COMO CREAR HERRAMIENTAS SEGURAS DE DESARROLLO USANDO CODIGO SEGURO

[*] **Luis Felipe Wanumen Silva**

Proyecto de Investigación Desarrollo de Herramientas para

La creación y manipulación de contenido XML,

Adscrito al grupo METIS de la Facultad Tecnológica

Resumen

El presente artículo muestra que el código seguro es una parte importante para crear aplicaciones dirigidas a desarrolladores de software. De otra parte se deja en evidencia que desde el mismo diseño y concepción de la herramienta de desarrollo se pueden tener prácticas que no permitan que el producto final sea seguro, sin que esto sea responsabilidad de los programadores que en últimas desarrollan la herramienta.

Como contraprestación a la anterior idea, el artículo plantea que también existen cosas en la seguridad de una herramienta de desarrollo que son responsabilidad de los programadores finales, pero que a veces no se tiene en cuenta para construir una herramienta insegura.

Palabras clave: Seguridad, herramienta, software, modelo, códigos, redes, diseño, desarrollo.

[*] Ingeniero de Sistemas, especialista en Ingeniería de Software de la Universidad Distrital Francisco José de Caldas. Docente de la Universidad Distrital, actualmente docente tiempo completo de la Facultad Tecnológica. Correo electrónico: lwanumen@udistrital.edu.co. Actualmente director del Grupo de Investigación Desarrollo de Herramientas para la creación y manipulación de Contenido XML.

Abstract

The present articulates sample that the sure code is an important part to create applications directed to software developers. Of another part it is left in evidence that from the same design and conception of the development tool they can be had you practice that they don't allow that the final product is safe, without this is the programmers' responsibility that in you finish they develop the tool.

As consideration to the previous idea, the one articulates it outlines that things also exist in the security of a development tool that you/they are the final programmers' responsibility, but that one doesn't sometimes keep in mind to build an insecure tool.

Keywords: Security, tool, software, model, codes, nets, design, develop.

CÓDIGO SEGURO EN LAS HERRAMIENTAS DE DESARROLLO

Desarrollar una herramienta de desarrollo debe tener seguridad y dicha seguridad debe garantizar que la herramienta CASE genere lo que debe generar, sirva en el entorno al cual está diseñada y sea utilizada para el fin previamente establecido. Cualquier anomalía en alguna de las anteriores cuestiones se puede tomar como una inseguridad en una herramienta de desarrollo.

LA SEGURIDAD INHERENTE A LAS CARACTERÍSTICAS DEL SOFTWARE

Al pretender hablar de seguridad en el software es importante primero observar las características del software debido a que es gracias a éstas características que pueden ser aplicables algunas políticas de software y otras probablemente no.[El autor]

El software se desarrolla no se fabrica en un sentido clásico [1 Pág. 8]. Esto quiere decir que la seguridad usada en una fábrica por ejemplo que produzca zapatos no será la misma en una empresa que produzca software.

El software no se estropea pero se deteriora [1 Pág. 9]. Esto quiere decir que el software no es como un componente de hardware que se dañe de tanto usarlo, sino que a medida que se usa se hace necesario hacerle algunas mejoras y de tantas mejoras llega el momento en el que no sirve para nada y es mejor cambiarlo. Es entonces importante tener en cuenta esta situación en la seguridad, debido a que el software será cada vez más inseguro cuando se le hagan mejoras y llegará el momento en el que la seguridad se vea tan afectada después de muchos parches que sea mejor pensar en otro software.

