Universidad de las Ciencias Informáticas

Trabajo de Diploma para optar por el título de Ingeniería en Ciencias Informáticas

Propuesta de SDL como una metodología de desarrollo de software seguro en la UCI

Autores: Mailin Gradaille Herrera
Yaislenis Landabe Barbarú

Tutor: Ing. Yohamnes Hernández Báez

Ciudad de La Habana, junio de 2007

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Universidad de la Ciencias Informáticas a que haga el uso que estime pertinente con el mismo.

Para que así conste firmamos la presente a los _____ días del mes de _____ de _____.

_____ _____

Yaislenis Landabe Barbarú — Autor

Mailin Gradaille Herrera — Autor

_____

Ing. Yohamnes Hernández Báez

Tutor

OPINIÓN DEL TUTOR SOBRE EL TRABAJO DE DIPLOMA

Título: Propuesta de SDL como una metodología de desarrollo de software seguro en la UCI

Autores: Yaislenis Landabe Barbarú
Mailin Gradaille Herrera

El tutor del presente Trabajo de Diploma considera que durante su ejecución las estudiantes mostraron las cualidades que a continuación se detallan:

Se demostró un alto grado de independencia y originalidad al enfrentarse a tareas ante las cuales no se tenía conocimiento ninguno y que solo pudieron superarse con gran empeño y laboriosidad. La responsabilidad con el trabajo a cumplir fue otro factor característico y que sin él, no hubiera sido posible llegar a los resultados obtenidos.

El trabajo que aquí se presenta tiene un alto nivel científico. El tema escogido es de gran actualidad e importancia en el desarrollo actual y futuro del software. La bibliografía consultada está de acuerdo a las últimas prácticas seguidas a nivel mundial.

Por todo lo anteriormente expresado, considero que las estudiantes están aptas para ejercer como Ingeniero Informático; y propongo que se le otorgue al Trabajo de Diploma la calificación de 5 puntos. Se propone que el trabajo forme parte de publicaciones y eventos, y que sirva de punto de arranque para una revolución en el desarrollo del software en nuestra universidad.

Ing. Yohamnes Hernández Báez

21 de junio de 2007

Pensamiento

¿Por qué esta magnífica tecnología científica, que ahorra trabajo y nos hace la vida más fácil, nos aporta tan poca felicidad? La repuesta es ésta: simplemente, porque aún no hemos aprendido a usarla con tino.

Albert Einstein

La fórmula del éxito es simple: haz tu mejor esfuerzo, y acaso le agrade a la gente.

Proverbio mexicano

Agradecimientos

***A mi mamá y mi abuela**, por quererme tanto, haberme guiado siempre, por soportar todos mis momentos de malacrianza.*
***A Dora (mi madre) y Alcides (mi padre)**, por quererme y guiarme estos años de universidad, por soportarme tanto, por ayudarme en mí trabajo de diploma.*
***A toda mi familia en general**, por su apoyo, su cariño y comprensión.*
***A mis hermanitos (Annia y Tito)**, por su cariño.*
***A todos mis amigos**, por los momentos inolvidables que pasamos juntos, por sus momentos de dedicación.*
***A mi tutor**, por su apoyo y ayuda para la realización de este trabajo.*
***A nuestro Comandante**, por hacer realidad este sueño tan maravilloso que es la UCI.*

Yaislenis

A todas las personas que me han apoyado y me han dado razones para vivir y avanzar, a los que me brindan su amor constantemente, a mi familia y amigos; a mi mamá, mi hermana, mi madre-abuela querida, a mi compañera de tesis Yaislenis, a esta Universidad por todos los recursos que nos confió y puso a nuestra disposición para nuestro desarrollo profesional, a mis queridos suegros: Martha Gómez y Domingo Cabrera, por aceptarme en sus vidas, y en especial a David, mi amor, por enseñarme el significado de estar vivo.
A todos gracias por el cariño.

Mailin

Dedicatoria

A mi mamá, mi papá que me acompaña siempre, mi abuela, mi madre Dora Emma Nicó, mis amigos, mi familia.

Yaislenis

A Teresa Vega, la mujer más grande que he conocido en mi vida y que me ha enseñado todo lo bueno que soy y que sé. A Manuel Gradaille, a Maritza Herrera, a Liliam Peraza y a Mayelin Gradaille, mi familia querida, por siempre confiar en mí. Y le dedico este trabajo también a mi abuelo querido, que vive por siempre en mi memoria.

Mailin

Resumen

En este trabajo se propone el uso de una metodología de seguridad durante el desarrollo de software, específicamente en la Universidad de las Ciencias Informáticas (UCI). Se parte de una investigación científica que se realizó en los Proyectos Productivos de la Facultad 2 y de las consultas bibliográficas que se han desarrollado, haciendo uso de los conocimientos de Seguridad Informática adquiridos en los cursos docentes durante la carrera de Ingeniería Informática en la UCI; de esta forma se le da solución a la necesidad vigente de controlar la seguridad de un software a medida que éste se va desarrollando; y también se da respuesta a la pregunta que muchos desarrolladores de software y programadores se hacen: ¿cómo obtener un software potencialmente seguro y confiable?

Todo este trabajo se hizo debido a que el desarrollo seguro de aplicaciones es una de las áreas en las cuales los profesionales están menos preparados, aún siendo las vulnerabilidades en las aplicaciones la principal causa de incidentes de seguridad en los servidores Web. La seguridad aplicada a la informática es un tema que está vigente y estará; con el crecimiento y evolución de los mercados, de la piratería, de los delitos informáticos, el avance y enriquecimiento cada día más del conocimiento de los crakers, entre otras vulnerabilidades existentes, es fundamental la necesidad de tener desarrollada la seguridad desde el inicio de la programación.

En este trabajo encontrará la descripción y fundamentación de una metodología de programación segura que se aconseja utilizar en todos los proyectos de programación en los que necesite desarrollar un modelo o aplicación de software.

Índice

Introducción

La seguridad informática es vital para las empresas. Es cada vez más importante identificar al trabajador o usuario que accede a un ordenador o a un software. Mantener un sistema informático o una red informática libre de intrusiones no es una tarea fácil. Si se desea un sistema seguro deberá esforzarse por ello, emplear contraseñas fuertes y diferentes y seguir todo un procedimiento para mantener el sistema libre de amenazas.

La necesidad de generar mejores soluciones sistematizadas, que den mayor valor agregado a sus clientes, se comprueba con las presiones e implicaciones que esto lleva en el desarrollo mismo de dichas soluciones. En este sentido, balancear la necesidad de obtener un producto de software de óptima calidad que permita a la organización adelantar con oportunidad y alto contenido estratégico sus directrices de negocio, enfrenta un doble desafío para los dedicados a la programación y construcción de los sistemas de información de las organizaciones, así como para los clientes de los mencionados sistemas.

En esta confluencia de formalidad administrativa y técnica, la construcción de sistemas de información ofrece un desafío práctico para las nuevas generaciones de programadores y administradores de proyectos informáticos, y una especial atención de los experimentados ingenieros de software para contextualizar sus aprendizajes en elementos conceptuales y formales que alimenten la práctica de la creación de software.

Por tanto y, considerando el gran reto de la construcción de software, la necesidad de soluciones de software intercomunicadas (orientadas al uso de redes de computadoras), la utilización de modernos lenguajes de programación (JAVA, PYTHON, PERL, PHP, entre otros), la necesidad de soluciones eficientes y de alta portabilidad, el uso de metodologías de aseguramiento de calidad, y la informatización que se lleva a cabo en nuestro país, se hace necesario revisar elementos relacionados con las prácticas y principios de programación segura como aspecto complementario del proceso de desarrollo de software.

Entre los principales problemas de seguridad que existen está el acceso no autorizado a una red informática o a los equipos que en ella se encuentran, lo que puede ocasionar la pérdida de los datos o la indisponibilidad de los mismos. Otro de los problemas más dañinos es el robo de información sensible y confidencial; la divulgación de la información que posee una empresa sobre sus clientes, por ejemplo, puede acarrear demandas millonarias contra la empresa.

Las técnicas más utilizadas para el robo de datos son:

1. *Ingeniería Social*
2. *El navegador de Internet*
3. *Phishing*
4. *Keystroke logging*
5. *Sniffing* [13]

Sobre estos temas muchos investigadores han adelantado estudios interesantes, buscando elementos comunes que permitan orientar a los desarrolladores de software en estrategias y principios que disminuyan potencialmente los problemas de seguridad.

Por tanto: el uso de una metodología que sirva como guía tanto a principiantes como a experimentados es la solución inmediata más eficaz para desarrollar un software seguro y con calidad.

El problema real que se refleja en este trabajo es la no aplicación de una metodología de seguridad en el ciclo de desarrollo de un software que sirva como guía para la construcción de los mismos incluyendo en ésta la seguridad desde un principio y pueda ser utilizado por todo tipo de desarrollador para un progreso más seguro de los software que se produzcan en la Universidad de las Ciencias Informáticas.

El objeto de estudio de este trabajo es el desarrollo de la seguridad en el ciclo de vida del software.

Este trabajo está orientado básicamente a los proyectos productivos de la Universidad de las Ciencias Informáticas, proponiendo el uso de esta metodología de control de la seguridad para todo producto que se cree o modele.

El objetivo general de este trabajo es proponer el uso de una metodología para el desarrollo seguro de software, la cual guíe a todos los desarrolladores y/o programadores de software que opten por un producto con mayor calidad y confiabilidad en la UCI.

Como objetivos específicos se tienen:

- Sistematizar los conceptos relacionados con la seguridad informática
- Estudiar las metodologías existentes en empresas de desarrollo de software
- Analizar detalladamente cuál es la mejor metodología a usar en la Universidad de las Ciencias Informáticas
- Presentar los resultados de su posible uso en los proyectos UCI

Como resultado de este trabajo se pretende proponer una metodología de seguridad para ser usada en todo el proceso de desarrollo de un software, lo que haría más eficiente su elaboración siguiendo paso a paso la seguridad con todos sus principios desde el inicio, garantizando mayor confiabilidad en el producto y la disminución de los posibles ataques y vulnerabilidades que puedan existir con el constante desarrollo de la informática en todo el mundo.

Para dar cumplimiento a estos objetivos se han propuesto una serie de tareas, que se mencionan a continuación:

- Estudio de otras metodologías relacionadas con el tema

- Estudio de libros y artículos relacionados con la ingeniería de software y la seguridad

- Aplicación de encuestas para conocer los diferentes criterios sobre seguridad en la ingeniería del software que se aplica en la UCI

- Revisión de sitios internacionales referentes al uso de estas metodologías en las distintas universidades existentes en el mundo

Para realizar estas tareas se utilizarán los siguientes métodos:

Métodos teóricos: permiten estudiar las características del objeto de investigación que no son observables directamente, facilitan la construcción de modelos e hipótesis de investigación y crean las condiciones para ir más allá de las características fenomenológicas y superficiales de la realidad.

Analítico – sintético: procesos que permiten, como métodos teóricos, buscar la esencia de los fenómenos, los rasgos que los caracterizan y los distinguen. Su objetivo en una investigación es analizar las teorías, documentos, permitiendo la extracción de los elementos más importantes que se relacionan con el objeto de estudio.

Métodos empíricos: estos métodos permiten extraer de los fenómenos analizados las informaciones que se necesitan sobre ellos a través de observaciones, del uso de técnicas opináticas y la propia experimentación.

Observación: registro visual de lo que ocurre en una situación real, en un fenómeno determinado, clasificando y consignando los hechos y acontecimientos pertinentes de acuerdo con algún esquema previsto, es decir, se realiza una guía de observación para ver cómo se comporta el objeto de estudio.

Encuesta: la encuesta es semejante a la entrevista pero escrita, donde a través de un conjunto de preguntas se pretende obtener una información sobre el mundo interior del encuestado o su percepción del fenómeno que se investiga.

El contenido de este trabajo se encuentra estructurado de la siguiente manera:

En el capítulo I, “Fundamentación Teórica”, se recoge todo lo que concierne al objeto de estudio, se hace una revisión de la bibliografía donde se analizan los principales conceptos sobre la seguridad, su importancia, las diferentes vulnerabilidades que puede tener un software, los principios de un diseño seguro, las normas, políticas y controles de seguridad, entre otros contenidos referentes a la seguridad.
En el capítulo II, “Etapas del desarrollo del software“, se exponen todas las fases por donde transita un software en su desarrollo, así como los diferentes flujos de trabajo que se llevan a cabo en las etapas, además de los resultados de la encuesta aplicada a los líderes de proyectos de la facultad 2.

En el capítulo III, “Aplicando SDL en cada fase del software“, se describen todos los pasos que se deben tener en cuenta para aplicar la metodología SDL en el software que se realiza.

Capítulo I: “Fundamentación Teórica”

Introducción

A finales del siglo XX tanto las empresas, organismos e incluso particulares comienzan a tomar verdadera conciencia de la importancia de la seguridad. Hoy día, tener un sistema que cumpla con los estándares de gestión de la seguridad es sinónimo de calidad de servicio. [1]

En este capítulo se abordará todo lo relacionado con la seguridad informática a nivel nacional e internacional, los diferentes conceptos enunciados por varias personalidades dedicadas a este aspecto tan importante en el mundo virtual, su importancia, los principales principios para un diseño seguro, las vulnerabilidades del software, los controles de seguridad, las políticas de seguridad, las normas básicas de seguridad y el por qué usar SDL.

1.1- Seguridad Informática

La seguridad informática es una necesidad creciente en todas las organizaciones. Hoy día, conforme se brindan servicios tanto a través de Internet como de Intranet, se abren oportunidades para que personas con cierto nivel de conocimiento o con ninguno pero con poderosas herramientas puedan no sólo hacer colapsar los sistemas, sino también robar uno de los activos más valiosos de las organizaciones: La Información.

No existe una definición única y exacta de seguridad informática o seguridad en cómputo, sin embargo es posible enunciar algunos conceptos que involucran los diversos factores que la conforman:

“Seguridad informática es el conjunto de recursos (métodos, documentos, programas y dispositivos físicos) encaminados a lograr que los recursos de cómputo e información, disponibles en un ambiente dado, sean accedidos única y exclusivamente por quienes tienen la autorización para hacerlo.”

“Seguridad informática es mantener bajo protección los recursos y la información con que se cuenta en la red.”

“Seguridad en cómputo es un conjunto de recursos destinados a lograr que los activos de una organización sean confidenciales, íntegros y disponibles para sus usuarios.” [4]

“Seguridad en el entorno de la informática es todo proceso que nos permite preservar la información y los objetos de nuestro sistema de los malos usos voluntarios o involuntarios de los usuarios.” [5]

“Seguridad informática es el estudio de los métodos y medios de protección de los sistemas de información y comunicaciones, frente a revelaciones, modificaciones o destrucción de la información, o ante fallos de proceso, almacenamiento o transmisión de dicha información, que tienen lugar de forma accidental o intencionada.”

La seguridad de la información se caracteriza como la protección frente a las amenazas de:

- Confidencialidad: garantiza que la información es accesible exclusivamente a quien está autorizado. Existen infinidad de posibles ataques contra la privacidad, especialmente en la comunicación de los datos. La transmisión a través de un medio presenta múltiples oportunidades para ser interceptada y copiada: las líneas "pinchadas", la intercepción o recepción electromagnética no autorizada o la simple intrusión directa en los equipos donde la información está físicamente almacenada.

- Integridad: protege la exactitud y totalidad de la información y sus métodos de proceso; bien durante el proceso de transmisión o en su propio equipo de origen. Es un riesgo común que consiste en que el atacante simplemente intercepta y borra un paquete de información, sabiendo que es importante, al no poder descifrarlo.

- Disponibilidad: garantiza que los usuarios autorizados tienen acceso a la información y a otros activos asociados en el momento que lo requieren; esto se resume en evitar su pérdida o bloqueo, bien sea por ataque doloroso, mala operación accidental o situaciones fortuitas o de fuerza mayor. [6]

Luego de un estudio sobre los diferentes criterios acerca del concepto de seguridad informática se concluye diciendo que:

Seguridad informática está definido como el conjunto de procedimientos y actuaciones encaminados a conseguir la garantía de funcionamiento del sistema de información, obteniendo eficacia (entendida como el cumplimiento de la finalidad para el que estaba establecido), manteniendo la integridad (entendida como la inalterabilidad del sistema por agentes externos al mismo), y alertando la detección de actividad ajena (entendida como el control de la interacción de elementos externos al propio sistema). Si se consigue todo esto, tarea harto difícil vaya por delante, se puede decir que disponemos de un sistema convenientemente seguro.

