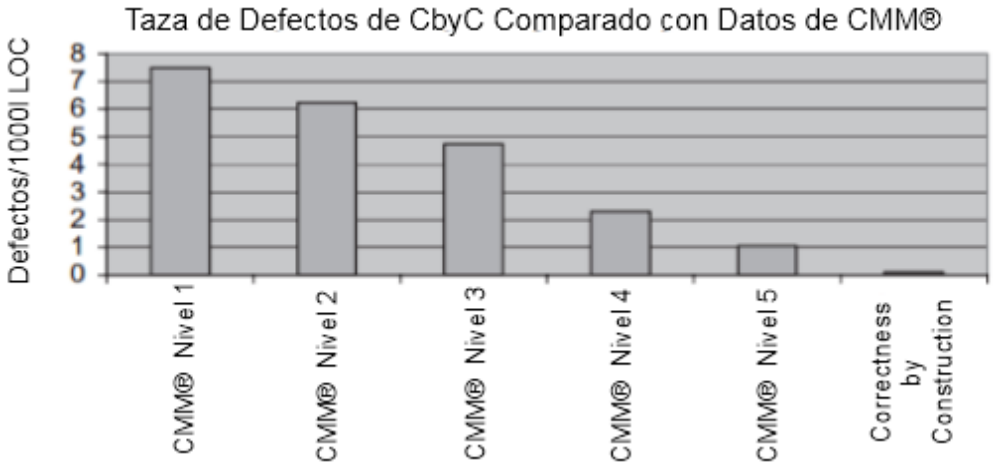

Figura 2.*Comparativa de taza de defectos CMMI contra CbyC (Hall, 2007).*

3.2 Security Development Lifecycle (SDL)

Es un proceso para mejorar la seguridad de software propuesto por la compañía de Microsoft en el año 2004; con dieciséis actividades enfocadas a mejorar la seguridad del desarrollo de un producto de software (Peterson, 2011).

Las prácticas que propone SDL van desde una etapa de entrenamiento sobre temas de seguridad, pasando por análisis estático, análisis dinámico, fuzz testing del código hasta tener plan de respuesta a incidentes. Una de las características principales de SDL es el modelado de amenazas que sirve a los desarrolladores para encontrar partes del código, donde probablemente exista vulnerabilidades o sean objeto de ataques (Korkeala, 2011).

3.2.1 Fases de la Metodología SDL

Existen dos versiones del SDL, la versión rígida y la orientada al desarrollo ágil. Las diferencias versan en que la segunda desarrolla el producto de manera incremental y en la frecuencia de la ejecución de las actividades para el aseguramiento de la seguridad. La versión rígida del SDL es más apropiada para equipos de desarrollo y proyectos más grandes y no sean susceptibles a cambios durante el proceso. SDL ágil es recomendable para desarrollos de aplicaciones web o basados en la web (Wood & Knox, 2012). En la Figura 3, se encuentra el flujo de las fases en la metodología de SDL cabe resaltar la existencia de una etapa previa a los requerimientos, enfocada al entrenamiento en seguridad y la última etapa del proceso que se encarga de darle seguimiento al producto en caso de algún incidente de seguridad.

Figura 3. *Proceso del desarrollo software del SDL (Microsoft Corporation, 2007).*

Las dieciséis actividades de SDL, que consolidan la seguridad, se detallan en la Tabla 1.

Actividades del SDL para la Seguridad	
1. Entrenamiento	**5. Verificación**
Entrenamiento de seguridad básica	Análisis dinámico
2. Requerimientos	Fuzz Testing
Establecer requerimientos de seguridad	Revisión de la superficie de ataques
Crear umbrales de calidad y límites de errores	**6. Lanzamiento**
Evaluación de los riesgos de seguridad y privacidad	Plan de respuesta a incidentes
3. Diseño	Revisión de seguridad final
Establecer requerimientos de diseño	Aprobar y archivar lanzamiento
Análisis de la superficie de ataques	**7. Respuesta**
Modelado de amenazas	Ejecutar el plan de respuesta a incidentes
4. Implementación	
Utilizar herramientas aprobadas	
Prohibir funciones no seguras	
Análisis estático	

Tabla 1. *Actividades del SDL para el aseguramiento de la seguridad (Microsoft Corporation, 2007)*

3.2.1.1 Fase de Entrenamiento

Todos los miembros de un equipo de desarrollo de software deben recibir una formación apropiada con el fin de mantenerse al corriente de los conceptos básicos y últimas tendencias en el ámbito de la seguridad y privacidad. Las personas con roles técnicos (desarrolladores, evaluadores y administradores de programas) que están directamente implicadas en el desarrollo de programas de software deben asistir como mínimo una vez al año a una clase de formación en materia de seguridad en conceptos fundamentales como: defensa en profundidad, principio de privilegios mínimos, modelos de riesgos,

saturaciones de búfer, inyección de código SQL, criptografía débil, evaluación de riesgos y procedimientos de desarrollo de privacidad.

