GUIA DE BUENAS PRÁCTICAS BASADAS EN LAS METODOLOGIAS SDL, CbyC PARA DESARROLLO DE SOFTWARE SEGURO APOYADO EN LINEAMIENTOS DE LA FUNDACION OWASP Y ESTANDARES DE LA DIVISION CERT DEL INSTITUTO SEI DE LA UNIVERSIDAD CARNEGIE MELLON.

ASDRUBAL GUAYARA RUBIO

UNIVERSIDAD NACIONAL ABIERTA Y A DISTANCIA UNAD
ESCUELA DE CIENCIAS BÁSICAS E INGENIERÍA
ESPECIALIZACIÓN EN SEGURIDAD INFORMÁTICA
CURSO: PROYECTO DE SEGURIDAD INFORMATICA
ABRIL 2018

GUIA DE BUENAS PRÁCTICAS BASADAS EN LAS METODOLOGIAS SDL, CbyC PARA DESARROLLO DE SOFTWARE SEGURO APOYADO EN LINEAMIENTOS DE LA FUNDACION OWASP Y ESTANDARES DE LA DIVISION CERT DEL INSTITUTO SEI DE LA UNIVERSIDAD CARNEGIE MELLON

ASDRUBAL GUAYARA RUBIO

Monografía para optar por el título de Especialista en Seguridad Informática

MG. LUIS FERNANDO ZAMBRANO

UNIVERSIDAD NACIONAL ABIERTA Y A DISTANCIA UNAD
ESCUELA DE CIENCIAS BÁSICAS E INGENIERÍA
ESPECIALIZACIÓN EN SEGURIDAD INFORMÁTICA
ABRIL 2018

Nota de Aceptación

_____

_____

_____

_____

Presidente del Jurado

_____

Jurado

_____

Jurado

CONTENIDO

LISTA DE IMAGENES

1. INTRODUCCION

El incremento de los ataques en gran parte obedece a las vulnerabilidades de seguridad que se presentan en los programas o software que soportan los sistemas de información o aplicaciones Web.

Independiente de la plataforma o Sistema operativo que se utilice o el lenguaje de programación que se empleado llámese C, C++, C#, JAVA, ASP, Visual Basic, .NET, Python o Perl para el desarrollo de las aplicaciones o acceso a Bases de Datos como Oracle, DB2, SQL Server, todos son susceptibles de presentar defectos de código que generen vulnerabilidades de seguridad en los sistema e información o productos de software o acceso a base de datos.

Teniendo en cuenta lo mencionado anteriormente, este proyecto de monografía está alineado a la necesidad de la comunidad de desarrolladores, de disponer de una guía que incluye mejores prácticas y recomendaciones de seguridad dentro del ciclo de desarrollo de software y para prevención de las vulnerabilidades más frecuentes.

Las metodologías SDL, CbyC, Fundación OWASP y la metodología de la división CERT de la Carnegie Mellon University Engineering Institute, fortalecen la construcción del software a nivel de seguridad. Adicionalmente en esta monografía se realiza un análisis de alto nivel a cerca de vulnerabilidades de mayor impacto y más frecuentes, así como algunas recomendaciones para evitarlas. Dentro de las vulnerabilidades de mayor frecuencia podemos tener: desbordamiento de enteros, errores de formato, inyección sql, Script sitios cruzado, desbordamiento de buffer, uso inapropiado de SSL y TLS, Perdida de autenticación, uso de componentes con vulnerabilidades conocidas, exposición de datos sensibles, entre otros.

2. TITULO

GUIA DE BUENAS PRÁCTICAS BASADAS EN LAS METODOLOGIAS SDL, CbyC PARA DESARROLLO DE SOFTWARE SEGURO APOYADO EN LINEAMIENTOS DE LA FUNDACION OWASP Y ESTANDARES DE LA DIVISION CERT DEL INSTITUTO SEI DE LA UNIVERSIDAD CARNEGIE MELLON.