1.2 - ¿Por qué es importante la seguridad informática?

Es importante controlar la seguridad en los sistemas informáticos por la existencia de personas ajenas a la información, también conocidas como piratas informáticos o hackers, que buscan tener acceso a la red para modificar, sustraer o borrar datos; tales personajes pueden incluso, formar parte del personal administrativo de sistemas de cualquier compañía. De acuerdo con expertos en el área, más del 70 por ciento de las violaciones e intrusiones a los recursos informáticos se realiza por el personal interno, debido a que éste conoce los procesos, metodologías y tienen acceso a la información sensible, es decir, a todos aquellos datos cuya pérdida puede afectar el buen funcionamiento de la organización.

Esta situación se presenta gracias a los esquemas ineficientes de seguridad con los que cuenta la mayoría de las compañías a nivel mundial, y porque no existe conocimiento relacionado con la planeación de un esquema de seguridad eficiente que proteja los recursos informáticos de las actuales amenazas combinadas.

El resultado es la violación de los sistemas, provocando la pérdida o modificación de los datos sensibles de la organización, lo que puede representar un daño con valor de miles o millones de dólares. [11]

Es por la existencia de un número importante de amenazas y riesgos, que la infraestructura de red y recursos informáticos de una organización deben estar protegidos bajo un esquema de seguridad que reduzca los niveles de vulnerabilidad y permita una eficiente administración del riesgo.

1.3 – Principios del diseño seguro

La formalidad en la evaluación del diseño de la solución desde el punto de vista de la seguridad es un punto en el que muchos investigadores a nivel internacional han adelantado estudios interesantes (LOSCOCCO, SMALLEY et al. 1999; COWAN, C., WAGLE, P et al. 1999; BISHOP, M. 1999), buscando elementos comunes que permitan orientar a los programadores en estrategias y principios que disminuyan potencialmente los problemas de seguridad asociados con el desarrollo del software.

Considerando estos esfuerzos internacionales y tratando de establecer elementos básicos que orienten criterios en el diseño seguro de aplicaciones, se sugieren algunos principios de diseño que se enumeran y comentan a continuación: (GALVIN, P. 1998; BISHOP, M. 2003, Cáp.13)

1. Menor Privilegio:

Este principio establece que un sujeto sólo debe dar a un objeto los privilegios que necesita para completar sus tareas asignadas.

2. Economía y simplificación de mecanismos de seguridad

Este principio establece que los mecanismos de seguridad que se establezcan deben ser tan sencillos como sea posible.

3. Configuraciones por defecto seguras.

Este principio comenta que a menos que un sujeto haya otorgado acceso explícito a un objeto, éste no debería tenerlo. Es decir, todo lo que no está estrictamente permitido es prohibido.

4. Mediación completa

Este principio afirma que todos los accesos a un objeto(s) deben ser verificados para asegurarse de que cuentan con el permiso para hacerlo.

5. Diseño Abierto

Este principio establece que la seguridad de un mecanismo no debería depender del secreto o confidencialidad de su diseño o implementación.

6. Privilegios Condicionados

Este principio dice que se deben mantener los privilegios necesarios en diferentes momentos, en diferentes rutinas o programas. Es decir, los privilegios no deben ser estáticos para los programas o rutinas en el tiempo y en ejecución.

7. Menor mecanismo común

Este principio comenta que debe existir el menor número de recursos compartidos entre sujetos u objetos.

8. Aceptación psicológica

Este principio expone que el mecanismo de seguridad que se establezca para un objeto no debe sugerir mayor dificultad a la que se establece si el mecanismo no estuviese presente. En otras palabras: el mecanismo de seguridad deber ser fácil de usar.

Estos principios básicos enunciados, más que sugerir elementos novedosos al diseño de aplicaciones, nos recuerda que son prácticas generales que intuitivamente se manejan, pero que en el momento de la construcción de aplicaciones generalmente se dejan marginados. Por tanto, los principios de un diseño seguro, son directrices generales que deben materializarse y que deben ser parte de las formalidades del desarrollo del software mismo.

Cuando estas mínimas sugerencias de diseño seguro no se consideran en la construcción de aplicaciones, la probabilidad de que surjan problemas de seguridad en el futuro es alta, dado que se compromete no solamente la funcionalidad de la aplicación sino también las condiciones de su elaboración y ambiente de ejecución que puede socavar la confianza de los clientes frente a fallas donde se comprometa la integridad de la información que posee o pertenece a la organización.

En este sentido, se abre la posibilidad de que la variedad de la inseguridad informática del sistema (CANO 2004) supere las medidas y controles establecidos, haciendo más vulnerable el software que se construye. El software es una disciplina que debe ir más allá de la funcionalidad de los módulos y considerar en todas sus etapas las propiedades sistémicas que sus componentes exhiben. [3]

1.4 - Fuentes de Vulnerabilidad en el Software

Complementario a los elementos establecidos alrededor de los principios de diseño seguro, es importante considerar e identificar elementos prácticos al buscar y establecer vulnerabilidades o fallas de seguridad en el software. Los elementos presentados a continuación responden a experiencias en el desarrollo de software que de alguna manera materializan la ausencia del uso de estándares de desarrollo de software y de adecuadas prácticas de programación, las cuales se encuentran

directamente relacionadas con los escenarios de prueba requeridos para verificar las condiciones y confiabilidad del software:

1. Cambios en el ambiente de ejecución

Los parches, los cambios en la configuración y variables de entorno alrededor de las aplicaciones son elementos críticos para mantener una ejecución adecuada y controlada de las rutinas y acciones previstas en el software. Al descuidar este aspecto, es probable involucrar efectos de borde o condiciones de excepción no previstas que comprometan no solamente un módulo de la aplicación sino el sistema de información mismo.

2. Desbordamientos y chequeos de sintaxis

Dos elementos importantes en la revisión y evaluación de software. Por un lado la evaluación de los desbordamientos bien sea de memoria o de variables específicas dentro de un programa y por otro lado, la verificación de buen uso de los comandos o palabras reservadas en el lenguaje de programación, que permitan al programador un uso adecuado y eficiente de las estructuras. Si este aspecto no se considera con el rigor necesario, se estará comprometiendo la integridad del ambiente de ejecución de la aplicación.

3. Convenientes pero peligrosas características del diseño del software

Esta fuente de vulnerabilidad nos presenta funcionalidades que son deseables en el software para aumentar la versatilidad de uso de las aplicaciones. Entre estas tenemos herramientas de depuración o debugging, conexiones remotas en puertos especiales, entre otras, las cuales ofrecen importantes elementos a los programadores y usuarios, pero que generalmente abren posibilidades de ingresos no autorizados que comprometen la integridad de sistemas y socavan la confianza del usuario frente a la aplicación.

4. Invocaciones no controladas

En esta parte hacemos referencia a un inadecuado manejo de errores o excepciones en las aplicaciones o exceso de privilegios de ejecución, los cuales se manifiestan en comportamientos inesperados del software que generalmente ofrecen mayores privilegios o accesos adicionales a la información del sistema. En este sentido, el control adecuado de interrupciones, mensajes de error y entorno de ejecución de los programas se vuelve crítico al ser estos elementos los que definen la interacción del software con el usuario final y su relación con el entorno de ejecución.

5. Bypass a bajo nivel

Las implicaciones de esta fuente de vulnerabilidades hacen referencia al aseguramiento que la aplicación debe tener al ser invocada o ejecutada en un ambiente computacional seguro. El programador debe fortalecer y asegurar una manera autorizada de ingreso a la aplicación por parte del usuario, estableciendo mecanismos de monitoreo y control que velen por que esto se cumpla. El sobrepasar un control de acceso a un objeto, bien sea a través de permisos deficientemente otorgados, artificios que interrumpan la normal ejecución (contraseñas de BIOS) o por la manipulación de la memoria de ejecución de la aplicación constituye un atentado directo contra la confiabilidad e integridad del software.

6. Fallas en la implementación de protocolos

Los elementos de seguridad mencionados en este apartado, hacen referencia a las medidas de seguridad en redes. Si bien, los protocolos utilizados para transmisión y control de datos presentan múltiples fallas, éstas con frecuencia no son consideradas dentro del proceso de implementación de una aplicación. En este sentido, sabemos que las aplicaciones que se ejecutan sobre TCP/IP tienen inherentes las fallas de este conjunto de protocolos, por tanto es menester del programador establecer junto con el encargado de la seguridad informática los posibles requerimientos de seguridad necesarios para que la aplicación funcione sobre un ambiente de red que brinde mayores niveles de seguridad y control de tráfico.

7. Fallas en software de base

Todas las aplicaciones finalmente se ejecutan bajo la supervisión de un software de base o sistema operacional. Generalmente cuando se desarrollan aplicaciones, las condiciones o aseguramiento del software de base no es condición suficiente para la adecuada ejecución de aplicaciones. Nada se gana con aplicar y efectuar un amplio espectro de pruebas y controles, cuando el ambiente de ejecución o el software base no ha pasado por una valoración y afinamiento necesarios para asegurar un ambiente de ejecución estable y seguro. En este punto, se llama la atención tanto a proveedores como a programadores, donde el trabajo conjunto debe ser una constante para incrementar los niveles de seguridad y disminuir las vulnerabilidades frecuentes inherentes al arte y la ciencia de programar. [3]

1.5 – Controles de seguridad

Un entorno operativo seguro exige la garantía de que sólo puedan acceder a una determinada información aquellos que están autorizados para ello, que la información se procese correctamente y que esté disponible cuando se necesita.

Para aplicar controles adecuados, es preciso comprender primero quién o qué es lo que amenaza dicho entorno, así como conocer los riesgos asociados a dichas amenazas si llegan a materializarse.

1.5.1 - Amenazas

Se entiende por amenaza una condición del entorno del sistema de información (persona, máquina, suceso o idea) que, dada una oportunidad, podría dar lugar a que se produjese una violación de la seguridad (confidencialidad, integridad, disponibilidad o uso legítimo). [7]

La información se ve sometida a distintas amenazas que pueden clasificarse en intencionadas, no intencionadas y naturales.

Las amenazas intencionadas las ejercen usuarios no autorizados que acceden de forma indebida a los datos o información sensible. Los usuarios no autorizados pueden ser externos o pertenecientes a la propia organización y se pueden clasificar como curiosos o maliciosos. Los curiosos normalmente ojean un poco y no siempre entran con unas pretensiones concretas, ni saben lo que van a encontrar. Los maliciosos entran a los sistemas para apropiarse de datos o información por intereses económicos, o bien con ánimo de dañar o destruir recursos. El acceso no autorizado de usuarios ya sean curiosos o maliciosos significa siempre una violación de la confidencialidad y con frecuencia acarrea violaciones de la integridad y de la disponibilidad.

Las amenazas no intencionadas provienen típicamente de empleados con poca formación o negligentes que no han seguido los pasos para proteger sus contraseñas, asegurar adecuadamente sus ordenadores o actualizar con la frecuencia debida el programa antivirus. Las amenazas no intencionadas también implican a veces a los programadores o personal de procesamiento de datos cuando no se siguen las normas y procedimientos de seguridad establecidos, cuando existen. Este entorno operativo es especialmente sensible ya que sencillos errores en un programa pueden afectar a la integridad de la aplicación global y de cualquier otra aplicación con la que comparta información en común.

Las amenazas naturales incluyen fallos de equipos y calamidades tales como incendios, inundaciones y terremotos que pueden causar la pérdida de equipos y datos. Las amenazas naturales suelen afectar la disponibilidad de los recursos y de la información.

1.5.2 – Riesgos

Los riesgos asociados a la pérdida de la confidencialidad, integridad o disponibilidad son de diverso tipo y casi siempre implican daños económicos incluyendo responsabilidad contractual, falsos datos financieros, mayores costes, pérdidas de activos, pérdida de negocios, descrédito y pérdida de la imagen pública.

Amenaza	Problema	Riesgo
Usuario no autorizado	Confidencialidad	Acceso no autorizado
Curioso	Integridad	Divulgación no autorizada
Observación no autorizada	Integridad	Divulgación no autorizada
Monitorización no autorizada	Integridad, Confidencialidad	Pérdida de la información
Sustracción de información	Integridad, Confidencialidad	Pérdida de la información
Copias no autorizadas	Integridad, Confidencialidad	Pérdida de la información
Usuario no autorizado	Confidencialidad(Acceso indebido)	Copia o sustracción de datos
Malicioso	Integridad	Alteración de datos, falsificación, fraude.
Usuario autorizado	Disponibilidad	Pérdida o alteración de información
Catástrofe Natural, Destrucción	Disponibilidad	Daños, averías, pérdida de servicio

1.5.3 – Salvaguardas

Estudios realizados en los últimos años demuestran que un alto porcentaje de organizaciones han experimentado algún tipo de pérdida de información o soporte físico, siendo las causas más frecuentes los errores no intencionados y los virus. Hoy día, debiera ser impensable tener PCs o servidores sin programas de protección contra virus. El aumento en el número de delitos informáticos está haciendo replantearse a los profesionales de las tecnologías de la información las medidas de seguridad y mecanismos de control. Los delitos informáticos se han convertido en un mayor riesgo debido por una parte a que hay un mayor número de personas que conocen la informática y tienen acceso a recursos informáticos, y por otra al alto nivel de interconexiones entre redes tanto internas como externas. Pero sigue ocurriendo que la mayoría de los incidentes se originan interiormente a la organización, tanto los no intencionados como los maliciosos.

Las salvaguardas no son uniformes para todos los sistemas. El nivel del riesgo debiera determinar el nivel de control adecuado. Cada riesgo se puede tratar mediante la aplicación de uno o varias salvaguardas de seguridad. Las salvaguardas se pueden clasificar en tres principales categorías:

- administrativas
- físicas
- técnicas

Las salvaguardas administrativas incluyen las políticas y procedimientos de seguridad. Las políticas establecen lo que los usuarios pueden y no pueden hacer al utilizar los recursos informáticos de la organización. También incluyen procedimientos para la renovación de claves de acceso, autorizaciones para acceder a sus recursos, la revisión y validación periódica de la vigencia del tipo de acceso, la asignación de responsabilidades, conocimientos de la seguridad y formación técnica, gestión y supervisión de las tecnologías y soluciones aplicadas, la recuperación tras averías o fallos y la realización y aplicación de planes de contingencia. Los controles administrativos también incluyen la revisión de seguridad e informes de auditoria, que se utilizan para identificar si los usuarios siguen las políticas y procedimientos. Finalmente, los controles administrativos incluyen la asignación de propiedad de los datos y los recursos. Cada persona de la organización debe tener claras sus responsabilidades relativas a la seguridad de cada componente a su cuidado, y estas responsabilidades se deben incluir en los objetivos de cada persona para asegurar que se les valore por el trabajo asignado y se les evalúe según el cumplimiento de sus obligaciones.

Las salvaguardas físicas limitan el acceso físico directo a los equipos. Las salvaguardas físicas consisten en cerraduras, bloqueos para teclados, vigilantes de seguridad, alarmas y sistemas ambientales para la detección de agua, fuego y humo. Las salvaguardas físicas también incluyen sistemas de respaldo y alimentación de reserva, tales como baterías y fuentes de alimentación ininterrumpida (UPS).

Las salvaguardas técnicas son controles que se implantan a través de soportes físicos o lógicos que típicamente son difíciles de vencer y, una vez implantados, pueden funcionar sin la intervención humana. El soporte lógico específico incluye antivirus, firmas digitales, cifrado, programas de control de biblioteca, herramientas de gestión de red, contraseñas, tarjetas inteligentes, control de acceso de llamadas, sistemas de re-llamada, seguimiento de huellas o trazas de auditoria y sistemas expertos de detección de intrusiones.