SEGURIDAD EN EL PROCESO DE DESARROLLO DE SOFTWARE

El proceso de desarrollo de software debe estar ligado con buenas prácticas que le permitan en un alto grado obtener excelentes productos de software. Existe un modelo CMM que plantea una serie de principios y prácticas que teóricamente conducirían a buenos productos de software. El CMM (Capability Matutity Model) –Modelo de madurez de la capacidad- fue desarrollado por el instituto de Ingeniería del Software[1]

El modelo CMM proporciona una medida de la efectividad global de las prácticas de Ingeniería del software y establece cinco niveles de madurez del proceso, que se definen de la siguiente forma:

Nivel 1: Inicial

Nivel 2. Repetible

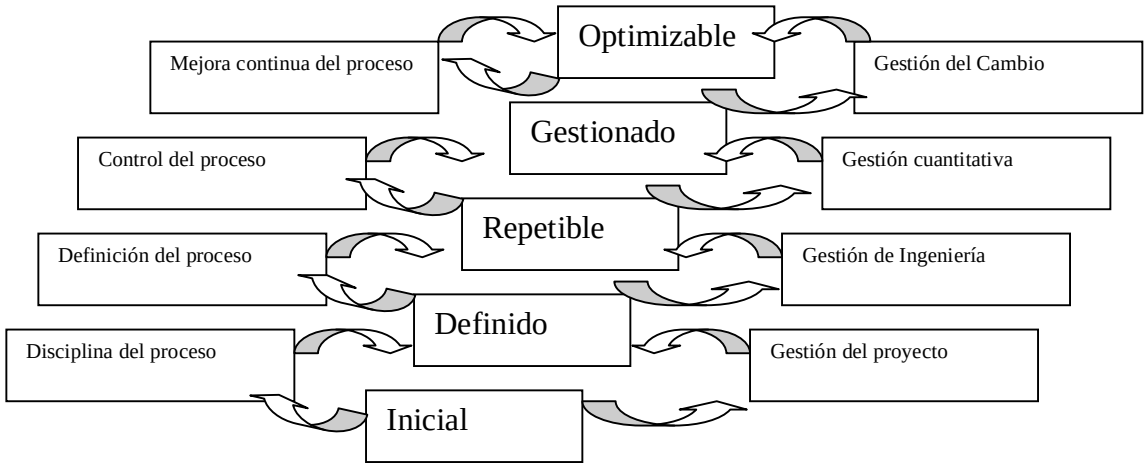
[1] La página oficial del Instituto de Ingeniería delG Software es http://www.sei.cmu.edu

Nivel 3: Definido

Nivel 4: Gestionado

Nivel 5: Optimización

Figura 1. Modelo CMM[2]



Fuente: Esta figura se encuentra disponible en http://www.sei.cmu.edu/cmm

Con la figura anterior es claro observar que los resultados esperados según CMM cuando solamente se gestiona el proyecto son bajos comparados con los resultados esperados cuando a parte de gestionar el proyecto se hace gestión de ingeniería y se hace gestión cuantitativa. En este caso particular se podría pensar que de llevarse a cabo éstas tres gestiones (proyecto, ingeniería y cuantivativa), se tiene un control del proceso de software aunque la organización con este control estaría lista a avanzar en el cuadro CMM y llegar a lograr una mejora continua del proceso si adicionalmente gestiona el cambio. La pregunta que viene es: ¿La gestión del cambio le da mayor seguridad al proceso de desarrollo de software?. La respuesta es sí y no porque intuitivamente se crea, sino porque la gestión del cambio se puede hacer soportada y ayudada en gran parte por herramientas como son los CVS (Colocar

[2] Esta figura se encuentra disponible en http://www.sei.cmu.edu/cmm

significado a las siglas) que entre otras tantas herramientas han sido comprobadas que funcionan en grandes proyectos de software.

LA SEGURIDAD DE UN SISTEMA INCLUYE ASPECTOS DEL MODELAMIENTO

Con el fin de hacer las aplicaciones más seguras desde el modelamiento se deben usar técnicas sofisticadas de modelamiento que permitan no cometer los mismos errores de diseño que se cometen a diario. Los retos futuros para las técnicas formales en tecnologías de objetos incluyen un tratamiento formal de patrones [14] y marcos (arquitecturas específicas de dominios para sistemas de objeto en ese dominio).

¿LA SEGURIDAD A NIVEL DE RED CON TCP/IP ES PLENA?