3. DESCRIPCION DEL PROBLEMA

La gran mayoría de las funcionalidades implementadas en los sistemas de información empresariales, de escritorio y las que están expuestas en la Web, se basan en Software; por tal razón el software es el nuevo centro de atracción para los criminales. En la actualidad se viene presentado de manera alarmante un incremento en los ataques o accesos no autorizados a los sistemas de información, generando en las empresas pérdidas millonarias y dejando un ambiente de inestabilidad en el sector que representan cada una de ellas, es decir, dejan a las organizaciones gravemente expuestas perdiendo la credibilidad de sus clientes y llegando incluso a poner en duda su continuidad.

Aunque no lo percibimos pero cualquier ataque que impacte los sistemas de información por ejemplo los sistemas de un aeropuerto, tiene una repercusión enorme en las personas porque como casi todo está interconectado y todas las actividades comerciales , financieras y de otra índole dependen en gran medida de los aplicativos y sistemas de información de tal manera que una falla en los sistemas puede originar desde la no prestación de servicios de índole , social, financiero, de salud , gubernamental entre otros; todo esto en detrimento de las empresas y traduciéndose en perdida de dinero, robo de información de clientes o información financiera o privada.

Para mencionar un ejemplo de perdidas generado por fallas en el software se puede evidenciar en el artículo del diario el País de España, sobre 3 incidentes ocurridos casi en simultánea en los Estados Unidos en el año 2015 y básicamente fueron: más de 3000 vuelos de American Airlines quedaron paralizados por un fallo informático, así mismo la bolsa de New York por similares circunstancias suspendió operaciones por varias horas y por último el portal del periódico digital Wall Streat Journal quedo indisponible.[1]

Para todos estos fallos el diagnóstico fue “fallos informáticos” es decir errores en el código.

[1] Marimar Jimenez.(Julio 2015).Fallos informáticos que hacen temblar la economía. {En Linea}. Recuperado de: https://cincodias.elpais.com/cincodias/2015/07/09/tecnologia/1436467530_190970.html

4. JUSTIFICACION

En la actualidad el desarrollo de software seguro es la base de la seguridad en los sistemas de Información en las empresas, por tal razón y dado que no se le ha dado la relevancia que debería, es necesario e importante el desarrollo del proyecto propuesto sobre este tema.

En los entornos de la modernidad la defensa a los ataques maliciosos debe ser una prioridad para los desarrolladores de software y solo se garantiza que los sistemas estén blindados ante cualquier ataque. Es por eso que las metodologías para el desarrollo de software han incorporado los componentes de seguridad para dar batalla y lucha contra los ataques de los hackers.

Este proyecto de monografía busca coadyuvar a que se fortalezca el desarrollo de software seguro a través de un análisis que ofrecen las metodologías SDL, CbyC, Fundación OWASP y la metodología de la división CERT de la Carnegie Mellon University Engineering Institute, así como un análisis de alto nivel y para las vulnerabilidades de mayor impacto y más frecuentes y algunas recomendaciones para su prevención.

5. FORMULACION DEL PROBLEMA

¿Cómo se puede disminuir las vulnerabilidades de los sistemas de información implementando las mejores prácticas y estándares de codificación segura en el desarrollo de software?

6. OBJETIVOS

6.1. OBJETIVO GENERAL

Generar una guía de buenas prácticas basadas en las metodologías SDL, CbyC para producir Software seguro, Apoyados en lineamientos de la Fundación OWASP y de la división CERT de la Carnegie Mellon University Engineering Institute, para minimizar vulnerabilidades de seguridad en los sistemas de información.

6.2. OBJETIVOS ESPECIFICOS

- Recolectar información que permita determinar el estado actual de la construcción de software seguro teniendo como base las metodologías recomendadas por OWASP foundation, metodología SDL, Correctness by Contruction CbyC y Estándares de da división CERT del Instituto SEI (Software Engineering Institute) de la universidad Carnegie Mellon.
- Diagnosticar e Identificar las vulnerabilidades de seguridad más frecuentes en el desarrollo Software.
- Realizar las Recomendaciones de Buenas Prácticas para Evitar Errores y Vulnerabilidad de Seguridad en el Software.