Los controles de acceso protegen contra la utilización o manipulación no autorizada de recursos mediante la verificación y la autorización. La verificación asegura la identidad del usuario o sistema que solicita acceso. Los controles de autorización aseguran que el acceso a la información y los recursos informáticos se limiten de acuerdo con las directrices de la dirección. Otro término relacionado a la seguridad es el no-repudio. El no-repudio previene la posibilidad de que una de las partes de un intercambio o transacción niegue en falso después de que haya tenido lugar la misma. El control técnico que se emplea para garantizar el no-repudio es la firma digital. La firma digital consiste en una clave privada que sólo conoce o puede duplicar el tenedor de la clave, parecida a su firma y rúbrica manuscrita, las firmas digitales verifican la fuente, autenticidad e integridad de mensajes electrónicos.

Las salvaguardas administrativas, físicas y técnicas se pueden subdividir en preventivas y correctivas. Las preventivas intentan evitar la ocurrencia de acontecimientos indeseados, mientras que las salvaguardas correctivas intentan identificar los incidentes y reducir sus efectos después de que hayan sucedido. [6]

1.6 – Políticas de seguridad

El término política de seguridad se suele definir como el conjunto de requisitos definidos por los responsables directos o indirectos de un sistema que indica en términos generales qué está y qué no está permitido en el área de seguridad durante la operación general de dicho sistema.

Objetivos de las políticas de seguridad:

- Informar al mayor nivel de detalle a los usuarios, empleados y gerentes de las normas y mecanismos que deben cumplir y utilizar para proteger los componentes de los sistemas de la organización.

Cualquier política ha de contemplar seis elementos claves en la seguridad de un sistema informático:

- Disponibilidad: Es necesario garantizar que los recursos del sistema se encontrarán disponibles cuando se necesitan, especialmente la información crítica.

- Utilidad: Los recursos del sistema y la información manejada ha de ser útil para alguna función.

- Integridad: La información del sistema ha de estar disponible tal y como se almacenó por un agente autorizado.

- Autenticidad: El sistema ha de ser capaz de verificar la identidad de sus usuarios, y los usuarios la del sistema.

- Confidencialidad: La información sólo ha de estar disponible para agentes autorizados, especialmente su propietario.

- Posesión: Los propietarios de un sistema han de ser capaces de controlarlo en todo momento; perder este control en favor de un usuario malicioso compromete la seguridad del sistema hacia el resto de los usuarios. [8]

1.7 – Normas básicas de seguridad

Existen diferentes normas básicas de seguridad que todo arquitecto de software, desarrollador, fabricante y profesional de la seguridad involucrado en el diseño, desarrollo, despliegue y verificación de las aplicaciones y servicios Web debe tener muy en cuenta.

Las normas son las siguientes:

- Validación de la entrada y salida de información: La entrada y salida de información es el principal mecanismo del que dispone un atacante para enviar o recibir código malicioso contra el sistema. Por tanto, siempre debe verificarse que cualquier dato entrante o saliente es apropiado y en el formato que se espera. Las características de estos datos deben estar predefinidas y debe verificarse en todas las ocasiones.

- Diseños simples: Los mecanismos de seguridad deben diseñarse para que sean los más sencillos posibles, huyendo de sofisticaciones que compliquen excesivamente la vida a los usuarios. Si los pasos necesarios para proteger de forma adecuada una función o módulo son muy complejos, la probabilidad de que estos pasos no se ejecuten de forma adecuada es muy elevada.

- Utilización y reutilización de componentes de confianza: Debe evitarse reinventar la rueda constantemente. Por tanto, cuando exista un componente que resuelva un problema de forma correcta, lo más inteligente es utilizarlo.

- Defensa en profundidad: Nunca confiar en que un componente realizará su función de forma permanente y ante cualquier situación. Hemos de disponer de los mecanismos de seguridad suficientes para que cuando un componente del sistema falle ante un determinado evento, otros sean capaces de detectarlo.

Por tanto, no debemos fiarnos únicamente de los mecanismos de seguridad "exteriores", sino que es preciso identificar cuáles son los puntos precisos en los que deben establecerse las medidas de seguridad.

- Verificación de privilegios: Los sistemas deben diseñarse para que funcionen con los menos privilegios posibles. Igualmente, es importante que los procesos únicamente dispongan de los privilegios necesarios para desarrollar su función, de forma que queden compartimentados.

- Ofrecer la mínima información: Ante una situación de error o una validación negativa, los mecanismos de seguridad deben diseñarse para que faciliten la mínima información posible. De la misma forma, estos mecanismos deben estar diseñados para que una vez denegada una operación, cualquier operación posterior sea igualmente denegada. [9]

1.8 - Ingeniería de sistemas de Computación

Muchas preguntas con respecto a la seguridad, son relacionadas con el ciclo vital de software. En particular, la seguridad del código y el proceso de software deben de ser considerados durante la fase del diseño y desarrollo. Además, la seguridad debe de ser preservada durante la operación y el mantenimiento para asegurar la integridad de una porción de software.

Pueden engañarle en la creencia de que su trabajo como diseñador de sistemas de seguridad ya realizado, es lo suficientemente seguro, pero sin embargo, las cadenas y computadoras son increíblemente inseguras. La falta de seguridad se origina en dos problemas fundamentales: los sistemas que son teóricamente seguros pueden ser inseguros en la práctica, además los sistemas son cada vez más complejos y la complejidad proporciona más oportunidades para los ataques. Es mucho más fácil probar que un sistema es inseguro que demostrar que uno es seguro -probar la inseguridad, simplemente se toma ventaja de ciertas vulnerabilidades del sistema. Por otra parte, probando un sistema seguro, requiere demostrar que todas las hazañas posibles puedan ser defendidas contra los diferentes ataques existentes.

Actualmente, no hay ninguna solución singular para asegurar completamente la ingeniería de software. Sin embargo, hay métodos específicos que mejoran la seguridad de los sistemas. En particular, podemos mejorar la confiabilidad del software. [10]

1.9 - Confiabilidad de software

La confiabilidad del software significa que un programa particular debe seguir funcionando con la presencia de errores. Los errores pueden ser relacionados al diseño, a la implementación, a la programación. Así como los sistemas llegan a ser cada vez más complejos, aumenta la probabilidad de errores. Como se mencionó, es increíblemente difícil demostrar que un sistema sea seguro. Aunque casi todos los software tengan errores, la mayoría de los errores nunca serán revelados debajo de circunstancias normales. Un atacante busca esta debilidad para atacar un sistema.

Muchos de los problemas de la seguridad hoy en día, son relacionados con el código defectuoso. Por ejemplo, el Morris Internet Worm (el gusano Internet de Morris) utilizó desbordamiento en un programa de UNIX para ganar acceso a las computadoras que ejecutaron el programa. Los ataques de desbordamiento de buffer han sido el tipo de ataque más común en los últimos diez años y consisten en sobre grabar instrucciones en el programa. Específicamente, una cantidad fija de memoria en la pila, puede ser reservada por el usuario; si la entrada de información del utilizador es más grande que este espacio reservado, el usuario puede sobre grabar las instrucciones del programa. Si esto se hace cuidadosamente, el usuario puede insertar sus propias instrucciones en el código del programa, así la máquina receptora realizará operaciones arbitrarias dictadas por el atacante. Mientras que tales ataques se pueden prevenir típicamente con bounds checking (revisando el tamaño de la entrada de información antes de copiarla), ésta es una cuestión de práctica de programación que confiamos en que el programador mismo seguirá. El aspecto difícil de desbordamiento de buffer es que puede ocurrir en una gran cantidad de lugares en cualquier programa, y es difícil de prevenir el suceso por todas partes. Este ha sido el caso en el pasado, especialmente, en los últimos 10 años. [10]

1.10 – Análisis de las metodologías de programación segura

1.10.1 - Metodología de Desarrollo Web Seguro Basado en Formación a Medida

Esta metodología está basada en la integración de cuatro actividades: análisis del entorno de desarrollo (AED), realización de pruebas de intrusión (RPI) sobre los servicios Web, sesiones de concienciación (SC) y formación especializada (FE) a cada uno de los perfiles del equipo de desarrollo.

Análisis del entorno de desarrollo (AED): En esta fase se estudian diferentes aspectos del enfoque de programación utilizado por el equipo de desarrollo Web de la organización. De esta forma se consigue una fotografía del estado actual que permite adecuar los nuevos procedimientos de desarrollo seguro a la metodología existente. Algunos de los aspectos que se contemplan en esta fase son los siguientes: tecnologías de desarrollo, herramientas de programación, arquitectura de los servicios Web, procedimientos de integración, organización del equipo de trabajo, matriz de responsabilidad sobre las tareas.

Realización de pruebas de intrusión (RPI) sobre los servicios Web: La revisión técnica preliminar se basa en la ejecución de pruebas de intrusión para detectar las vulnerabilidades o deficiencias que existen actualmente en los servicios Web de la organización.

Sesiones de concienciación (SC): En estas sesiones el equipo de desarrollo conoce las posibilidades que brinda a un atacante el empleo de metodologías de programación que no contemplan cuestiones de seguridad. Se presentan las evidencias recogidas durante la revisión técnica preliminar para contextualizar problemas genéricos de seguridad en el ámbito de los servicios Web de la propia organización. El objetivo es concienciar a los miembros del equipo de desarrollo acerca de la necesidad de cambiar algunos aspectos de la metodología actual.

Formación especializada (FE): El objetivo es formar a cada uno de los perfiles involucrados en la creación de servicios Web, indicando qué medidas pueden incorporar al desarrollo de su actividad

para garantizar la adecuación de los resultados a los requisitos de seguridad que exigen dichos servicios.

Aplicación de la nueva metodología: En esta fase se pone en marcha la nueva metodología de desarrollo. La guía de programación segura se convierte en el vehículo de la actividad desarrollada por cada uno de los perfiles del equipo, garantizado que en cada una de las tareas se emplean los procedimientos adecuados.

Evaluación de resultados: Las sucesivas revisiones técnicas se utilizan como mecanismo de evaluación del grado de aceptación que la nueva metodología está teniendo en el equipo de desarrollo. El grado de aceptación se incrementa a medida que se corrigen las deficiencias encontradas en cada fase del proceso de revisión técnica. Para llevar a cabo dicha corrección se identifican a los responsables de cada deficiencia con el objetivo de incidir en los aspectos de las sesiones formativas que no han sido tenidos en cuenta durante el desarrollo del servicio Web.

Esta etapa se da por finalizada cuando no es posible identificar nuevos procedimientos de intrusión para comprometer los servicios Web de la organización. Obviamente, en este punto se habrán alcanzado los objetivos de seguridad fijados al comienzo del proyecto.

La puesta en marcha de un plan de formación personalizado puede garantizar el cumplimiento de los requisitos de seguridad en los servicios Web de una organización. El know-how adquirido por los responsables de la creación de dichos servicios puede emplearse en futuros desarrollos y permite abordar la problemática de la seguridad desde un punto de vista más general, creando una base de conocimiento que elimina de forma proactiva los riesgos asociados a la exposición de servicios en redes públicas. [28]

Esta metodología depende en gran medida de la formación personalizada que se lleve a cabo al inicio de la construcción del proyecto aplicada a cada uno de los responsables de seguridad; por lo que su eficiencia se torna incontrolable en cierta medida, ya que la misma dependería de la capacidad de adquisición de los conocimientos impartidos, provocando la escasa confiabilidad en la aplicación.

1.10.2 - Plan de Seguridad Integral de los Sistemas de Información de la Diputación Foral de Gipuzkoa

La metodología surge por:

- Vulnerabilidades que se descubren constantemente.
- Acciones de los hackers
- La toma aislada de iniciativas
- Diferentes criterios de seguridad.

La metodología esta dividida en 4 fases:

- Fase I: Conocimiento del entorno.
    - Conocimiento del entorno informático.
    - Identificación de aplicaciones.
    - Identificación de la seguridad de las aplicaciones.

- Fase II: Marco de referencia
    - Ámbito de actuación
    - Amenazas y buenas practicas (ISO 17799)
    - Definición de dominios de seguridad

- Fase III: Modelo de seguridad
    - Política de seguridad
    - Organización de seguridad
    - Cuerpo normativo de primer nivel del sistema de gestión documental de seguridad
    - Definición de controles de seguridad en base a la ISO/IEC 17799
    - Por cada control: Niveles de seguridad desarrollados en base al ISO 17799 y objetivos de cumplimiento.
    - Revisión de seguridad y diagnostico de seguridad interna/externa

- Fase IV: Plan de acción
    - Análisis de las referencias
    - Informe de riesgos
    - Informe de proyectos
    - Plan de implantación

Alcance del Plan:

Áreas que se consideraron:

- Política de Seguridad
- Estructura Organizativa para la Seguridad
- Clasificación y Control de Activos
- Seguridad frente a Acciones Humanas
- Seguridad Física y del Entorno
- Gestión de Comunicaciones y Operaciones
- Control de Accesos
- Desarrollo y Mantenimiento de Sistemas
- Gestión de Continuidad de Negocio
- Conformidad legal [29]

Esta metodología tiene en cuenta, en 4 fases, los principales requisitos de seguridad que se deben considerar en la realización de un proyecto software, además de hacer revisiones periódicas. La deficiencia de la misma consiste en que las observaciones y controles que se hacen no tienen una delimitación exacta de las etapas en las que se debe probar qué parte del producto se está controlando. Aunque se debe reconocer que hace una exhaustiva verificación de la seguridad en las diferentes fases que plantea la metodología.

1.10.3 – SPSMM (Secure Programming Standards Methodology Manual): Standard de Programación Segura

El objetivo de esta metodología es realizar un Standard de programación segura que pueda ser usado en cualquier proceso, manual o automático, y permita alcanzar los requerimientos de seguridad para maximizar el uso y evitar su abuso. El resultado indirecto es la creación de una disciplina que pueda actuar como un punto central en todas las pruebas de seguridad independientemente del lenguaje de programación, ambiente de ejecución y herramientas de desarrollo.

Esta metodología está enfocada a determinar dónde está ubicado el error de seguridad en el código, cómo arreglarlo y evitar su aparición en futuras codificaciones. [30]

Esta metodología es ineficiente puesto que se centra en la fase de implementación principalmente y el objetivo de este trabajo es la necesidad de controlar la seguridad desde el inicio de la construcción del proyecto, no sólo en su fase de implementación, sino en todas las que le preceden y suceden.

1.10.4 - Metodología para la programación segura (OSSTMM)

La metodología para la programación segura es un suplemento de la metodología de seguridad OSSTMM (Open Source Security Testing Methodology Manual) que facilita una serie de estándares en vistas al desarrollo de código que deberá encontrarse accesible en Internet. Por tanto se parte desde un principio con la idea de que este código debe estar preparado para sobrevivir en un entorno altamente hostil.

Estos conceptos se plantean de forma universal para que puedan ser aplicados a los diversos procesos de desarrollo, ya sean manuales o automáticos y el objetivo es poder alcanzar unos requisitos de seguridad que permitan conjugar la máxima productividad del código evitando cualquier posible mal uso que pueda provocar agujeros en la seguridad de los sistemas o aplicaciones.

Los aspectos sobre los que se hace énfasis son:

- Comprobación de los datos entrantes
- Protección del proceso
- Control de los datos de salida

Por otra parte se realiza un análisis de los diversos errores de programación que pueden provocar agujeros en la seguridad: desbordamiento de memoria intermedia ("buffer overflow"), ausencia de control en las cadenas de caracteres, el compromiso remoto y la retención de recursos. [12]

Esta metodología al igual que la *SPSMM* está enfocada a estándares de programación segura para controlar la seguridad de la aplicación básicamente en la fase de implementación, lo cual representa una debilidad del sistema ya que se pueden cometer errores anteriores a esta fase que impliquen en fallas que provoquen la inestabilidad en la seguridad de la aplicación.

1.10.5 - Desarrollo de productos de software seguros en sintonía con los modelos SSE-CMM, COBIT E ITIL

Esta metodología analiza los modelos SSE-CMM, COBIT E ITIL para determinar cómo se enlazan y reafirman las propuestas existentes para el desarrollo de productos de software seguro.

Systems Security Engineering Capability Maturity Model (SEI)

Es un modelo de referencia para la incorporación de la Ingeniería de Seguridad en las organizaciones. Dentro del alcance de SSE-CMM, se encuentra la presentación de una serie de actividades para lograr el desarrollo de productos de software confiables, y alcanzar un ciclo de vida para sistemas seguros.

SSE-CMM divide la ingeniería de seguridad en tres áreas básicas: riesgo, ingeniería y aseguramiento.