3.2.1.2 Fase de Requerimientos

En la fase de requerimientos, el equipo de producción es asistido por un consultor de seguridad para revisar los planes, hacer recomendaciones para cumplir con las metas de seguridad de acuerdo al tamaño del proyecto, complejidad y riesgo. El equipo de producción identifica como será integrada la seguridad en el proceso de desarrollo, identifica los objetivos clave de seguridad, la manera que se integrará el software en conjunto y verificarán que ningún requerimiento pase desapercibido.

3.2.1.3 Fase de Diseño

Se identifican los requerimientos y la estructura del software. Se define una arquitectura segura y guías de diseño que identifiquen los componentes críticos para la seguridad aplicando el enfoque de privilegios mínimos y la reducción del área de ataques. Durante el diseño, el equipo de producción conduce un modelado de amenazas a un nivel de componente por componente, detectando los activos que son administrados por el software y las interfaces por las cuales se pueden acceder a esos activos. El proceso del modelado identifica las amenazas que potencialmente podrían causar daño a algún activo y establece la probabilidad de ocurrencia y se fijan medidas para mitigar el riesgo.

3.2.1.4 Fase de Implementación

Durante la fase de implementación, se codifica, prueba e integra el software. Los resultados del modelado de amenazas sirven de guía a los desarrolladores para generar el código que mitigue las amenazas de alta prioridad. La codificación es regida por estándares, para evitar la inyección de fallas que conlleven a vulnerabilidades de seguridad. Las pruebas se centran en detectar vulnerabilidades y se aplican técnicas de fuzz testing y el análisis de código estático por medio de herramientas de escaneo, desarrolladas por Microsoft. Antes de pasar a la siguiente fase, se hace una revisión del código en búsqueda de posibles vulnerabilidades no detectadas por las herramientas de escaneo.

3.2.1.4 Fase Verificación

En ésta parte el software, ya es funcional en su totalidad y se encuentra en fase beta de prueba. La seguridad es sujeta a un “empujón de seguridad”; que se refiere a una revisión más exhaustiva del código y a ejecutar pruebas en parte del software que ha sido identificado como parte de la superficie de ataque.

3.2.1.5 Fase de Lanzamiento

En el lanzamiento, el software es sujeto a una revisión de seguridad final, durante un periodo de dos a seis meses previos a la entrega con el objetivo de conocer el nivel de seguridad del producto y la probabilidad de soportar ataques, ya estando liberado al cliente. En caso de encontrar vulnerabilidades que pongan en riesgo la seguridad, se regresa a la fase previa para enmendar esas fallas.

3.2.1.6 Fase de Respuesta

Sin importar la cantidad de revisiones que se haga al código o las pruebas en búsqueda de vulnerabilidades, no es posible entregar un software cien por ciento seguro; así que, se debe de estar preparado para responder a incidentes de seguridad y a aprender de los errores cometidos para evitarlos en proyectos futuros (Lipner, 2004).

La compañía Microsoft utilizó el proceso de SDL para hacer el Windows Vista, reduciendo en un 50% las vulnerabilidades comparadas con Windows XP SP2. Después lanzó Microsoft Office 2007, con una reducción del 90% de las vulnerabilidades, a causa de dos requerimientos de SDL: del requerimiento de integer overflow y el requerimiento de fuzzing (Microsoft Corporation, 2007).

4. Comparativa de las metodologías CbyC y SDL

Las metodologías presentadas en éste artículo, tienen como objetivo establecer una forma de desarrollar software que sea más seguro y capaz de soportar ataques maliciosos. En la Tabla 2 se hace una comparativa entre las características de ambas.

Atributo/Metodología	CbyC	SDL
Utilización de métodos formales	X	-
Utiliza desarrollo iterativo	X	-
Entrenamiento en temas de seguridad	-	X
Establece requerimientos de seguridad	X	X
Estudia la trazabilidad de los requerimientos	X	-
Asistencia de personal especializado en la seguridad	-	X
Realiza modelado de amenazas	-	X
Análisis estático	X	X
Análisis dinámico	-	X
Fuzz testing	-	X
Revisión de código	X	X
Desarrolla un plan en caso de incidentes de seguridad	-	X

Tabla 2. *Comparativa entre las metodologías CbyC y SDL*

De la tabla anterior se podría llegar a pensar que SDL representa mejor seguridad en el producto final. Sin embargo CbyC enfoca su proceso en crear un producto que sea correcto y con el menor número de defectos posibles. SDL tiene varias actividades durante todo el proceso para educar, para descubrir vulnerabilidades en el código y para reducir al área de ataque maliciosos. Las dos tienen limitaciones respecto a los lenguajes de programación, ya que por una parte CbyC se debe comprar la versión del lenguaje SPARK que soporta exigencias críticas de seguridad; y por otro lado SDL siendo impulsada por Microsoft utiliza lenguajes y suite de programación familiares a la compañía, al igual que las herramientas proporcionadas únicamente corren bajo ambiente del sistema operativo Windows.