La seguridad es una preocupación importante siempre que se conecta una estación al exterior. El software básico de TCP/IP **no cifra los datos por sí mismo**, debe realizarlo la aplicación. Si no se cifran los datos, la contraseña se enviará en texto ASCII y podrá ser leída fácilmente durante el trayecto por personas que dispongan de medios y conocimientos. Es importante que los datos (sobretodo la contraseña) vayan cifrados para evitar que sean examinados por personas ajenas.[5 Pág 124]

¿POR QUÉ DEBEMOS CREAR CÓDIGO SEGURO?

Como programadores, es de nuestro máximo interés crear código seguro. Algunas empresas hacen inspecciones del código binario antes de su instalación, lo que puede impedir que vendamos código mal escrito a esas organizaciones. Asimismo, como programadores, incluso si estamos haciendo desarrollo

en código fuente abierto y suministrando el producto tal y como es, es una cuestión de orgullo entregar nuestro código tan libre de fallos como podamos.[4 Pág 706]

- Aunque probablemente no estemos de acuerdo con este punto de vista extremista, el sentimiento general está bien planteado. Los problemas de seguridad pueden provenir de. [4]
- Falta de conciencia sobre la seguridad.
- Pobre diseño del código
- Pruebas de código pobres.
- Imprevisión de posibilidades.
- Código demasiado complicado.
- Código demasiado sencillo.
- No comprobar la entrada.
- Mala comprobación de los límites.
- Condiciones de adelanto.

Es importante notar que el código demasiado complicado afecta la seguridad de una herramienta de desarrollo y esta complicación puede ser causada desde el mismo análisis y diseño de la herramienta. Por ejemplo “El abuso del polimorfismo y del enlace dinámico, hacen difícil el control de la versión de un método, que será realmente ejecutado, en una invocación particular” [12]. El pobre diseño del código también puede deberse a una mala utilización de los mecanismos de la herencia. Por ejemplo “La herencia puede ser usada para fragmentar la definición de un método a través de muchas clases, haciendo su significado difícil de determinar”[13]

LA SEGURIDAD CUANDO EL SOFTWARE ES CONCURRENTE

La seguridad de un proyecto de software concurrente podría pensarse que se llega a medir en el momento en el que se termine el software, pero la verdad es que en muchas de las etapas del desarrollo del mismo se pueden hacer éstas mediciones.. Por ejemplo "Un objetivo importante del diseño del sistema es identificar los objetos que deben estar activados concurrentemente, y los objetos que tienen actividad que sea mutuamente exclusiva. Estos últimos objetos se pueden plegar y juntar en único hijo de control, o tarea"[2 Pág 270]

En la metodología OMT se habla que el modelo dinámico es la guía para identificar la concurrencia. Dos objetos inherentemente concurrentes si pueden recibir sucesos al mismo tiempo sin interactuar. Si los sucesos no están sincronizados, los objetos no se pueden plegar en un único hilo de control. Los diagramas de estado de objetos individuales según ésta metodología y de intercambio de sucesos entre ellos, pueden mostrar si es posible encontrar que muchos objetos se puedan plegar en un único hilo de control. Un hilo de control es una vía a través de un conjunto de diagramas de estados en la cual solamente está activado un objeto en cada instante. Permanece dentro del diagrama de estados hasta que un objeto envía un suceso a otro objeto y espera otro suceso. El hilo pasa el receptor del suceso hasta que, eventualmente vuelve al objeto original. Se prescinde si el objeto envía un suceso y sigue ejecutándose. [2 Pág. 271].

Imagínese por un momento los múltiples flujos de control de un sistema concurrente. Cuando el flujo pasa a través de una operación, se dice que en un momento dado el lugar donde se encuentra el control es la operación. Si esa operación está definida en alguna clase, también se dice que en un momento dado el lugar donde se encuentra el control es una instancia específica de esa clase. Puede haber varios flujos de control en una misma operación (y por lo tanto en un objeto), y puede haber diferentes flujos

de control en diferentes operaciones (pero que todavía producen varios flujos de control en el objeto. El problema aparece cuando en un instante dado existe más de un flujo de control en un objeto. Si no se tiene cuidado, los flujos pueden interferir entre sí, corrompiendo el estado del objeto. Este es el problema clásico de la exclusión mutua. Si no se trata correctamente, se producen toda clase de condiciones de competencia e interferencias, que hacen que los sistemas concurrentes fallen de formas misteriosas e irrepetibles.