- Riesgo, busca identificar y priorizar los peligros asociados al desarrollo de productos o sistemas.

- Ingeniería, trabaja con otras disciplinas para implementar soluciones a los peligros identificados, en este caso, se relaciona con la Ingeniería de Software.

- Aseguramiento, tiene como objetivo certificar que las soluciones implementadas son confiables.

El modelo se estructura en dos dimensiones: Dominios y Capacidades. Un dominio es un conjunto de prácticas básicas que definen la ingeniería de seguridad, mientras que una capacidad se refiere a las prácticas genéricas que determinan la administración del proceso e institucionalizan la capacidad.

Control Objectives for Information and related Technology (COBIT)

El objetivo de este framework es organizar y armonizar distintos estándares internacionales, relacionados con la administración de la Tecnología de la Información (TI) en las organizaciones. Presenta un conjunto de mejores prácticas, enfocadas en el control más que en la ejecución, que permiten optimizar la inversión en TI que una organización realiza.

El modelo define un conjunto de criterios de control, en base a requisitos de calidad, confianza y seguridad. Se organiza en función de cuatro dominios, los que a su vez se dividen en procesos formados de actividades específicas, que definen los objetivos de control que una organización debería implementar.

Information Technology Infrastructure Library (ITIL)

Ofrece un marco común para todas las actividades de la organización, como parte de la provisión de servicios, basado en la infraestructura tecnológica. Estas actividades se dividen en procesos, que proporcionan un marco eficaz para lograr una Gestión de Servicios de Tecnologías de la Información más madura. Proporciona una descripción detallada de una serie de buenas prácticas, a través de una amplia lista de roles, tareas, procedimientos y responsabilidades que pueden adaptarse a cualquier organización de TI. [27]

Los tres modelos anteriormente presentados, incorporan la seguridad desde distintas perspectivas, algunas de las cuales son coincidentes y otras complementarias. A partir de estas singularidades y similitudes se puede determinar una definición multidimensional de la seguridad en relación al desarrollo de software.

Los modelos *COBIT y SSE-CMM* definen un conjunto de prácticas que deberían tenerse en consideración.

Tanto *SSE-CMM como COBIT* plantean la necesidad que existan procesos de desarrollo de software que estén claramente definidos.

Los modelos *COBIT e ITIL*, aportan la perspectiva de cómo administrar los productos correspondientes a TI.

La dimensión de Factores Humanos, se ve beneficiada por el modelo *SSE-CMM*, donde se presentan una serie de procesos que deberían ser desarrollados en las organizaciones para incorporar la seguridad, justificando la necesidad de que existan Ingenieros de Seguridad, que trabajen en conjunto a los otros dominios de la Ingeniería, en especial con los Ingenieros de Sistemas e Ingenieros de Software.

Esta metodología integra tres modelos que se refieren a la seguridad desde diferentes puntos de vista, al igual que SDL lleva a cabo la seguridad en el ciclo de vida del software proponiendo la necesidad de un equipo de seguridad, pero sin hacer hincapié en las diferentes fases por la que este transcurre durante su desarrollo.

1.10.6 - SDL ("Security Development Lifecycle")

SDL es un proceso que se utiliza para desarrollar software que pueda resistir ataques malintencionados. Incorpora varias actividades y materiales relacionados con la seguridad a cada una de las fases del proceso de desarrollo de software que incluyen el desarrollo de modelos de amenazas durante el diseño de software, el uso de herramientas de exploración del código de análisis estático durante la implementación y la realización de revisiones del código y pruebas de seguridad durante una "campaña de seguridad". Antes del lanzamiento de software sometido al SDL, un equipo independiente del grupo de desarrollo debe realizar una revisión final de seguridad. Con esto se logra un alto nivel de confiabilidad de la aplicación en cualquiera de las fases de su desarrollo.

1.10.7 - Otras ideas sobre la programación segura

Los desbordamientos de buffer son un problema relacionado con código incorrecto.

Existen dos posibles estrategias:

- Abordar el problema utilizando una metodología de programación que elimine los errores de programación. Generar código correcto.

- Utilizar mecanismos de protección para que en tiempo de ejecución detecten los posibles fallos e impidan crear exploits. Impedir que código erróneo se convierta en fallos de seguridad.

Consejos de programación

Una breve lista de cuestiones a tener presentes siempre que se programa:

- Utiliza lenguajes de programación que permitan verificar rangos de vectores en tiempo de compilación: Java, ADA, etc.

- Valida TODOS los datos que introduce el usuario: valores leídos de ficheros, nombres de ficheros, datos de teclado, direcciones URL, mensajes de red, variables de entorno, etc.

- No utilices funciones de manipulación de cadenas que no comprueben la longitud de la cadena destino. sprintf, fscanf, scanf, sscanf, vsprintf, gets, strcpy, strcat, etc.

- Verifica todos los valores de error devueltos por las llamadas a sistema y la librería.

El problema del desbordamiento de buffer es un excelente argumento para usar otros lenguajes de programación en vez de C y C++ como Perl, Python, Java y Ada95. Después de todo, casi todos los lenguajes de programación que se usan hoy día (excepto lenguaje ensamblador) protegen de esta amenaza; aunque usar estos otros lenguajes no elimina, por supuesto, todos los problemas de programación. [31]

1.11 - ¿Por qué optar por SDL “Security Development Lifecycle”?

La necesidad es la madre de la creación. Este lema resume el porqué del nacimiento de SDL. Un proceso riguroso que hace de la seguridad un foco crítico para cada línea de código.

Las mejores defensas que se pueden tener para enfrentar los ataques son: código limpio, software seguro por defecto y productos de seguridad que bloqueen los ataques o se recuperen de ellos automáticamente.

Los objetivos de este proceso de control de la seguridad en el desarrollo de software son:

- Reducir el número de problemas de privacidad y vulnerabilidades de seguridad del producto que se desarrolla

- Reducir el grado de las vulnerabilidades que queden después de haber cumplido el primer objetivo

No se pueden eliminar todas las vulnerabilidades de seguridad y privacidad de un sistema, porque cuando un producto se hace, es construido basado en las mejores prácticas de seguridad de cada día; pero las investigaciones y descubrimientos sobre ataques nunca terminan. SDL no es estático. Ningún proceso de desarrollo de software que se centre en la seguridad podría serlo porque ésta evoluciona a alta velocidad. El proceso de cambio o actualización está diseñado para incorporar solo los requisitos que constituyan a una mejora para la seguridad del software. Este proceso lo inicia el usuario como una recomendación o un requerimiento para futuros productos.

Dado el gran nivel de interconexión mundial que existe se ha incrementado el nivel de exposición a amenazas, y esto a su vez ha producido un aumento del riesgo de seguridad que sufren todos los usuarios que navegan por la red en busca de información. El crimen informático aparece y se hace más peligroso pues el “criminal” puede acceder a información privada o sensible o usar un sistema comprometido para posteriores ataques a otros usuarios.

El proceso de control de la seguridad en el desarrollo del software se debe aplicar a los productos que se ajusten a los siguientes modelos:

- Cualquier producto que se use continuamente en un negocio: Ej. Correo electrónico y bases de datos

- Cualquier producto que almacene y procese información personal de los usuarios

- Cualquier producto que se conecte regularmente a Internet. Ej. Software de mensajería instantánea, buscadores Web, clientes de correos electrónicos o juegos con opción multi-jugador en línea

Los sistemas operativos también están en la categoría de los software que debemos asegurar ya que deben proteger la confidencialidad e integridad de la información del usuario, la cual debe estar disponible incluso en casos de ataques.

El riesgo de cualquier producto software depende de cuán expuesto esté a las amenazas potenciales y del valor de la información que se utilice. Debemos considerar también el efecto que tendría si la información estuviera a disposición de todos los usuarios, si se modificara o se destruye. Incluso los softwares que puedan no ser calificados para aplicarle SDL deberían ser revisados para asegurarse de que no hay forma de que comprometa o exponga la plataforma en la que corre.

Ej. Instalar puentes de usuarios sin protección o archivos ejecutables que puedan modificarse. [26]

1.12 – ¿Quiénes utilizan SDL (Security Development Lifecycle)?

Desarrollado en Microsoft, SDL es un acercamiento que los desarrolladores de aplicaciones de toda la industria pueden utilizar para mejorar la seguridad y calidad del software que ellos crean: hoy Microsoft esta trabajando con un número de compañías de software ayudándoles a revisar sus procesos de desarrollo y así pueden tomar ventaja del SDL. [17]

Conclusiones Parciales

Se ha podido apreciar que la seguridad del software busca que un producto desarrollado continúe funcionando correctamente ante ataques maliciosos y pueda ser vista como una medida de robustez de un sistema de software.

La seguridad es un tema que todos deben tener en cuenta a la hora de desarrollar un software, ya que la misma es muy cambiante, día a día surgen nuevos métodos y metodologías relacionadas con la misma, pero paralelo al surgimiento de estos métodos y metodologías se crean los diferentes ataques que tienen como único pretexto el colapso del software.

Capítulo II: “Etapas de desarrollo del software”

Introducción

En el capítulo anterior se enunciaron los principales conceptos sobre seguridad informática, su importancia, las diferentes fuentes de vulnerabilidad que puede tener el software, las normas de seguridad, las diferentes metodologías estudiadas, entre otros aspectos importantes referentes a la seguridad. Se comenzó a abordar el tema del porque usar SDL “Security Development Lifecycle” y cuáles son los principales software que requieren de su uso.

A partir de lo planteado en el primer capítulo y haciendo énfasis en el campo de acción de este trabajo, este capítulo estará referido a las fases por las que transcurre un software durante todo su desarrollo.

Para tener conocimiento de los diferentes criterios relacionados con la aplicación de la seguridad en las distintas fases de desarrollo del software se aplicó una encuesta (Anexo 1) principalmente a líderes de proyectos de la facultad 2. De un total de 11 proyectos de la facultad mencionada fueron encuestados 8 de ellos, representando un 72.7%. En este capítulo se verán también los resultados de dicha encuesta.

2.1- Fases

Una fase describe todas las actividades que hay que realizar para obtener un conjunto concreto de productos de desarrollo del software.

Cada fase se describe a partir de los siguientes conceptos:

- Introducción: Objetivo de la etapa y relaciones con otras etapas

- Flujos de trabajo: Una agrupación de actividades que se realizan juntas. Además, se identifican los productos de desarrollo del software que se obtienen de cada actividad

- Actividades: La definición de todas las actividades que se llevan a cabo en la etapa

- Productos de desarrollo del software: Se describen todos los productos de desarrollo del software que se obtienen en la etapa. [3]

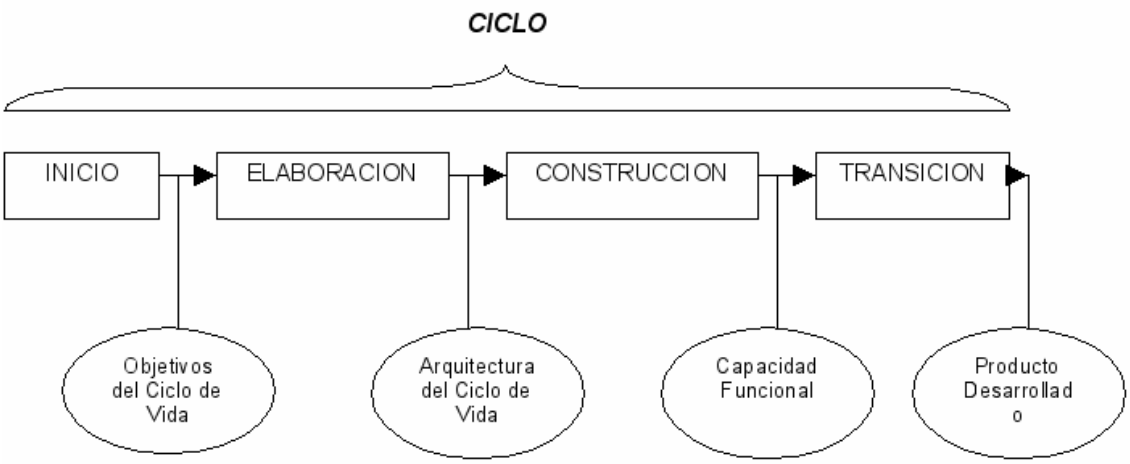

Figura 1: Fases del desarrollo del software

2.1.1 - Fase de Inicio

Esta fase se centra sobre todo en la comprensión de los requisitos en conjunto y en la determinación del ámbito del esfuerzo de desarrollo.

Se establece la visión del producto final y su proceso de negocio, así como la definición del ámbito del proyecto. A medida que se va iterando dentro de esta fase, se identifican las entidades externas con las que se trata (actores) y se define la interacción a un nivel de abstracción muy alto, identificando todos los casos de uso y describiendo algunos con más detalle (los que se consideren más importantes). Se diseñan las arquitecturas candidatas y se estima la agenda y el presupuesto de todo el proyecto, en particular para la siguiente fase de elaboración. Típicamente es una fase breve que puede durar unos pocos días o unas pocas semanas.

Esta fase culmina con la obtención de los objetivos del ciclo de vida, donde las partes interesadas en el desarrollo deben acordar el alcance y la estimación de tiempo y costo necesario para su realización, así como comprender los requerimientos plasmados en los casos de uso.

2.1.2 - Fase de elaboración

Los objetivos básicos de esta fase son la obtención de requerimientos: se analiza el dominio del problema, se establece una arquitectura base sólida, se desarrolla un plan de proyecto e se intentan eliminar los elementos de mayor riesgo para el desarrollo exitoso del proyecto.

Se trata de planificar las actividades necesarias y los recursos requeridos, especificando las características y el diseño de la arquitectura del software.

Al final de la fase se realiza un análisis para determinar los riesgos y se evalúan los gastos hechos contra los originalmente planeados. Esta fase culmina con la arquitectura del ciclo de vida.

2.1.3 - Fase de construcción

El objetivo básico es pasar del prototipo inicial al primer producto operativo; desarrollar el producto y evolucionar la visión, la arquitectura y los planes hasta que esté lista una primera versión del producto para ser enviado a los usuarios.

En esta fase todas las componentes restantes se desarrollan e incorporan al producto. Todo es probado en profundidad. El énfasis está en la producción eficiente y no ya en la creación intelectual. Puede hacerse construcción en paralelo, pero esto exige una planificación detallada y una arquitectura muy estable.

Esta fase culmina con la capacidad funcional inicial. Se obtiene una versión beta del producto que debe decidirse si puede ponerse en ejecución sin mayores riesgos.

2.1.4 - Fase de transición

El objetivo básico es realizar la transición del producto a los usuarios, lo cual incluye la fabricación, el envío, el entrenamiento (para conseguir autosuficiencia de los usuarios), el soporte y el mantenimiento del producto hasta que el cliente esté satisfecho. Se persigue el asegurar que el sistema tiene la calidad para alcanzar los objetivos.

Esta fase culmina con una versión operativa del producto, la cual a su vez concluye el ciclo.

Una vez instalada esta versión surgirán nuevos elementos que implicarán nuevos desarrollos (ciclos).

Estas fases y las iteraciones realizadas dentro de ellas no tienen por qué ser de la misma duración, sino que varían dependiendo de la naturaleza del proyecto y del momento en que se encuentra el proceso de desarrollo. Lo importante son los objetivos de cada fase y el suceso final de cada una de ellas. [14]

2.2 - Flujos de trabajo

2.2.1 - Modelado del Negocio

La necesidad de esta etapa surge ante el hecho de que muchos de los productos software que se desarrollan, automatizan algunos o todos los procesos existentes en un negocio y es necesario estudiar las implicaciones de los cambios producidos por la adopción de estos productos. Hay que entender cómo funciona el negocio que se desea automatizar para tener garantía de que el software desarrollado va a cumplir su propósito, y por esto, se hace un estudio en el dominio del negocio además del de dominio del software.

Así, los objetivos de la etapa de modelado del negocio son los siguientes:

- Entender los problemas actuales en la organización o empresa para identificar los aspectos a mejorar
- Comprender la estructura y el dinamismo de la organización o empresa para la cual se va a desarrollar el sistema software
- Estudiar el impacto que pueden producir los cambios a nivel organizativo
- Asegurar que los clientes, usuarios finales, desarrolladores y otros involucrados tienen una visión común de la organización considerada
- Obtener los requisitos del sistema software
- Entender cómo el sistema software encaja en la organización

Se obtendrán por tanto los objetivos del negocio u organización y se plasmarán las distintas visiones de éste bajo un modelo común.