7. POSIBLES COLABORADORES

Para la realización de este proyecto se contará con la participación de Ingenieros de sistemas Colegas, Ingenieros de sistemas y Electrónicos compañeros de Trabajo e ingenieros de sistemas expertos en herramientas de desarrollo del software.

8. RECURSOS DISPONIBLES

8.1. RECURSO HUMANO

El proyecto será de responsabilidad de la persona que aparece como el autor del proyecto, adicionalmente se va a requerir asertoria de los ingenieros que apoyan el proyecto en tutoría y dirección del mismo en la universidad Nacional Abierta y A Distancia.
También es posible que se tenga la necesidad de asertoria/consultoría y aportes de ingenieros colegas expertos en el tema de desarrollo de software.

8.2. RECURSOS FISICOS Y TECNOLOGICOS.

Los recursos con los que se cuentan para el desarrollo de esta monografía, serian:

- Libros especializados
- Revistas Técnicas
- Consultas en la Web.
- Portátil Dell Inspiron Core i7
- Portátil Lenovo Core i7

9. MARCO DE REFERENCIA

9.1. MARCO TEORICO

Con este proyecto se pretende enmarcar las buenas prácticas del desarrollo de software seguro dentro del contexto de la seguridad de la información. Debemos tener en cuenta muchos conceptos que son importantes en el proceso de alcanzar el óptimo punto de calidad y disminución de fallas del software.

El desarrollo de software es particularmente especial dado que para la comunidad de desarrolladora es muy claro que se deben evitar cometer errores en la construcción del software, pero es muy fácil cometer uno y el impacto que puede causar a los resultados inesperados o en caso de ser una debilidad que seguramente será aprovechada por los hackers.

- “19 puntos críticos sobre seguridad de software: fallas de programación y como corregirlas, Howard Michael, Leblanc David “, En esta bibliografía el autor Michael Howard hace una referencia muy importante sobre el paradigma de la seguridad en el desarrollo de software, en cual se dejan evidenciados toda la argumentación que sostiene que, en definitivamente trabajar por la calidad y seguridad del software, reduce las probabilidades de fallas de los programas y sistemas de información. [2]

9.1.1. Metodología SDL.

Es una metodología propietaria de Microsoft implementada desde mediados de la década del 2000, para prevenir ataques de hackers y piratas informáticos y así mejorar la seguridad en el desarrollo de software. Tiene como base la etapa de entrenamiento a cerca de los temas de seguridad así como de análisis estático y dinámico, con testing para identificar incidentes. El principal elemento de esta metodología es que permite tiene un control más eficaz para amenazas en secciones de código fuente donde se vea impactado por posibles ataques mal intencionados. Se basa en una metodología con diferentes fases donde se pueden

[2] HOWARD, M y LEBLANC, D. 19 puntos críticos sobre seguridad de software. Editorial: McGraw-Hill Interamericana, enero 2007.

implementar los controles de seguridad en la codificación del código fuente. Se realizan entre otras actividades: simulación de diseño de modelo de una posible amenaza en la etapa de diseño del software, se explota el código, se realizan pruebas de seguridad y antes de liberarlo a producción el producto debe ser sometido a una revisión final para minimizar la posibilidad de vulnerabilidades.
Entre las principales fases de la metodología tenemos:
Entrenamiento
Requerimiento
Diseño
Implementación
Verificación
Lanzamiento
Respuesta.

9.1.2. Metodología CbyC.

Metodología CbyC, Traduce Correctness by Construccion, este método tiene requiere un nivel crítico de seguridad, con errores disminuidos al cero los errores y una flexibilidad para los controles de cambio.