Para solucionar el problema anterior algunos autores plantean tres alternativas, cada una de las cuales implica asociar ciertas propiedades de sincronización a las operaciones definidas en la clase. En UML se pueden modelar los tres enfoques: [6 Pág. 276]

Secuencial

Los emisores deben coordinarse fuera del objeto para que en un instante dado sólo exista un flujo de control en el objeto. En presencia de varios flujos de control, no se puede garantizar la semántica ni la integridad del objeto.

Con guardas

La semántica y la integridad del objeto se garantizan en presencia de múltiples flujos de control, tratando secuencialmente todas las llamadas a las operaciones con guardas del objeto. En efecto, en un momento dado sólo puede ser invocada una operación sobre el objeto, lo que lleva a una semántica secuencial.

Concurrente

La semántica y la integridad del objeto se garantizan en presencia de múltiples flujos de control tratando la operación como atómica.

EL USO DE LA VIRTUALIDAD MEJORA LA SEGURIDAD

Se dice que la virtualidad mejora la seguridad de una aplicación cuando se usa en forma apropiada y no excede su uso. Para el caso del lenguaje c++ tenemos sobre la virtualidad que una llamada a un método virtual se resuelve siempre en función del tipo del objeto referenciado. Una llamada a un método normal (no virtual) se resuelve en función del tipo del puntero o de la referencia. Una llamada a un método virtual específico exige utilizar el operador (**::**) de resolución de ámbito [3 Pág. 602]. Para comprender mejor esto recordemos que si tenemos una función "f()"que recibe una referencia a un objeto y ejecuta de este último un método virtual "v1" el compilador buscará el método "v1" basándose en el tipo del objeto que se pasó inicialmente a la función "f()" y no del tipo del objeto que se encuentra en la declaración y definición del método "f()" con lo cual se garantiza que cada objeto invoque necesariamente a métodos de su misma clase.(Obviamente partiendo del hecho que estos métodos virtuales han sido sobrescritos en cada una de las subclases). Esto sería especialmente útil cuando se intenta asegurarse que en verdad los objetos que llamen ejecuten sus propios métodos y la forma de violar un sistema con esta programación es difícil debido básicamente a que si un objeto hijo se pasa a la función "f()" haciéndose pasar por otro y para ello usando las conversiones implícitas que tiene c++, no podría ejecutar el método "v1" de la clase padre, sino únicamente el del tipo de clase que instancia verdaderamente el objeto que fue pasado a la función "f()".[El autor]

Como gran conclusión sobre esta parte se pude decir que si no se tiene control sobre la autenticidad de los objetos que invocan ciertos métodos de nuestras aplicaciones haciendo uso de los métodos virtuales logramos que verdaderamente el objeto físicamente tenga que ser de determinado tipo para ejecutar dichos métodos virtuales.(Recordemos que obviamente estos métodos virtuales se recomienda que estén sobrescritos en las clases derivadas para que el ejemplo tenga coherencia)[El autor.]

En forma análoga se puede pensar que el uso de métodos no virtuales es aconsejable cuando se tenga total control sobre el tipo y la autenticidad de los objetos que invocan las funciones que después a su vez invocan los métodos de las clases. Es entonces pues decisión del desarrollador defender el uso o no uso de la virtualidad dependiendo la situación específica que se esté tratando.