Para conseguir estos objetivos el flujo de trabajo de la etapa de Modelado del Negocio consta de las siguientes etapas:

- Evaluar el estado del Negocio
- Análisis del Negocio
- Identificar Procesos de Negocio
- Definir y Refinar los Procesos de Negocio

- Diseño de la Realización de los Procesos de Negocio

- Evaluación

El flujo de trabajo de la etapa de Modelado del Negocio se desarrolla principalmente en la fase de inicio, donde se crea una primera versión del Modelo de Negocio que describe el contexto del sistema a construir. Durante la fase de elaboración se obtiene el modelo completo del negocio. Al final de esta fase se ha avanzado en el conocimiento del proyecto hasta el punto de haber estrechado notablemente los márgenes de error. De esta forma, se puede preparar una apuesta económica y desarrollar el análisis de negocio dentro de los márgenes muchos más estrechos de la práctica empresarial. Uno de los propósitos de la apuesta de negocio, desarrollada al final de la fase de elaboración, es el de servir de guía al jefe de proyecto y los inversores para ejecutar la fase de construcción. Con este objetivo, el jefe de proyecto comparará el progreso real al final de cada iteración con la agenda, esfuerzo y costes planificados. A medida que el jefe de proyecto adquiere un mayor conocimiento sobre los costes y capacidades del producto durante la fase de construcción, puede encontrar necesario actualizar el análisis de negocio y comunicar el nuevo análisis a los inversores. Al final de esta fase se modifica el modelo de negocio para reflejar la situación final de la fase.

2.2.2 - Requisitos

El proceso de desarrollo de software se inicia en la etapa de requisitos, en la cual se realiza la recolección de información sobre los requerimientos y objetivos del cliente, los usuarios, capacidades técnicas disponibles, sistemas existentes, etc.

Los objetivos de la etapa de requisitos son los siguientes:

- Establecer y mantener acuerdos con los clientes y otros implicados sobre lo que el sistema debería hacer y por qué

- Proporcionar a los desarrolladores del sistema una descripción mejor de los requisitos del sistema

- Definir los límites del sistema

- Proporcionar una base para planificar los contenidos técnicos de las iteraciones

- Proporcionar una base para estimar los costos y tiempo de desarrollo del sistema

Para conseguir estos objetivos, el flujo de trabajo de la etapa de Requisitos consta de las siguientes etapas:

- Analizar el problema

- Análisis de las necesidades de los implicados en el proceso de desarrollo

- Definir el sistema

- Gestionar el ámbito del sistema
- Evaluación

- Gestión de requisitos

El flujo de trabajo de la etapa de requisitos y sus productos de desarrollo del software adquieren diferentes formas durante las distintas fases y sus iteraciones:

- Durante la fase de inicio, los analistas identifican la mayoría de los casos de uso para delimitar el sistema y el alcance del proyecto y para detallar los más importantes

- Durante la fase de elaboración, los analistas capturan la mayoría de los requisitos restantes para que los desarrolladores puedan estimar el tamaño del esfuerzo de desarrollo que se requerirá. El objetivo es haber capturado sobre un 80 por ciento de los requisitos y haber descrito la mayoría de los casos de uso al final de esta fase de elaboración

- Los requisitos restantes se capturan e implementan durante la fase de construcción

- Casi no hay captura de requisitos en la fase de transición, a menos que haya requisitos que cambien.

2.2.3 - Análisis

Una vez recolectada toda la información necesaria se inicia la etapa de análisis de dicha información con el objeto de conceptualizar la estructura general del software, sus módulos e interacciones internas y externas, así como establecer prioridades entre todas las funciones requeridas, y estimar el tiempo y recursos necesarios para completar las diferentes partes. Al final de esta etapa se completa la planificación del proyecto, que incluye el cronograma de actividades, presupuesto, asignación de recursos y el acuerdo de las entregas a realizar.

En esta etapa se deben identificar las entradas del problema, los resultados deseados o salidas y cualquier requerimiento o restricción adicional en la solución:

- Identificar qué información se proporciona (datos del problema)

- Identificar qué resultados deben calcularse y/o desplegarse

- Determinar la forma y las unidades en que se deben desplegar los resultados

- Acotar las teorías, fundamentos y/o principios necesarios haciendo los supuestos y simplificaciones necesarias

- Identificar los tipos y estructuras de datos necesarios para los datos del problema y para los resultados

- Identificar las funciones u operaciones necesarias para cubrir los requerimientos del problema

2.2.4 - Diseño

En la etapa de diseño se define el modelo de datos a utilizar en el software y la interfaz de usuario para cada uno de los módulos del software. Una vez concebidos, se realiza el desarrollo del prototipo del software, el cual permite una visualización más clara de cómo se verá el software al ser completado. En esta etapa se finaliza la especificación funcional del software donde se detalla el alcance del proyecto. También se diseña el plan de pruebas del software y el criterio de aceptación.

El diseño consiste básicamente en desarrollar una lista de pasos llamados algoritmo o receta de la solución, verificando que el problema se resuelve como se desea. Se auxilia de técnicas de diseño como pseudocódigo y diagramas de flujo.

En concreto, los propósitos de la etapa de diseño son:

- Adquirir una comprensión en profundidad de los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y concurrencia, tecnologías de interfaz de usuario, tecnologías de gestión de transacciones, etc.

- Crear una entrada apropiada y un punto de partida para actividades de implementación subsiguientes capturando los requisitos o subsistemas individuales, interfaces y clases

- Ser capaces de descomponer los trabajos de implementación en partes más manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo, teniendo en cuenta la posible concurrencia

- Capturar las interfaces entre los subsistemas en el ciclo de vida del software. Esto ayuda cuando reflexionamos sobre la arquitectura y cuando utilizamos interfaces como elementos de sincronización entre diferentes equipos de desarrollo

- Ser capaces de visualizar y reflexionar sobre el diseño utilizando una notación común

- Crear una abstracción sin costuras de la implementación del sistema, en el sentido de que la implementación es un refinamiento directo del diseño que rellena lo existente sin cambiar la estructura. Esto permite la utilización de tecnologías como la generación de código y la ingeniería de ida y vuelta entre el diseño y la implementación

- Definir el modelo de interfaces de usuario

Dentro del ciclo de vida del software la etapa de diseño es el centro de atención al final de la fase de elaboración y el comienzo de las iteraciones de construcción. Esto contribuye a una arquitectura estable y sólida y a crear un plano del modelo de implementación. Más tarde, durante la fase de construcción cuando la arquitectura es estable y los requisitos están bien entendidos, el centro de atención se desplaza a la etapa de Implementación.

2.2.5 - Implementación

Esta etapa consiste en implementar o escribir el algoritmo como un programa de computadora en un lenguaje de programación, convirtiendo cada paso del algoritmo en instrucciones en el lenguaje de programación.

Se requiere como mínimo de las siguientes herramientas:

- Un editor de texto para escribir el código fuente como un archivo de tipo texto plano (por ejemplo notepad para guardar los archivos como .html)

- Un intérprete que procese el código fuente y lo ejecute (por ejemplo un browser que ejecuta scripts en javaScript al cargar la página Web)

- Un debugger que nos ayude a depurar los errores y a corregir el código fuente hasta lograr un programa ejecutable sin errores (por ejemplo el mismo browser que envía mensajes a encontrar errores al ejecutar nuestro programa)

Se deben utilizar los tipos y estructuras de datos más adecuados que permita el lenguaje de programación, teniendo especial cuidado en el uso de tipos de datos reales y los errores de redondeo que introducen y pueden alterar los resultados. [16]

El objetivo principal de la etapa de implementación es desarrollar la arquitectura y el sistema como un todo. De forma más específica, los propósitos de la Implementación son:

- Definir la organización del código

- Planificar las integraciones de sistema necesarias en cada iteración

- Implementar las clases y subsistemas encontrados durante el Diseño

Para conseguir estos objetivos el flujo de trabajo de la etapa de Implementación consta de las siguientes etapas:

- Estructurar el Modelo de Implementación

- Crear el Plan de Integración.

- Implementar componentes

- Validar componentes implementados

- Integrar subsistemas

- Validar Subsistemas implementados

- Integrar el Sistema Software

En el ciclo de vida del software la etapa de Implementación es el centro durante las iteraciones de construcción, aunque también se lleva a cabo el trabajo de implementación durante la fase de elaboración, para crear la línea base ejecutable de la arquitectura, y durante la fase de transición, para tratar defectos tardíos.

Ya que el modelo de implementación denota la implementación actual del sistema en términos de componentes y subsistemas de implementación, es natural mantener el modelo de implementación a lo largo de todo el ciclo de vida del software.

2.2.6 - Prueba

La prueba del software es un elemento crítico para la garantía de la calidad del software.

El objetivo de la etapa de prueba es garantizar la calidad del producto desarrollado.

Además, esta etapa implica:

- Verificar la interacción de componentes

- Verificar la integración adecuada de los componentes

- Verificar que todos los requisitos se han implementado correctamente

- Identificar y asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente

- Diseñar pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo

La prueba no es una actividad sencilla, no es una etapa del proyecto en la cual se asegura la calidad, sino que la prueba debe ocurrir durante todo el ciclo de vida: se puede probar la funcionalidad de los primeros prototipos; probar la estabilidad, cobertura y rendimiento de la arquitectura; probar el producto final, etc. Lo que conduce al principal beneficio de la prueba: proporcionar feedback mientras hay todavía tiempo y recursos para hacer algo.

La prueba es un proceso que se enfoca sobre la lógica interna del software y las funciones externas. La prueba es un proceso de ejecución de un programa con la intención de descubrir un error. Un buen caso de prueba es aquel que tiene alta probabilidad de mostrar un error no descubierto hasta entonces. Una prueba tiene éxito si descubre un error no detectado hasta entonces. La prueba no puede asegurar la ausencia de defectos; sólo puede demostrar que existen defectos en el software.

Cualquier proceso de ingeniería puede ser probado de una de dos formas:

- Se pueden llevar a cabo pruebas que demuestren que cada función es completamente operativa.

- Se pueden desarrollar pruebas que aseguren que la operación interna se ajusta a las especificaciones y que todos los componentes internos se han comprobado de forma adecuada.

La primera aproximación se denomina prueba de la caja negra y la segunda prueba de la caja blanca.

Prueba de caja blanca:

Permiten examinar la estructura interna del programa. Se diseñan casos de prueba para examinar la lógica del programa. Es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivar casos de prueba que garanticen que:

- Se ejercitan todos los caminos independientes de cada módulo
- Se ejercitan todas las decisiones lógicas
- Se ejecutan todos los bucles
- Se ejecutan las estructuras de datos internas

Prueba de caja negra:

Las pruebas se llevan a cabo sobre la interfaz del software, y es completamente indiferente el comportamiento interno y la estructura del programa.

Los casos de prueba de la caja negra pretenden demostrar que:

- Las funciones del software son operativas
- La entrada se acepta de forma adecuada
- Se produce una salida correcta
- La integridad de la información externa se mantiene

Se derivan conjuntos de condiciones de entrada que ejerciten completamente todos los requerimientos funcionales del programa.

La prueba de la caja negra intenta encontrar errores de las siguientes categorías:

- Funciones incorrectas o ausentes
- Errores de interfaz
- Errores en estructuras de datos o en accesos a bases de datos externas
- Errores de rendimiento
- Errores de inicialización y de terminación

Los casos de prueba deben satisfacer los siguientes criterios:

- Reducir, en un coeficiente que es mayor que uno, el número de casos de prueba adicionales
- Que digan algo sobre la presencia o ausencia de clases de errores

Tipos de Pruebas

Prueba de unidad:

La prueba de unidad se centra en el módulo. Usando la descripción del diseño detallado como guía, se prueban los caminos de control importantes con el fin de descubrir errores dentro del ámbito del módulo. La prueba de unidad hace uso intensivo de las técnicas de prueba de caja blanca.

Prueba de integración:

El objetivo es coger los módulos probados en la prueba de unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño.

Hay dos formas de integración:

- Integración no incremental: Se combinan todos los módulos por anticipado y se prueba todo el programa en conjunto

- Integración incremental: El programa se construye y se prueba en pequeños segmentos

En la prueba de integración el foco de atención es el diseño y la construcción de la arquitectura del software.

Las técnicas que más prevalecen son las de diseño de casos de prueba de caja negra, aunque se pueden llevar a cabo unas pocas pruebas de caja blanca.

Prueba del sistema:

Verifica que cada elemento encaja de forma adecuada y que se alcanza la funcionalidad y el rendimiento del sistema total. La prueba del sistema está constituida por una serie de pruebas diferentes cuyo propósito primordial es ejercitar profundamente el sistema basado en computadora.

Algunas de estas pruebas son:

Prueba de validación: Proporciona una seguridad final de que el software satisface todos los requerimientos funcionales y de rendimiento. Además, valida los requerimientos establecidos comparándolos con el sistema que ha sido construido. Durante la validación se usan exclusivamente técnicas de prueba de caja negra.

Prueba de recuperación: Fuerza un fallo del software y verifica que la recuperación se lleva a cabo apropiadamente.

Prueba de seguridad: Verificar los mecanismos de protección.

Prueba de resistencia: Enfrenta a los programas a situaciones anormales.

Prueba de rendimiento: Prueba el rendimiento del software en tiempo de ejecución.

Prueba de instalación: Se centra en asegurar que el sistema software desarrollado se puede instalar en diferentes configuraciones hardware y software y bajo condiciones excepcionales, por ejemplo con espacio de disco insuficiente o contínuas interrupciones.

Pruebas de regresión: Las pruebas de regresión son una estrategia de prueba en la cual las pruebas que se han ejecutado anteriormente se vuelven a realizar en la nueva versión modificada, para asegurar la calidad después de añadir la nueva funcionalidad.

El propósito de estas pruebas es asegurar que:

- Los defectos identificados en la ejecución anterior de la prueba se ha corregido
- Los cambios realizados no han introducido nuevos defectos o reintroducido defectos anteriores

La prueba de regresión puede implicar la re-ejecución de cualquier tipo de prueba. Normalmente, las pruebas de regresión se llevan a cabo durante cada iteración, ejecutando otra vez las pruebas de la iteración anterior.

Estrategias de pruebas del software:

Una estrategia de prueba del software integra las técnicas de diseño de casos de prueba en una serie de pasos bien planificados que llevan a la construcción correcta del software.

Las características generales son:

- La prueba comienza en el nivel de módulo y trabaja “hacia afuera”
- En diferentes puntos son adecuadas a la vez distintas técnicas de prueba
- La prueba la realiza la persona que desarrolla el software y (para grandes proyectos) un grupo de pruebas independiente
- La prueba y la depuración son actividades diferentes

Una estrategia de prueba para el software debe constar de pruebas de bajo nivel, así como de pruebas de alto nivel.

Más concretamente, los objetivos de la estrategia de prueba son:

- Planificar las pruebas necesarias en cada iteración, incluyendo las pruebas de unidad, integración y las pruebas de sistema. Las pruebas de unidad y de integración son necesarias dentro de la iteración, mientras que las pruebas de sistema son necesarias sólo al final de la iteración
- Diseñar e implementar las pruebas creando los casos de prueba que especifican qué probar, cómo realizar las pruebas y creando, si es posible, componentes de prueba ejecutables para automatizar las pruebas

- Realizar diferentes pruebas y manejar los resultados de cada prueba sistemáticamente. Los productos de desarrollo de software en los que se detectan defectos son probados de nuevo y posiblemente devueltos a otra etapa, como diseño o implementación, de forma que los defectos puedan ser arreglados

Para conseguir estos objetivos el flujo de trabajo de la etapa de prueba consta de las siguientes etapas:

- Planificación de las pruebas
- Diseño de las pruebas
- Implementación de las pruebas
- Ejecución de las pruebas
- Evaluación de las pruebas

Durante la fase de Inicio puede hacerse parte de la planificación inicial de las pruebas cuando se define el ámbito del sistema. Sin embargo, las pruebas se llevan a cabo cuando un producto de desarrollo software es sometido a pruebas de integración y de sistema. Esto quiere decir que la realización de pruebas se centra en las fases de elaboración, cuando se prueba la línea base ejecutable de la arquitectura y de construcción, cuando el grueso del sistema está implementado. Durante la fase de transición el centro se desplaza hacia la corrección de defectos durante los primeros usos y a las pruebas de regresión.