Por su flexibilidad es apropiado para manejar requerimiento que exijan un nivel de seguridad crítico. Existen empresas muy productivas al implementar esta metodología, con tasas de errores muy bajos en relación con la producción de líneas de código y si son introducidos que se eliminen lo antes posible.

Las Fases de la Metodología CbyC son:
Fase de Requerimientos
Diseño de Alto Nivel
Especificación del software
Especificación de pruebas
Diseño Detallado
Especificación del módulo
Código
Construcción del Software.

9.1.3. Proyecto OWASP.

El proyecto OWASP, tiene como finalidad de dar lineamientos a las organizaciones para que desarrollen y mantengan software o aplicaciones confiables y seguros. En el proyecto OWASP se identifican 10 principales riesgos de seguridad en aplicaciones WEB como, por ejemplo: inyección, Secuencia de comando de sitios cruzados, Perdida de autenticación y gestión de sesiones. Por naturaleza es de código abierto y enfocado básicamente a las aplicaciones WEB y tiene colaboradores en todo el mundo los cuales participan con la documentación y el desarrollo del mismo. OWASP[3] es básicamente un proyecto para facilitar que los desarrollos sean más seguros, basándose en la colaboración de los miembros a nivel mundial generando documentación y herramientas sobre la seguridad de los sistemas de información. [4] Por otro lado cuenta con un proyecto denominado OWASP ASVS, es un proyecto de verificación de seguridad de las aplicaciones cuyo objetivo más importante es la revisión de seguridad del código de aplicativos WEB utilizando estándares vigentes en el sector.

OWASP tiene los siguientes proyectos vigentes, los cuales están fuera del alcance de este trabajo:
Top ten: Fallos de seguridad más frecuentes.
WebGoat: Programa que muestra vulnerabilidades
ESAPI: Librería de código abierto útil para seguridad de los desarrollos.
AntiSamy: Librería que no permite que se le inyecte código externo o malicioso.
Testing Project: guía de metodología de pruebas.

9.1.4. Normas division CERT software engineering institute.

La división CERT hace parte del instituto de ingeniería de software de la universidad Carnegie Mellon. Es una entidad que abandera la lucha para mejorar la seguridad de los sistemas informáticos y ciberseguridad.

[3] Owasp. (agosto 2017) The open Web Aplication Security Project, Los 10 riesgos más críticos en aplicaciones Web. {En línea}. Recuperado de: http://www.owasp.org/index.php/Top_10

[4] Owasp. (agosto 2017) OWASP. {En línea}. Recuperado de: https://www.owasp.org/index.php/Main_Page

La guía CERT[5] divulga de manera sincronizada las vulnerabilidades de los sistemas. La guía presenta la conceptualización y definiciones para realizar la divulgación de las vulnerabilidades. A través de CVD (Coordinate Vulnerability Disclosure) reúnen las vulnerabilidades que arrojan los buscadores de vulnerabilidades y coordinadamente hace su divulgación.

También la división CERT publica y promueve estándares seguros para la codificación de software para evitar los huecos de información que eventualmente puedan generase en los sistemas. Estos estándares se tienen considerados para los diferentes lenguajes de programación, por ejemplo: SEI CERT C Coding Standar, SEI CERT C++ Coding Standar, SEI CERT Oracle Coding Estándar for Java, SEI CERT Perl Coding Estándar.

9.2. MARCO DE ANTECEDENTES

Para el desarrollo de este proyecto monografía, se apoyarán en los proyectos que describo a continuación y en los lineamientos de la fundación OWASP y los siguientes Estudios:

"Metodologías para el desarrollo de software seguro, Presentado por Ferrán López Proveció". [6] Con este proyecto de la facultad de informática de Barcelona, el ingeniero Ferrán López busca minimizar los riesgos y las debilidades que existen entre las integraciones a nivel de red y de software que se existen entre la comunicación de todos los proyectos desarrollado entre las diferentes facultades de la Universidad UPC de España.