EL USO ADECUADO DE EXCEPCIONES MEJORA LA SEGURIDAD

No todos los programas necesitan responder lanzando una excepción a cualquier situación anómala que se produzca. Por ejemplo, si partiendo de unos datos de entrada estamos haciendo una serie de cálculos más o menos complejos con la única finalidad de observar unos resultados, quizás la respuesta más adecuad a un error sea interrumpir sin más el programa, no antes de haber lanzado un mensaje apropiado y haber liberado los recursos adquiridos que aún no hayan sido liberados. Otro ejemplo, podemos utilizar la clase excepción que sirva para manejar el error que se produce cuando se rebasan los límites de una matriz, pero es más fácil usar una sentencia if para prevenir que esto no suceda. En cambio si estamos construyendo una biblioteca, estaremos obligados a evitar todos los errores que se puedan producir cuando su código sea ejecutado por cualquier programa que la utilice. Por último, no todas las excepciones tienen que servir para manipular errores. Puede también manejar excepciones que no sen errores.[3 Pág. 719]

EL CODIGO FUENTE MALIGNO AUMENTA LA INSEGURIDAD

Las aplicaciones de interfaz gráfica de usuario suelen estar escritas en código fuente que los ingenieros compilan en una versión de ejecución. El hardware de sobremesa almacena la versión de ejecución, o el servidor guarda la versión de ejecución y se descarga al hardware de sobremesa cuando se realiza la autenticación de usuario. Si se activan las aplicaciones desde el exterior de sus entornos normales, una

función legítima puede tener resultados no deseados si esta activación se realiza en un momento poco adecuado. Por ejemplo una persona curiosa podría provocar la activación de una herramienta de captura de pantalla durante el proceso de inicio de sesión de usuario para capturar el Id de inicio de sesión de usuario, la contraseña y otra información [9 Pág. 280]

HERRAMIENTAS DE DESARROLLO MULTIUSUARIO DEBEN TENER OTROS FACTORES EN CUENTA PARA LA SEGURIDAD

En la actualidad muchas de las herramientas de desarrollo son multiusuarios, es decir herramientas que permiten desarrollar sistemas entre varios desarrolladores. Este tipo de herramientas de desarrollo son bastante complejas por cuanto la seguridad en ellas de implicar a demás de las seguridades de código fuente, sistemas que permitan administrar los archivos que se compartan [el autor].

El control del código fuente generado por ejemplo con herramientas como las de Microsoft para el caso de aplicaciones Web incluye la administración de archivos compartidos. “Para administrar los cambios en los archivos Web en Microsoft, se puede usar Visual Source Safe junto con Microsoft Visual Studio. Mediante el control de código fuente, los integrantes del equipo de Web pueden compartir archivos, modificarlos de forma independiente y, más adelante, combinar los cambios. Visual SourceSafe también guarda las versiones anteriores de los archivos en una base de datos, controla la fecha y hora de los cambios y proporciona una opción para mantener un registro de comentarios” [10 Pág. 97].

JAVA COMO LENGUAJE PARA CREAR HERRAMIENTAS DE DESARROLLO SEGURAS

La ventaja de Java es que sus API están en constante evolución [7 Pág. 852]. Las opciones de seguridad de la plataforma Java 2 han sido ampliadas con varias extensiones orientadas a la seguridad:

La Java Cryptography Extensión: Extensión criptográfica de Java (JCE) 1.2 proporciona la base para el desarrollo de la encriptación, la generación de claves, el acuerdo de claves y los algoritmos de autentificación. También proporciona las implementaciones de algunos algoritmos criptográficos conocidos [9 Pág. 93]

La Java Secure Socket Extensión, Extensión de socket seguro de java (JSSE), ofrece una implementación Java de los protocolos Secure Sockets Layer(SSL, Capa de Sockets segura) en su versión 3 y Transport Layer Security (TLS, seguridad en la capa de transporte) en su versión 1. Estos protocolos se pueden emplear para añadir seguridad a los protocolos Internet existentes, como http, Telnet y POP3 [9 Pág. 253].

El Java Authentication and Authorization Service, Servicio de Autentificación y Autorización de Java (JAAS), ofrece soporte para la autentificación de usuarios y los controles de acceso. Proporciona una implementación Java del Pluggable Authentication Module, Módulo de autentificación conectable (PAM), así como clases e interfaces que controlan que usuarios son capaces de ejecutar software sensible a la seguridad.[9 Pág. 331].