Debido a la naturaleza iterativa del esfuerzo de desarrollo, algunos de los casos de prueba que especifican cómo probar los primeros productos de desarrollo software pueden ser utilizadas también como casos de prueba de regresión que especifican cómo llevar a cabo las pruebas de regresión sobre los productos de desarrollo software siguientes. El número de pruebas de regresión necesarias

crece de forma estable a lo largo de las iteraciones, lo que significa que las últimas iteraciones requerirán un gran esfuerzo en pruebas de regresión. Es natural, por tanto, mantener el modelo de pruebas a lo largo del ciclo de vida del software completo, aunque el modelo de pruebas cambia constantemente debido a:

- La eliminación de casos de prueba obsoletos
- El refinamiento de algunos casos de prueba en casos de prueba de regresión
- La creación de nuevos casos de prueba para cada nuevo producto de desarrollo de software.

2.2.7 - Mantenimiento

Una vez que el software está en funcionamiento empieza la etapa de Mantenimiento. Es previsible que en proyectos de software hayan funcionalidades que sean de interés para el cliente, pero que fueron dejadas fuera del proyecto por razones de tiempo, presupuesto o simplemente para evaluarlas mejor en un futuro.

Hay empresas que ofrecen servicio de mantenimiento de software para esta etapa en la que se sigue trabajando en el desarrollo del software mediante la implementación de actualizaciones y correcciones. [15]

2.3 – Resultados de la encuesta aplicada

Al aplicar la encuesta se llegó a la conclusión de que existe una homogeneidad en los criterios emitidos por los diferentes encuestados. A continuación exponemos algunas de las ideas más generales de los encuestados.

En las distintas fases de desarrollo se llevan a cabo procesos de seguridad, quizás no siempre los más óptimos, pero sí se tiene en cuenta que los métodos y herramientas de invasión de la privacidad cambian continuamente. Por esta razón es importante considerar la seguridad como un aspecto

esencial desde el inicio de construcción de un software, ya que prevenir las irregularidades y fallos que éste pueda tener influirá grandemente en la calidad del producto y por consiguiente en la aceptación del mismo por el usuario final: el cliente.

Es necesario designar en un grupo de producción de software, un grupo de personas cualificadas respecto a la seguridad de un producto; este grupo se encargaría de revisar constantemente el progreso seguro de la aplicación que se necesite desarrollar y supervisaría a los desarrolladores indicándoles las mejores prácticas de programación que ayudarán a que las operaciones garanticen un por ciento aceptable de seguridad, lo que indicará en gran medida el grado de calidad que tendrá el resultado final.

Se impone el uso de una metodología enfocada a la seguridad, ya que de alguna manera se tendría una homogeneidad en todos los productos que salieran de la UCI, los productos tendrían una mayor seguridad y con esto, una mejor aceptación; así se podrían prever fallos en el software desde sus inicios, y serviría para que después de que el producto se haya terminado permita la recuperación del mismo ante cualquier rasgo de seguridad que se haya olvidado.

Conclusiones Parciales

En este capítulo se tuvieron en cuenta las diferentes fases por las que transcurre un software durante su desarrollo como paso preliminar al próximo capítulo en el que se encontrará con una explicación más detallada sobre la seguridad a lo largo de este proceso de desarrollo.

De los resultados de la encuesta aplicada a los diferentes líderes de proyecto, se obtuvo una homogeneidad de criterios en cuanto a la importancia que conlleva el aplicar la seguridad desde los comienzos hasta el último día de explotación al producto, así como lo beneficioso que sería la existencia de una metodología enfocada a la seguridad del software en todas sus fases de desarrollo.

En el próximo capítulo se abordará cómo esta metodología que hemos seleccionado: SDL, garantiza la seguridad del software.

Capítulo III: “Aplicando SDL en cada fase del software”

Introducción

Todos los proveedores de software deben tener en cuenta las amenazas de seguridad. La seguridad es un requisito básico dada la necesidad de proteger infraestructuras de gran importancia y crear y preservar la confiabilidad en la computación. Uno de los retos más importantes a los que se enfrentan todos los proveedores de software es crear un software más seguro que requiera menos actualizaciones y una administración de seguridad menos onerosa.

En el sector del software, la clave para cumplir la exigencia actual de una mayor seguridad está en implementar procesos reproducibles que proporcionen de manera confiable una mayor seguridad que se pueda medir. Por tanto, los proveedores de software deben adoptar un proceso de desarrollo más estricto que se centre, en mayor medida, en la seguridad. Este proceso debe diseñarse para minimizar el número de vulnerabilidades de seguridad presentes en el diseño, la programación y la documentación, así como para detectarlas y eliminarlas cuanto antes en el ciclo de vida de desarrollo. La necesidad de disponer de este proceso es mayor para el software profesional y doméstico que suele procesar información procedente de Internet, procesar información de identificación personal o controlar sistemas de gran importancia que pueden sufrir ataques.

Para lograr un software más seguro, hay que tener en cuenta tres aspectos:

- proceso reproducible,
- conocimientos del ingeniero e
- indicadores y responsabilidad

Después de tener identificado los principales problemas con la aplicación de la seguridad en todas las etapas de desarrollo del software y haber estudiado varias metodologías que han sido elaboradas a partir del año 2003, se propone como metodología a aplicar “SDL (Security Development Lifecycle)” ya que es una metodología que se encuentra un poco más completa en el tema de la seguridad del software en todas sus etapas de desarrollo, por lo que en este capítulo se verá principalmente la

reproducibilidad del proceso que propone SDL, aunque también aborda los conocimientos del ingeniero y ofrece algunos indicadores globales que muestran el impacto actual de la aplicación de un subconjunto de SDL.

SDL implica cambiar los procesos de una organización de desarrollo de software mediante la integración de medidas que mejoren la seguridad.

SDL resulta obligatorio para todo el software que:

- Se utilice para procesar información personal o confidencial
- Se utilice en una empresa u otra organización (incluidas organizaciones de enseñanza, gubernamentales o no lucrativas)
- Se conecte a Internet o se utilice de otra manera en un entorno de red

Existen determinados software que no necesitan la aplicación directa de la metodología de seguridad, por ejemplo aplicaciones independientes como juegos infantiles para niños de corta edad, sobre todo aquellas aplicaciones que no interfieran con la seguridad de la plataforma, o sea, sistema operativo u otro software en el que se ejecute la aplicación.

SDL se compone de un gran número de subprocesos componentes que se distribuyen por todo el ciclo de vida de desarrollo del software. A estos subprocesos se les debe asignar una prioridad en función de la eficacia, por ejemplo determinar cuáles son útiles y cuáles se han probado y resultan menos eficaces.

Según estudios realizados, el modelado de amenazas es el componente de mayor prioridad de SDL. Evidentemente, el modelado de amenazas no se aplica de manera aislada, sino que afecta al diseño, la revisión del código y las pruebas. Las estadísticas en forma de recuentos de errores tienden a subestimar la función del modelado de amenazas, ya que gran parte de la contribución que realiza el

modelado de errores consiste en asegurarse de que muchos de los errores que producirían vulnerabilidades de seguridad no lleguen a crearse. Sin embargo, la función del modelado de amenazas es de tal importancia para el proceso de desarrollo de software seguro que se sitúa claramente en el primer puesto de la lista.

Ni siquiera las más avanzadas herramientas actuales consiguen detectar todos los errores por lo que las revisiones manuales del código siguen siendo necesarias en SDL, tanto para detectar los errores que estas herramientas no detectan como para descubrir otras formas de mejorar las mismas.

Acudir a los consultores de seguridad, o sea personas sabias en el tema pero ajenas a las entidades involucradas en el desarrollo del software no es recomendable pues se le estaría confiando información que no debe de hacerse pública. Es más recomendable crear los grupos de seguridad para los proyectos sin que exista un grupo central que esté encargado de la seguridad de todos los proyectos pues cada uno de ellos tiene sus particularidades.

La creación de un grupo que se encargue de la seguridad para cada proyecto requeriría contratar a consultores que participen en la creación y entrenamiento de los miembros del equipo, pero luego de formado, este equipo sería el único que se encargaría de la seguridad en el proyecto y no un agente externo. Vale la pena emplear tiempo y recursos contables en la preparación de profesionales que garantizarán una alta calidad del producto resultante y de esta forma crecerá la satisfacción del cliente y la confianza en nuestro trabajo siempre que se use esta metodología de desarrollo seguro.

3.1 - El proceso de línea de base

El proceso de línea base es similar a una espiral. Los requisitos y el diseño con frecuencia se revisan durante la implementación para responder a los cambios en las realidades y las necesidades del mercado que surgen durante la realización del software. Además, el proceso de desarrollo destaca la necesidad de disponer de código ejecutable prácticamente en todos los puntos, por lo que cada uno de los puntos básicos más importantes puede dividirse en realidad en la entrega de una serie de versiones que sean operativas y se puedan probar (por el equipo de desarrollo) de manera continua.

3.2 - Introducción al desarrollo de la seguridad en el ciclo de vida de un software

La experiencia en seguridad del software real ha permitido establecer una serie de principios de alto nivel para lograr un software más seguro. Estos principios son: Seguro por diseño, Seguro por definición, Seguro en distribución y Comunicaciones. A continuación, se incluye una breve definición de estos principios:

- Seguro por diseño: la arquitectura, el diseño y la implementación del software se deben realizar de manera que proteja tanto el software como la información que procesa, además de poder resistir ataques.

- Seguro por definición: en el mundo real, el software no es nunca totalmente seguro, por lo que los diseñadores deben asumir que habrá errores de seguridad. Para minimizar los daños que se producirán cuando los atacantes descubran estos errores, el estado predeterminado del software debe elegir las opciones más seguras. Por ejemplo, el software debe ejecutarse con los mínimos privilegios necesarios y los servicios y las características que no sean necesarios de manera habitual deben deshabilitarse de manera predeterminada o establecer que sólo unos pocos usuarios puedan tener acceso a ellos.

- Seguro en distribución: se debe incluir con el software, información y herramientas que ayuden a los administradores y a los usuarios a utilizarlo de forma segura. Las actualizaciones deben implementarse de forma sencilla.

- Comunicaciones: los programadores de software deben estar preparados para detectar las vulnerabilidades de seguridad del producto y deben comunicarse de manera abierta y responsable con los usuarios y los administradores para ayudarles a tomar las medidas de protección adecuadas (como la actualización o la implementación de soluciones alternativas).

Aunque todos estos principios imponen ciertos requisitos durante el proceso de desarrollo, los dos primeros elementos, seguro por diseño y seguro por definición, son los que más favorecen la seguridad. Seguro por diseño obliga a utilizar procesos que tratan de evitar la inclusión de

vulnerabilidades de seguridad desde el principio, mientras que seguro por definición exige que la exposición predeterminada del software, la "superficie de ataque", sea la mínima posible.

Es importante señalar que el programa de educación es básico para que el SDL se realice correctamente. Al terminar los estudios universitarios o profesionales sobre informática y disciplinas afines, por lo general no se dispone de los conocimientos necesarios para comenzar a trabajar en un equipo destinado al diseño, el desarrollo o la prueba de software seguro. En general, los diseñadores de software, los ingenieros y los encargados de las pruebas también carecen de los conocimientos adecuados sobre seguridad.

Teniendo esto en cuenta, toda organización que quiera desarrollar software seguro deberá asumir la responsabilidad de asegurarse de que sus empleados adquieren los conocimientos necesarios. La manera concreta de hacerlo depende del tamaño de la organización y los recursos disponibles. Una organización con un gran número de ingenieros puede crear un programa de educación interno que proporcione a su personal los conocimientos de seguridad de manera continua, mientras que una organización de menor tamaño es posible que deba recurrir a servicios de enseñanza externos.

3.3 - Fase de requisitos

La necesidad de considerar la seguridad "de abajo a arriba" es uno de los principios fundamentales del desarrollo de sistemas seguros. Teniendo en cuenta que muchos proyectos de desarrollo generan la siguiente versión a partir de la anterior, la fase de requisitos y el planeamiento inicial de una nueva versión o lanzamiento ofrece una oportunidad estupenda para crear software seguro.

Durante la fase de requisitos, el equipo de desarrollo del producto se pone en contacto con el equipo de seguridad del proyecto para solicitar la asignación de un asesor de seguridad que actúa como punto de contacto, recurso y guía a través de los procedimientos de planeamiento. El asesor de seguridad ayuda al equipo de desarrollo del producto revisando los planes, aportando recomendaciones y asegurándose de que el equipo de seguridad planea los recursos necesarios de acuerdo con el programa de fechas del equipo de desarrollo. El asesor de seguridad advierte de los puntos básicos de seguridad y los criterios de salida que serán necesarios en función del tamaño, la

complejidad y los riesgos del proyecto; éste también actúa como contacto entre el equipo de desarrollo del producto y el equipo de seguridad desde el inicio del proyecto hasta la finalización de la revisión final de seguridad y el lanzamiento del software. El asesor de seguridad también hace de contacto entre el equipo de seguridad y la administración del equipo de producto, informando a este último del correcto avance de la seguridad del proyecto para evitar sorpresas de última hora relacionadas con la seguridad.

La fase de requisitos es la oportunidad ideal para que el equipo de producto se plantee cómo se integrará la seguridad en el proceso de desarrollo, identifique los objetivos de seguridad clave y, por lo demás, maximice la seguridad del software procurando minimizar el impacto sobre los planes y los programas. Como parte de este proceso, el equipo debe considerar cómo se integrarán las características de seguridad y las medidas de control con otros programas que probablemente se utilizarán con el software que está desarrollando. (El funcionamiento con otros programas es vital para responder a la necesidad de los usuarios de integrar los productos en sistemas seguros) La consideración general de los objetivos, los retos y los planes de seguridad debe reflejarse en los documentos de planeamiento generados durante la fase de requisitos. Aunque es posible que estos planes cambien a medida que el proyecto avanza, articularlos desde el principio garantiza que no se pasa por alto ningún requisito ni surgen sorpresas de última hora.

Cada equipo de desarrollo del producto debe considerar los requisitos de características de seguridad como parte de esta fase. Aunque algunos requisitos de características de seguridad aparecerán a partir del modelo de amenazas, es probable que sean los requisitos de los usuarios los que dictaminen la inclusión de características de seguridad como respuesta a las demandas de los clientes. Los requisitos de características de seguridad también surgirán a partir de la necesidad de cumplir los estándares del sector y los procesos de certificación, como los criterios comunes. El equipo de producto debe detectar y reflejar estos requisitos como parte de su proceso de planeamiento normal.

3.4 - Fase de diseño

La fase de diseño identifica la estructura y los requisitos globales del software. Desde el punto de vista de la seguridad, los elementos clave de la fase de diseño son:

- Definir la arquitectura de seguridad y las directrices de diseño: definir la estructura global del software desde el punto de vista de la seguridad e identificar los componentes cuyo correcto funcionamiento es esencial para la seguridad. La identificación de técnicas de diseño, como el uso de capas o lenguaje con tipos inflexibles, la aplicación de privilegios mínimos y la minimización de la superficie de ataque, que se aplican al software de manera global. (El uso de capas se refiere a la organización del software en componentes bien definidos que se estructuran para evitar dependencias circulares entre componentes. Los componentes se organizan en capas y una capa superior puede depender de los servicios de capas inferiores, pero se prohíbe que las capas inferiores dependan de las capas superiores.) Los detalles específicos de cada uno de los elementos de la arquitectura se indican en las especificaciones de diseño individuales, pero la arquitectura de seguridad corresponde a una perspectiva global sobre el diseño de seguridad.

- Documentar los elementos de la superficie de ataque del software. Teniendo en cuenta que el software no logrará una seguridad perfecta, es importante que únicamente se expongan de manera predeterminada las características que utilicen la mayoría de los usuarios y que dichas características se instalen con el mínimo nivel de privilegios posible. La medición de los elementos de la superficie de ataque ofrece al equipo de producto un indicador continuo de la seguridad predeterminada y les permite detectar las instancias en las que el software es más susceptible de recibir ataques. Aunque algunas instancias con mayor superficie de ataque pueden estar justificadas por una mayor facilidad de uso o unas mejores funciones del producto, es importante detectar y considerar cada una de estas instancias durante el diseño y la implementación para lanzar el software con la configuración predeterminada más segura posible.