"Hacia la ingeniería de software Seguro, Presentado por Martha Castellaro[7], Susana Romani – Facultad regional de Santafé Universidad tecnológica Nacional".
Este estudio centra su enfoque en la integración entre la ingeniería de software y la ingeniería de Seguridad, considerando la seguridad como parte esencial de los ciclos de desarrollo de software.

---

[5] MCMANUS, José.(noviembre 2017). SEI CERT Oracle Coding Standard for Java. Recuperado de https://www.securecoding.cert.org/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java.

[6] FERRAN, Provencio. (enero 2015). Metodología para el desarrollo de software seguro, recuperado de: https://upcommons.upc.edu/bitstream/handle/2099.1/24902/103275.pdf

[7] CASTELLARO, M y ROMANI, S. (2009). Hacia la ingeniería de software seguro. Recuperado de: http://sedici.unlp.edu.ar/bitstream/handle/10915/21332/Documento_completo.pdf?sequence=1

"Guía para el desarrollo de software Seguro, Matei Adriana (2015), Boadilla del monte Madrid". Este trabajo de grado de la ingeniera Matei, se basa en la necesidad de dar una relevancia mayor al software para la realización de trabajos críticos, y por ende la seguridad del software toma un valor preponderante en todos los ámbitos que involucran manejo de software. Su estudio se basa en información de CERT, CMMI entre otros.[8]

"CEI CERT Coding Standard, Normas y Estándares para el desarrollo de Software Seguro" [9], este Standard Proporciona reglas para la codificación segura para diferentes lenguajes de programación como C y JAVA.

"Marco de Referencia de arquitectura empresarial para la gestión de las tecnologías de la información."[10] , este Marco de referencia es un modelo para las entidades estatales colombianas, se fundamenta en componentes básicos: Principios, Dominios y Bases de conocimientos.

9.3. MARCO LEGAL

Para el desarrollo de este proyecto se debe tener como base la normatividad establecida en Colombia para el desarrollo de software como son: "Manual de Gobierno en línea" y "Marco de Referencia de arquitectura empresarial para la gestión de las tecnologías de la información".

Otro Marco Legal que tenemos es la Política Nacional de Seguridad Digital de Colombia, descrita en el documento COMPES 3854 de 2016, donde se definen las políticas estrategias y recomendaciones del gobierno para que se implementen mejores políticas para contrarrestar los incidentes digitales.

---

[8] Matei Adriana. (2015).Guia para desarrollo de software seguro.{En línea}.Recuperado de: http://oa.upm.es/34770/1/PFC_ADRIANA_MATEI.pdf

[9] Mcmanus José. (Noviembre 2017). SEI CERT Oracle Coding Standard for Java.{En Linea} Recuperado de: https://www.securecoding.cert.org/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java

[10] MINTIC.(2018). Marco de Referencia Para la gestión de las tecnologías de la información. { En línea}.Recuperado de: http://www.mintic.gov.co/arquitecturati/630/w3-channel.html

La seguridad en el software es uno de los temas fundamentales contra todo intento de vulnerar los sistemas de información que están penalizados por el gobierno colombiano por medio de la ley 1273 de 2009[11] :