En Java cuando se trata de applets: “El SecurityManager proporciona los medios para establecer y configurar muchas de las inspecciones de seguridad que monitorean la potencia de sus programas. Estas inspecciones son necesarias porque los applets de Java tienen la habilidad de correr en intranets al igual que en Internet (redes donde la seguridad y privacidad son factores importantes)”[9 Pág. 217]

CONCLUSIONES

Desarrollar una aplicación segura se logra no solamente con buenos algoritmos de seguridad, sino que la seguridad de un producto final de software se siembra desde el mismo momento en el que se gestiona el proyecto de software y debe estar presente en todo el ciclo de vida y de desarrollo del mismo para lograr que verdaderamente la seguridad en una herramienta sea una realidad por lo menos en un alto grado, dado que la seguridad plena en software nunca existe.

La seguridad en cualquier aplicación se logra con una mentalidad clara de seguridad en la planeación, el diseño, las implementación, el despliegue, etc. y éste razonamiento se puede extender también al desarrollo de herramientas CASE, por cuanto todas las reglas expuestas en el presente artículo son también válidas para sistemas CASE.

Algo oculto que muy pocas empresas tienen en cuenta a la hora de desarrollar un software seguro es tener buenas prácticas de diseño, pues un mal diseño afecta en gran medida la seguridad de cualquier sistema que se construya y en el caso específico de este artículo el desarrollo de una herramienta CASE.

No basta con tener un buen diseño y con tener una planeación y una gestión del proyecto que permita construir software CASE seguros, sino que también se hacen necesarias las implantaciones de códigos seguros que muy pocas veces la gente analiza.

La construcción de código seguro es algo plenamente difícil y exige incluso la especialización de los profesionales dedicados al desarrollo del mismo.

Bibliografía

[1] Ingeniería del Software. Un enfoque práctico. Cuarta edición. Roger S. Pressman. Mc Graw Hill. ISBN: 84-481-1186-9

[2] Modelado y diseño orientado a objetos. Metodología OMT. James Rumbaugh, Michael Blaha, William Premerlani, Frederick Hedí y William Lorensen. Prentice Hall. ISBN: 0-13-240698-5

[3] Enciclopedia del lenguaje C++. Francisco Javier Cevallos. Alfaomega Ra-ma. ISBN:970-15-0964-1

[4] Seguridad en Microsoft Windows 2000. Guía Avanzada. Jeff Schmidt. Prentice Hall.

[5] Redes Locales. José Luis Raya, Cristina Raya. Alfaomega Ra-ma. ISBN: 970-15-0712-6

[6] El lenguaje unificado de modelado. Booch, Jacobson y Rumbaugh. Addison Wesley. ISBN: 84-7829-028-1

[7] Cómo programar en Java. Deitel y Deitel. Prentice Hall. Pearson Educación. ISBN: 970-17-0044-9

[8] Seguridad en Java. Edición especial. Jaime Jaworski, y Paul J. Perrone. Prentice Hall. Pearson Educación. ISBN: 84-205-3134-0

[9] Microsoft Windows 2000 Server. Administración y control. Kenneth L. Spencer y Marcus Goncalves. Prentice Hall. Pearson Educación. ISBN: 84-205-2977-X

[10] Microsoft Visual Interdev 6.0. Manual del programador. Microsoft Press. Mc Graw Hill. ISBN: 84-481-2078-7

[11] Java Soluciones Instantáneas. Michael Aferran. PHP Prentice Hall Hispanoamericana, S.A. ISBN: 968-880-804-0

[12] Polymorphism considered barmful. Ponder C. Y Brush B. ACM Sigplan Notices, 27(6), junio de 1992.

[13] Maintenance support for object-oriented programs. Wide N. y Huit R. En proccedings of Conference on Software Maintenance. IEE Computer Society Press,1991

[14] Smaltalk-80. The language and its implementation. Autores: Goldberg A. y Robson D. Addison Wesley,1983