- Realizar un modelado de las amenazas. El equipo debe realizar un modelado de amenazas por componentes. Mediante una metodología estructurada, el equipo de componentes identifica los activos que debe administrar el software y las interfaces que permitirán el acceso a dichos activos. El proceso de modelado de amenazas identifica las amenazas que pueden dañar a estos activos y la probabilidad de que se inflija dicho daño (estimación del riesgo). A continuación, el equipo de componente identifica las contramedidas que pueden mitigar el riesgo, ya sea mediante características de seguridad (por ejemplo, el cifrado) o mediante un funcionamiento correcto del software que proteja a los activos del daño. Por tanto, el modelado de amenazas ayuda al equipo de producto a identificar las necesidades de características de seguridad y las áreas en las que es necesario revisar con especial minuciosidad el código y probar la seguridad. El proceso de modelado de amenazas debe realizarse con una herramienta capaz de capturar modelos de amenazas en un formato que pueda leer un equipo para almacenarlo y actualizarlo.

- Definir los criterios de publicación adicionales. Aunque los criterios de publicación de seguridad básicos deben definirse para toda la organización, puede que existan criterios concretos para determinados equipos de producto o lanzamientos de software que sea preciso cumplir para poder lanzar el software. Por ejemplo, un equipo de producto dedicado al desarrollo de una versión actualizada del software que se enviará a los clientes y que está expuesta a un gran número de ataques, puede optar por exigir que la nueva versión no presente ninguna de las vulnerabilidades de seguridad detectadas durante cierto tiempo antes de considerar que está lista para su lanzamiento (es decir, el proceso de desarrollo debe descubrir las vulnerabilidades de seguridad y solucionarlas antes de que se detecten, en vez de tener que solucionarlas después de su detección).

3.5 - Fase de implementación

Durante la fase de implementación, el equipo de producto programa, prueba e integra el software. Los pasos destinados a eliminar los errores de seguridad o a impedir que se incluyan desde el principio son de gran utilidad, ya que reducen considerablemente la probabilidad de que las vulnerabilidades de seguridad lleguen a la versión final del software que se lanzará a los clientes.

Los resultados del modelado de amenazas ofrecen una orientación especialmente importante durante la fase de implementación. Los programadores deben asegurarse de que escriben correctamente el código para mitigar las amenazas de alta prioridad, mientras que los encargados de las pruebas deberán asegurarse de que estas amenazas se han bloqueado o mitigado de manera efectiva.

Los elementos del SDL que se aplican en la fase de implementación son:

- Aplicar estándares de codificación y de pruebas. Los estándares de codificación evitan que los programadores incluyan errores que puedan producir vulnerabilidades de seguridad. Por ejemplo, el uso de construcciones de manipulación de búferes y de manejo de cadenas más seguras y coherentes ayuda a evitar la aparición de vulnerabilidades de seguridad de saturación del búfer. Los estándares de pruebas y las prácticas recomendadas permiten garantizar que las pruebas se centran en detectar posibles vulnerabilidades de seguridad, en vez de centrarse únicamente en el funcionamiento correcto de las características y las funciones del software.

- Aplicar herramientas de comprobación de seguridad, incluidas herramientas de confusión. Estas herramientas ofrecen entradas estructuradas pero no válidas a las interfaces de programación de aplicaciones (API) de software y a las interfaces de red para maximizar la probabilidad de detectar errores que puedan ocasionar vulnerabilidades de seguridad del software.

- Aplicar herramientas de exploración del código de análisis estático. Las herramientas pueden detectar algunos tipos de errores de codificación que producen vulnerabilidades de seguridad, incluidas saturaciones de búfer, desbordamientos con enteros y variables no inicializadas.

- Realizar revisiones del código. Las revisiones del código complementan las herramientas automatizadas y las pruebas, ya que aplican el esfuerzo de programadores expertos para examinar el código fuente y detectar y eliminar posibles vulnerabilidades de seguridad. Estas revisiones constituyen un paso fundamental para eliminar las vulnerabilidades de seguridad del software durante el proceso de desarrollo.

3.6 - Fase de comprobación

La fase de comprobación es el punto en el que software ya incorpora toda la funcionalidad y los usuarios pueden comenzar a probar la versión beta. Durante esta fase, mientras se prueba la versión beta del software, el equipo de producto realiza una campaña de seguridad que incluye revisiones del código de seguridad aparte de las realizadas en la fase de implementación, así como la realización de pruebas centradas en la seguridad.

Ejemplo:

Microsoft realizó campañas de seguridad durante la fase de comprobación de Windows Server 2003 y otras versiones de software a principios de 2002. Existían dos motivos para introducir la campaña de seguridad en el proceso:

- El ciclo de vida del software de las versiones en cuestión había alcanzado la fase de comprobación, que era un punto adecuado para realizar las revisiones del código y las pruebas necesarias

- La realización de la campaña de seguridad durante la fase de comprobación asegura que la revisión del código y las pruebas se realizan con la versión terminada del software y ofrece

una oportunidad de revisar tanto el código desarrollado o actualizado durante la fase de implementación como el código heredado que no se ha modificado

El primero de estos motivos refleja un accidente histórico: la decisión de iniciar una campaña de seguridad se tomó en principio durante la fase de comprobación. Pero Microsoft ha llegado a la conclusión de que es una buena idea realizar una campaña de seguridad durante esta fase, tanto para asegurar que el software final cumple los requisitos como para permitir una revisión en detalle de todo el código heredado de versiones anteriores del software.

Hay que resaltar que las revisiones del código y las pruebas del código de alta prioridad (código que forma parte de la "superficie de ataque" del software) son esenciales para varias partes de SDL. Por ejemplo, estas revisiones y pruebas deben ser obligatorias en la fase de implementación para corregir cuanto antes los problemas, así como para identificar y corregir los orígenes de dichos problemas. También son fundamentales en la fase de comprobación cuando esté a punto de finalizarse.

3.7- Fase de lanzamiento

Durante la fase de lanzamiento, el software debe someterse a una revisión final de seguridad (FSR). Esta FSR debe responder a la siguiente pregunta: "Desde el punto de vista de la seguridad, ¿está este software preparado para los clientes?" La FSR se realiza en un plazo de dos a seis meses antes de la finalización del software, según el alcance del software. El software debe ser estable antes de la FSR y es de esperar que antes del lanzamiento sólo se realicen cambios mínimos y no relacionados con la seguridad.

La FSR es una revisión independiente del software que realiza el equipo de seguridad central de la organización. El asesor de seguridad del equipo de seguridad aconseja al equipo de producto sobre el ámbito de la FSR que requiere el software y ofrece al equipo de producto una lista de los requisitos de recursos antes de la FSR. El equipo de producto proporciona al equipo de seguridad los recursos y la información necesarios para llevar a cabo la FSR. Al comienzo de la FSR, el equipo de producto debe rellenar un cuestionario [Anexo 2] y entrevistarse con un miembro del equipo de seguridad asignado a la FSR. En toda FSR se deben revisar los errores que se identificaron en un principio

como errores de seguridad, pero que tras analizarlos se consideró que no afectaban a la seguridad, y de esta forma asegurarse de que este análisis es correcto. Una FSR también incluye una revisión de la capacidad del software para soportar vulnerabilidades de seguridad detectadas recientemente en un software similar. Una FSR para una versión de software importante requerirá realizar pruebas de penetración y, posiblemente, recurrir a asesores de revisión de seguridad externos que ayuden al equipo de seguridad.

La FSR no es únicamente un examen que se puede aprobar o suspender ni tampoco pretende detectar todas las vulnerabilidades de seguridad que quedan en el software, lo que no sería factible, sino proporcionar al equipo de producto y a la administración superior de la organización una idea global del nivel de seguridad del software y de la probabilidad de que pueda resistir ataques una vez que se haya entregado a los clientes. Si la FSR detecta patrones de vulnerabilidades de seguridad restantes, no bastará con solucionar las vulnerabilidades detectadas, sino que habrá que repetir la fase anterior y tomar las acciones necesarias para tratar los orígenes (por ejemplo, mejorar los conocimientos, mejorar las herramientas).

3.8 - Fase de servicio técnico y mantenimiento

A pesar de la aplicación del SDL durante el desarrollo, las prácticas de desarrollo más avanzadas no consideran que se pueda publicar software que no tenga ninguna vulnerabilidad de seguridad (y existen buenos motivos para creer que siempre será así). Incluso aunque el proceso de desarrollo pudiera eliminar todas las vulnerabilidades de seguridad del software que se va a publicar, se descubrirían nuevos ataques y el software considerado "seguro" pasaría a ser vulnerable. Por tanto, los equipos de producto deben prepararse para responder a nuevas vulnerabilidades en el software que se entrega a los clientes.

Parte del proceso de respuesta consiste en la preparación para evaluar los informes de vulnerabilidades y lanzar consejos y actualizaciones de seguridad siempre que sea necesario. El otro componente del proceso de respuesta consiste en realizar una autopsia de las vulnerabilidades detectadas y tomar las medidas oportunas. Estas medidas pueden oscilar desde el lanzamiento de

una actualización que resuelva un error aislado hasta la renovación de las herramientas de actualización del código para iniciar revisiones de los subsistemas principales. El objetivo durante la fase de respuesta consiste en aprender de los errores y utilizar la información de los informes de vulnerabilidades para detectar y eliminar otras vulnerabilidades antes de que se descubran en la práctica y se utilicen para poner en peligro a los clientes. El proceso de respuesta también ayuda a los equipos de producto y de seguridad a adaptar los procesos para que otros errores similares no se repitan en el futuro. [26]

Conclusiones Parciales

SDL reduce la incidencia de vulnerabilidades de seguridad. La implementación inicial de SDL (en Windows Server 2003, SQL Server 2000 Service Pack 3 y Exchange 2000 Server Service Pack 3) ha conseguido mejorar considerablemente la seguridad del software. Además, las versiones posteriores de software, que incorporan las mejoras realizadas al SDL, también parece que siguen mejorando la seguridad del software.

La implementación incremental de los elementos que componen el SDL también ha producido un aumento en las mejoras, lo que se considera fruto de la eficacia del proceso. El proceso no es perfecto y sigue evolucionando, de hecho, es poco probable que alcance la perfección o deje de evolucionar en un futuro próximo.

Se recomienda la utilización de SDL dada su organización en el control de la seguridad y su estrecha relación con el proceso de desarrollo de acuerdo a la Ingeniería de Software. Esta metodología garantiza un gran por ciento de seguridad que se espera precisar si se realiza su despliegue en algunos de los proyectos productivos que se desarrollan o se desarrollarán en la UCI. Muchas veces los programadores o desarrolladores de software pasan por alto los métodos de diseño o planificación establecidos, pero esta importante proyección de lo que se desarrollará es lo que dará la certeza o la mayor probabilidad de que no se han cometido errores en estos aspectos cuando se enfrentan directamente a la materialización de la aplicación o el producto.

Conclusiones

En este trabajo se ha demostrado la importancia de tener en cuenta la seguridad del software y de las aplicaciones cuando programamos o cuando se va a usar, por poco complejos que sean éstos.

Dada esta necesidad se demuestra también que la UCI como institución docente productora de software, precisa de una metodología de programación segura que controle el avance del producto y la posterior prueba y ejecución del mismo para, de esta forma, evitar las posibles fallas en el sistema que se desarrolla y controlar las violaciones a las que pueda exponer.

Se desarrolló un proceso investigativo en busca de una metodología convincente para producir un software permisiblemente seguro, a partir del cual se decidió el uso de una en específico: SDL, que reportaba la mayor cantidad de características que hacen un producto más fuerte y confiable.

Se profundizó en los aspectos básicos del desarrollo y construcción de un software observando cada una de las fases que le da vida, haciendo especial atención en las fases conocidas de la disciplina Ingeniería de Software.

Después de haber realizado este análisis sobre metodologías de desarrollo de software teniendo en cuenta como aspecto fundamental la seguridad o la falta de la misma en la que se ve enmarcado un producto, se concluyó que el uso de SDL prevé huecos de seguridad en el sistema que se desarrolla y ayuda a construir un software altamente confiable y por tanto más eficaz.

Como principales beneficios de usar esta metodología y resultados más probables de aplicarla a los proyectos productivos determinamos:

- Se obtiene un software más confiable y seguro
- Se minimiza el período de prueba
- Se aumenta considerablemente la calidad del producto resultante
- Se gana la confianza del cliente

El desarrollo y la implementación del ciclo de desarrollo de seguridad del software representan una importante inversión y constituye un importante cambio en la manera de diseñar, desarrollar y probar el software. La importancia cada vez mayor del software para la sociedad resalta la necesidad de mejorar la seguridad del mismo.

Recomendaciones

Este trabajo propone una metodología enfocada a la seguridad del software en todas sus fases de desarrollo como una forma de obtener un producto seguro y una homogeneidad en su realización. No obstante la investigación y análisis que se ha realizado y expresado se puede profundizar aún más y/o aportarse ideas que prevengan las vulnerabilidades de seguridad en los productos software que se realicen; además de que también pueden proponerse métodos que hagan el uso de estas metodologías de programación segura, más dinámico y atractivo para los programadores. Dado que este tema es muy cambiante y surgen constantemente diversas formas de ataque, se asegura que su búsqueda o su valoración serán de mucho valor para los usuarios.

Referencias bibliográficas

Libros Electrónicos

1. Dr. Ramió Aguirre, Jorge. Consultado en: marzo, 2007. Seguridad Informática y Criptografía Versión 4.1. Universidad Politécnica de Madrid. Disponible en: http://www.criptored.upm.es/guiateoria/gt_m001a.htm

2. M. en C. Farias-Elinos, Mario. Consultado en: abril, 2007. La Importancia de los Estándares en Seguridad Informática. Lab. de Investigación y Desarrollo de Tecnología Avanzada (LIDETEA). Universidad La Salle. Disponible en: http://seguridad.internet2.ulsa.mx/congresos/2002/enep-aragon/std_seguridad.pdf

Sitios Web

3. Ing. Cano, Jeimy. Consultado en: febrero, 2007. Breves reflexiones sobre la programación segura. Belt Ibérica S_A. Universidad de los Andes. Disponible en: http://www.belt.es/expertos/HOME2_experto.asp?id=2560

4. Gómez Velazco, Lidia Elena y Rico Rodríguez, Verónica. Consultado en: mayo, 2007. Importancia de la seguridad informática. Disponible en: http://gaceta.cicese.mx/ver.php?topico=breviario&ejemplar=60&id=374

5. Dossier Seguridad. Consultado en: abril, 2007. Seguridad en los sistemas informáticos. Concepto. Disponible en: http://www.recursos-as400.com/dossier/seguretat/02.shtml

6. Marjedan, Alejo. Centro de Alerta Temprana sobre Virus Informáticos. Marzo 2.002. Consultado en: mayo, 2007. La seguridad de la información. Disponible en: http://alerta-antivirus.red.es/seguridad/ver_pag.html?tema=S&articulo=4&pagina=1

7. Álvarez Marañón, Gonzalo. Consultado en: marzo, 2007. Amenazas deliberadas a la seguridad de la información. Disponible en:
http://www.iec.csic.es/criptonomicon/seguridad/amenazas.html

8. Políticas de seguridad.
http://es.tldp.org/Manuales-LuCAS/doc-unixsec/unixsec-html/node333.html

9. Guía para el desarrollo de aplicaciones Web seguras. Disponible en:
http://www.desarrolloweb.com/articulos/996.php

10. La seguridad en la Ingeniería de Software. Disponible en:
http://www.acm.org/crossroads/espanol/xrds7-4/onpatrol74.html

11. Seguridad Informática. Revista RED, La comunidad de expertos en redes. Noviembre, 2002. Disponible en:
http://ciberhabitat.gob.mx/museo/cerquita/redes/seguridad/intro.htm

12. Metodología OSSTMM (Open Source Security Testing Methodology Manual). Disponible en:
http://www.ideahamster.org/osstmm-description.htm

13. Bonaverdi, Ignacio. De la Redacción de LANACION.com. Disponible en:
http://www.lanacion.com.ar/tecnologia/topfive.asp?tema_id=4&nota_id=814895

14. Etapas del Proceso de Desarrollo del Software. Disponible en:
http://lsi.ugr.es/~arroyo/inndoc/doc/requisitos/requisitos_ft.php

15. Ciclo de vida del software. Disponible en:
http://lsi.ugr.es/~arroyo/inndoc/doc

16. Modelo de desarrollo de software. Disponible en:
http://www.angelfire.com/scifi/jzavalar/apuntes/ds.html

17. Microsoft comprometido con Latinoamérica. Disponible en:
http://www.pcwla.com/pcwla2.nsf/articulos/9000D9A77AA61AE2852572CE0001C166

18. Vulnerabilidad. Disponible en:
http://es.wikipedia.org/wiki/Vulnerabilidad

19. Glosario de Seguridad. Disponible en:
http://alerta-antivirus.red.es/seguridad/ver_pag.html?tema=S&articulo=9&letra=A

20. Diccionario básico de Informática. Disponible en:
http://usuarios.lycos.es/Resve/diccioninform.htm

21. Información y significado de Overflow. Disponible en:
http://www.alegsa.com.ar/Dic/overflow.php.

22. Actualidad Informática - Diccionario - Definición de Configurar. Disponible en:
http://www.mastermagazine.info/termino/4404.php

23. Software. Disponible en:
http://es.wikipedia.org/wiki/Software

24. E-arquitectura. Disponible en:
http://muchomaz.blogspot.com/2006/12/informtica-o-computacin.html

Revistas digitales

25. IEEE STD, IEEE Software Engineering Standard: Glossary of Software Engineering Terminology. IEEE Computer Society Press, 1993

Libros

26. Howard, Michael y Lipner, Steve. The Security Development Lifecycle. Microsoft Press y Redmond Washington Junio, 2006.