5. Conclusiones

La seguridad ha dejado de ser un atributo de calidad, se encuentra en cada capa de la arquitectura de software, y por lo tanto no se puede dejar como un elemento aislado, sino que es transversal y multidimensional. Los hackers parece que siempre están dos pasos más delante de las organizaciones, si se continúa haciendo software de la manera tradicional; esa brecha será aprovechada y seguirán explotando vulnerabilidades que pudieron haber sido evitadas utilizando metodologías como las que en éste documento se citaron.

Este trabajo presentó de manera general, procesos desarrollo de software que tienen actividades enfocadas a la mejora de la seguridad. La utilización de una metodología de éstas características es imperativa para mitigar y tratar de evitar los ataques que día a día, las organizaciones y las personas son víctimas, causando pérdidas millonarias. Recordando que la mejor metodología es aquella que se adapta al contexto del producto a desarrollar, eso da la pauta para escoger la metodología idónea. En un estudio posterior, se podría formular la integración de la versión ágil del SDL con Scrum y aplicarlo en un caso práctico.

Referencias

Amey, P. (2006). Correctness by Construction. Consultado el 29 de septiembre del 2013, en https://buildsecurityin.us-cert.gov/articles/knowledge/sdlc-process/correctness-by-construction

Brito, E. (2010). A (Very) Short Introduction to SPARK : Language , Toolset , Projects , Formal Methods & Certification (pp. 479–490). Portugal: INForum 2010 - II Simpósio de Informática.

Croxford, M., & Chapman, R. (2005). Correctness by Construction : A Manifesto for High-Integrity Software. The Journal of Defense Software Engineering, 18(12), 5–8.

Davis, N. (2005). Secure Software Development Life Cycle Processes : A Technology Scouting Report (pp. 14–20).

Fowler, M., & Highsmith, J. (2001). The Agile Manifesto. Consultado el 07 de julio del 2013, en http://www.pmp-projects.org/Agile-Manifesto.pdf

Hall, A. (2007). Realising the Benefits of Formal Methods. Journal of Universal Computer Science, 13(5), 669–678.

Korkeala, M. (2011). Integrating SDL for Agile in an ongoing software development project. Cloud Software Finland, 1–17.

Laskowski, J. (2011). Agile IT Security Implementation Methodology (primera ed., pp. 13–21). Birmingham, Reino Unido: Packt Publishing Ltd.

Lipner, S. (2004). The Trustworthy Computing Security Development Lifecycle. Annual Computer Security Applications Conference, pp. 2 – 11.

Microsoft Corporation. (2007). The Trustworthy Computing Security Development Lifecycle The Microsoft SDL Team. Consultado en http://www.microsoft.com/en-us/download/details.aspx?id=12379

Norton. (2012). Norton Cybercrime Report 2012 (pp. 1–9). Consultado en http://us.norton.com/cybercrimereport

Peterson, G. (2011). Security Architecture Blueprint. Dublin: Secure Application Development. Consultado en http://secappdev.org/handouts/2011/Gunnar Peterson/ArctecSecurityArchitectureBlueprint.pdf

Vanfosson, T. (2006). Plan-driven vs . Agile Software Engineering and Documentation : A Comparison from the Perspectives of both Developer and Consumer Submitted for the PhD Qualifying Examination. CiteSeerX

Wood, C., & Knox, G. (2012). Guidelines for Agile Security Requirements Engineering. Software Requirements Engineering (pp. 1 –5). Rochester, Nueva York.

Notas biográficas

Carlos Joaquín Brito Abundis tiene el grado de Licenciado en Derecho por la Universidad Autónoma de Durango, actualmente cursa el último semestre de la carrera de Ingeniería en Software en la Universidad Autónoma de Zacatecas. Participó con el proyecto de investigación “Evaluación de la Seguridad de la Red Universitaria mediante Pruebas de Penetración” en el Primer Foro de Jóvenes Investigadores realizado en la ciudad de Fresnillo, Zacatecas. Asimismo, elaboró el poster “Enfoques de Desarrollo para Software Seguro” en el marco del Segundo Congreso Internacional de Mejora de Procesos de Software. Los temas de su interés son pruebas de penetración en dispositivos móviles y la evaluación de la seguridad en las empresas.