Artículos

27. Tovar Edmundo2, Carrillo José2, Vega Vianca1 y Gasca Gloria2. Desarrollo de productos de software seguros en sintonía con los modelos SSE-CMM, COBIT E ITIL. Universidad Católica del Norte1 y Universidad Politécnica de Madrid2. Septiembre, 2006.

28. Carrasco, Rafael San Miguel. Metodología de Desarrollo Web Seguro Basado en Formación a Medida. Septiembre, 2005.

29. Plan de Seguridad Integral de Sistemas de Información de la Diputación Foral de Gipuzkoa. Marzo, 2005

30. Rodríguez, Víctor A. SPSMM: Standard de programación segura. Mayo, 2002

31. Wheeler, David A. Secure Programming for Linux and Unix HOWTO. Marzo 2003

Glosario de términos

Metodología: Conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar nuevo software.

Vulnerabilidad: En términos de Seguridad de la Información, una vulnerabilidad es una debilidad en los procedimientos de seguridad, diseño, implementación o control interno que podría ser explotada (accidental o intencionalmente) y que resulta en una brecha de seguridad o una violación de las políticas de seguridad del sistema. [18]

Parches: Conjunto de ficheros adicionales al software original de una herramienta o programa informático, que sirven para solucionar sus posibles carencias, vulnerabilidades, o defectos de funcionamiento. También conocidos como actualizaciones. En sistemas operativos Windows, son normalmente programas ejecutables que reemplazan las anteriores versiones para complementarlas y solucionar problemas sobre agujeros de seguridad; en otras plataformas también son conocidos como PTFs (Program Temporary Fixes). Existen conglomerados de parches, más estables, que incluyen varias actualizaciones a diversas fallas, que suelen ser llamados Service Packs, y son liberados cada cierto tiempo por las empresas responsables de las aplicaciones y sistemas operativos más utilizados. [20]

Configuración: Adaptar una aplicación software o un elemento hardware al resto de los elementos del entorno y a las necesidades específicas del usuario. Es una tarea esencial antes de trabajar con cualquier nuevo elemento. La tendencia actual es a reducir las necesidades de configuración mediante sistemas que permiten al nuevo elemento detectar en qué entorno se instala, configurándose automáticamente sin requerir la participación del usuario. Cuando ésta es necesaria, se intenta facilitar al máximo el proceso de configuración. [23]

BIOS: Basic Input / Output System) Identifica el software o conjunto de programas que arrancan el ordenador (antes de encontrarse un disco de sistema) cuando se pulsa el botón de encendido. El BIOS se encuentra siempre en la memoria principal, pero no en la RAM (Random Access Memory)

pues al apagar el ordenador se borraría, sino en la ROM (Read Only Memory - Memoria de Sólo Lectura), cuyo almacenamiento es permanente. [20]

Protocolos: Normas a seguir en una cierta comunicación: formato de los datos que debe enviar el emisor, cómo debe ser cada una de las respuestas del receptor, etc. [21]

TCP/IP: (Transfer Control Protocol / Internet Protocol) Protocolo de control de transmisión/Protocolo de Internet. Conjunto de protocolos sobre los cuales funciona Internet, permite la comunicación entre los millones de equipos informáticos conectados a dicha red. El protocolo IP, que se ocupa de transferir los paquetes de datos hasta su destino adecuado y el protocolo TCP, que se ocupa de garantizar que la transferencia se lleve a cabo de forma correcta y confiable. [20]

Overflow (desbordamiento): Exceso de datos que pueden ser perdidos o transferidos. [21]

Buffer: Memoria intermedia para el almacenamiento de datos temporales en la comunicación entre un ordenador y un dispositivo externo (p.ej., una impresora). Cuando es un programa informático el que hace la misión de almacenamiento intermedio para los datos que se envían a la impresora, a dicho programa se le suele llamar Spooler. [21]

Software: Todos los componentes intangibles de una computadora, es decir, al conjunto de programas y procedimientos necesarios para hacer posible la realización de una tarea específica, en contraposición a los componentes físicos del sistema (hardware). Esto incluye aplicaciones informáticas tales como un procesador de textos, que permite al usuario realizar una tarea, y software de sistema como un sistema operativo, que permite al resto de programas funcionar adecuadamente, facilitando la interacción con los componentes físicos y el resto de aplicaciones. [24] Probablemente la definición más formal de software es la atribuida a la IEEE en su estándar 729: «la suma total de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de cómputo». [25]

E-arquitectura: El arte de proyectar y construir soluciones tecnológicas para el uso del hombre.[24]

Anexos

Anexo 1

El siguiente cuestionario ha sido diseñado para realizar un estudio en la comunidad universitaria de la cual usted forma parte, sobre el control de la seguridad en todo el ciclo de desarrollo del software, esperamos su total seriedad y sinceridad en las respuestas a estas preguntas.

Nota: Cuando hablamos de seguridad nos referimos a la seguridad que debe de tener todo software para enfrentar los ataques informáticos, dígase virus, hackers, entre otros.

Marque el proyecto al que usted pertenece:

___ 171
___ Procyon
___ IL-96
___ PKI
___ SEGURMÁTICA
___ Calidad
___ Call Center
___ Informatización
___ ICRT
___ Informática UCI
___ Grupo Arquitectura

¿Se controla la seguridad del software en su proyecto?

___ Si ___ No

¿En cuáles etapas del desarrollo de software usted controla la seguridad del mismo?

___ Modelamiento del negocio
___ Levantamiento de requisitos
___ Análisis y diseño
___ Implementación

___ Prueba
___ Instalación
___ Administración del proyecto
___ Administración de configuración y cambios
___ Ambiente

¿Qué opina usted sobre el control de la seguridad en todas las etapas o flujos de desarrollo del software?

_____
_____
_____
_____
_____
_____

¿Qué propone usted para que se lleve a cabo la seguridad en todo el ciclo de desarrollo del software?

_____
_____
_____
_____

Cree usted necesario una metodología que sirva de guía para llevar a cabo la seguridad en el ciclo de desarrollo del software. ¿Por qué?

_____
_____
_____
_____

Le agradecemos el tiempo dedicado por usted para responder este formulario.

Anexo 2

Documento de Evaluación de Riesgos (Versión 3.2, 22/05/ 2007)

Información de Revisión	
Fecha de aplicación de la encuesta:	
Nombres de las personas que llenaron esta encuesta:	
¿Quién es el principal encargado de las actividades en torno a la seguridad en su proyecto?	
Información del Producto/Componente	
Nombre del componente:	
¿Dónde está su código fuente?	
¿Donde tienen su BD de errores?	
¿Dónde están sus modelos de prueba?	
¿Qué plataformas de Sistema Operativo soporta?	☐ Win2000, ☐ Windows XP, ☐ Windows Vista, ☐ Mac, ☐ *nix, ☐ Windows CE, ☐ Otro:
¿Su aplicación soporta o corre en	☐ Si, ☐ No

plataformas de bajo nivel?	
Seguridad General	
¿El funcionamiento de su componente varía cuando se utiliza en una plataforma multiusuario?	☐ Si, ☐ No Explique:
¿Su componente requiere que el usuario sea administrador?	☐ Si, ☐ No, ☐ N/A
¿De qué servicios Standard de Windows depende su producto?	☐ No sé, Depende de:
Autenticación	
¿UD realiza alguna especie de autenticación?	☐ Si, ☐ No
¿UD realiza alguna especie de Autenticación avanzada?	☐ Si, ☐ No, ☐ N/A
¿Cómo lo hace?	

Autorización	
UD utiliza alguna especie de mecanismo de autorización?	☐ Si, ☐ No
UD utiliza los ACL (Access Control List) de WINDOWS?	☐ Si, ☐ No, ☐ N/A
¿UD modifica algún ACL en particular?	☐ Si, ☐ No, ☐ N/A
¿UD realiza alguna especie de Autorización avanzada?	☐ Si, ☐ No, ☐ N/A
¿Cómo lo hace?	
¿UD implementa algún tipo de Autorización basada en la pertenencia	☐ Si, ☐ No

a grupos de usuarios o acceso? (Ej. Examinando el token del usuario para determinar si el mismo es miembro de algún grupo específico antes de permitir la ejecución de alguna operación).	
Encriptación	
¿UD utiliza Encriptación de canales??	☐ Si, ☐ No
¿Si es así, cuál utiliza?	☐ SSL ☐ RPC privacidad/integridad, ☐ DCOM privacidad/integridad, ☐ Otra
¿UD utiliza la Encriptación de datos persistentes?	☐ Si, ☐ No
¿UD tiene su propio código de Encriptación en su árbol de código fuente?	☐ Si, ☐ No
Si es así, ¿Dónde y por qué?	
¿UD utiliza Crypto API o System.Security.Cryptography?	☐ Si, ☐ No, ☐ Para algunas cosas:
¿UD almacena contraseñas?	☐ Si, ☐ No
¿Dónde las almacena?	☐ Secretos LSA, ☐ Protección de Datos API, ☐ Otros:
¿UD utiliza alguna forma de generación de códigos o números aleatorios para la seguridad?	☐ Si, ☐ No
Si es así, ¿Qué API UD utiliza para estos números?	
¿Para qué UD utiliza estos códigos o números aleatorios?	
¿UD integra alguna forma de	☐ Si, ☐ No

contraseña o clave secreta en su componente?	
¿UD integra algún tipo de ruta o ubicación para claves públicas en su componente?	☐ Si, ☐ No

¿Dónde?	
Archivos y Datos	
¿UD modifica explícitamente los ACLs en estos archivos temporales?	Mencione todos los directorios donde los programas(ejecutables) son almacenados
¿Qué nombre le pone UD a estos archivos temporales?	¿Cuáles son los ACLs en los programas (ejecutables) y sus directorios?
¿UD almacena información de los usuarios?	☐ Si, ☐ No
¿Dónde? ¿Cómo la protege?	
¿UD cambia los ACLs en esta ubicación en el momento de la instalación?	☐ Si, ☐ No
¿UD almacena información recibida de una fuente desconocida?	☐ Si, ☐ No
Si es así, ¿Cuál? ¿Cómo la elimina?	
¿UD cambia los ACLs en esta ubicación en el momento de la actualización?	☐ Si, ☐ No
¿Dónde es almacenada? ¿Cómo limita la cantidad de información que puede almacenar un cliente?	
¿UD cambia los ACLs en esta ubicación en el momento de la instalación?	☐ Si, ☐ No
Capacidades de Acceso Remoto y Servicio de Mensajería	
¿UD tiene herramientas de control remoto?	☐ Si, ☐ No
Mencione todas las claves de registro que crea o modifica su componente.	
¿Cómo trabaja esto a través de un Firewall?	¿Qué son los ACLs en esta clave?
¿UD cambia los ACLs en esta clave en el momento de la instalación?	☐ Si, ☐ No
¿UD cambia los ACLs en esta clave en el momento de corrida?	UD usa: ☐ DCOM, ☐ RPC, ☐ Sockets, ☐ Named Pipes, ☐ DDE ☐ Si, ☐ No ☐ NetDDE, ☐ LDAP, ☐ SOAP, ☐ Otros:
Para RPC	
¿UD almacena archivos temporales?	☐ Si, ☐ No

¿UD realiza algún tipo de monitoreo o control avanzado de RPC?	☐ Si, ☐ No, ☐ N/A
¿UD crea recursos compartidos SMB?	☐ Si, ☐ No
Si es así ¿Qué comparte?	

ActiveX	
¿UD utiliza algún control de Active X?	☐ Si, ☐ No
Por favor menciónelos:	
Servicios	
¿UD instala algún servicio?	☐ Si, ☐ No Mencione sus nombres:
¿Están instalados basados en algún Standard de Windows? (o de su producto)?	☐ Si, ☐ No Comente:
¿Es el servicio diferente dependiendo de la versión del producto?	☐ Si, ☐ No Explique:
¿Cuál es el tipo de inicio Standard de su servicio luego de la instalación?	☐ Automático, ☐ Manual, ☐ Deshabilitado
¿Utiliza su servicio memoria compartida?	☐ Si, ☐ No, Mencione, y Explique cómo se utiliza:
¿Cómo funciona su servicio?	☐ Sistema Local, ☐Servicio Local, ☐ Servicio de red, ☐ Configurable:
¿Los servicios de su componente muestran directamente UI en el escritorio del usuario?	☐ Si, ☐ No Explique:
¿Su servicio interactúa con alguna	☐ Si, ☐ No Si es así, mencione la(s) interfase(s):

interfaz o estructura de red?	
Contraseñas	
¿UD maneja contraseñas de algún tipo?	☐ Si, ☐ No
Mencione los componentes involucrados en el manejo de contraseñas	
Dispositivos	
¿UD instala algún “driver” de dispositivos?	☐ Si, ☐ No Por favor, mencione:
Cuentas y privilegios	
¿UD crea cuentas personalizadas?	☐ Si, ☐ No
Si es así, ¿Cuáles cuentas y para qué se utilizan?	
¿UD requiere o permite sesiones NULL (conexiones anónimas o desconocidas)?	☐ Si, ☐ No Explique:
¿UD requiere de privilegios especiales para ejecutar la aplicación?	☐ Si, ☐ No
Si es así, ¿Cuáles privilegios y por qué?	
HTML	
¿UD utiliza HTML para cualquier tipo de UI?	☐ Si, ☐ No
¿Existe la posibilidad de que la información insertada por un usuario sea mostrada UI HTML por otro usuario?	☐ Si, ☐ No
Si es así, ¿cómo controla la	

información contaminada para asegurarse de que ningún código HTML puede ser insertado en el UI?	
Bases de Datos	
¿UD utiliza una base de datos?	☐ Si, ☐ No
¿Cuál?	☐ SQL Server, ☐ Otros:
¿UD realiza consultas incluyendo directamente los campos de texto obtenidos de la información entrada por los usuarios?	☐ Si, ☐ No Mencione:
Si es así, ¿Cómo se corrige o chequea las entradas del usuario para asegurarse de que no contenga comandos de SQL?	
¿UD utiliza o crea procedimientos almacenados?	☐ Si, ☐ No Mencione:
Documentación	
¿UD tiene prácticas seguras de documentación?	☐ Si, ☐ No
Redes	
Usan escucha en algún socket?	☐ Si, ☐ No, Mencione:
¿UD usa o se basa en tráfico de multidifusión?	☐ Si, ☐ No
¿Su componente soporta IPv6?	☐ Si, ☐ No
¿UD utiliza HTTP.SYS?	☐ Si, ☐ No