“CAPITULO PRIMERO De los atentados contra la confidencialidad, la integridad y la disponibilidad de los datos y de los sistemas informáticos.”
“Artículo 269A. ACCESO ABUSIVO A UN SISTEMA INFORMÁTICO. El que, sin autorización o por fuera de lo acordado, acceda en todo o en parte a un sistema informático protegido o no con una medida de seguridad, o se mantenga dentro del mismo en contra de la voluntad de quien tenga el legítimo derecho a excluirlo, incurrirá en pena de prisión de cuarenta y ocho (48) a noventa y seis (96) meses y en multa de 100 a 1000 salarios mínimos legales mensuales vigentes.”
“Artículo 269B. OBSTACULIZACIÓN ILEGÍTIMA DE SISTEMA INFORMÁTICO O RED DE TELECOMUNICACIÓN. El que, sin estar facultado para ello, impida u obstaculice el funcionamiento o el acceso normal a un sistema informático, a los datos informáticos allí contenidos, o a una red de telecomunicaciones, incurrirá en pena de prisión de cuarenta y ocho (48) a noventa y seis (96) meses y en multa de 100 a 1000 salarios mínimos legales mensuales vigentes, siempre que la conducta no constituya delito sancionado con una pena mayor.”
“Artículo 269C. INTERCEPTACIÓN DE DATOS INFORMÁTICOS. El que, sin orden judicial previa intercepte datos informáticos en su origen, destino o en el interior de un sistema informático, o las emisiones electromagnéticas provenientes de un sistema informático que los trasporte incurrirá en pena de prisión de treinta y seis (36) a setenta y dos (72) meses.” [12]

[11] Diario Oficial. (Enero 2009). Ley 1279 de 2009. {En línea}. Recuperado de: http://www.sic.gov.co/sites/default/files/normatividad/Ley_1273_2009.pdf.
[12] Ibid., Pag. 1.

10. MARCO METODOLOGICO

Este proyecto se llevará a buen término utilizando Análisis Documental, Análisis de contenido, incluyendo un análisis cualitativo en cuanto al estado actual del nivel de seguridad en el desarrollo del software.

10.1. TIPO DE INVESTIGACION

10.1.1. Investigación documental

Para el desarrollo de este proyecto se va a utilizar “Tipo Investigación Documental” dado que se va a profundizar sobre el estado del arte de la información relacionado con las modelos de desarrollo de software seguro y a partir de estas fuentes y aplicando un proceso de intelecto generar una guía de las mejores prácticas basadas en las metodologías para desarrollo de software.

En el análisis documental se tiene un proceso de comunicación el cual recupera o recopila la información, un proceso que transforma a un documento intermedio y luego genera un documento definitivo donde la información se interpreta y sintetiza.

Adicionalmente y teniendo como base los objetivos planteados en este proyecto podemos determinar que la modalidad de la investigación que se llevara a cabo va llevar a cabo dentro de este trabajo es investigación de Modalidad Básica, puesto que se va a investigar y recopilar información sobre modelos de codificación o desarrollo de software seguro.

10.1.2. Análisis de contenido

Se recopilará de diferentes fuentes como documentos, blogs, datos estadísticos y experiencias personales y de colegas y autores sobre el tema que nos ocupa en el proyecto. Y de acuerdo a las fuentes podemos recurrir al análisis Bibliográfico.

10.1.3. Investigación bibliográfica

A través de ella se recopilará la mayor cantidad de información relevante para el proyecto y que sirva conocer el estado del arte y que sirva de insumo para la construcción del proyecto.
Se tomarán como base estudios anteriores y casos de uso que se han llevado a cabo en el ámbito del desarrollo de software seguro.

10.2. TECNICAS DE ANALISIS Y PROCESAMIENTO DE DATOS

10.2.1. Metodología cuantitativa

A través de recolección de datos estadísticos relacionado con las fallas productos de los vacíos de seguridad del software, podemos construir una métrica adecuada para la medir cuando un sistema se encuentra en criticidad y debe ser intervenido con los lineamientos para el desarrollo de software seguro.
La idea de tener ese método es complementarse con el método cualitativo y poder llegar a hacer generalizaciones sobre el tema y proyecto que se trata en este documento.
Intentaremos identificar leyes generales, apoyados en los datos estadísticos y experiencia de los expertos en el tema.

Dentro de las técnicas de recolección de datos lo que vamos a utilizar es la recolección de información de datos secundarios encaminados a recopilar la mayor cantidad de información que se tienen sobre aplicabilidad de estándares de codificación seguro de software experiencias de los desarrolladores sobre este tema.

De acuerdo a la percepción que demuestran las organizaciones sobre el nivel de seguridad que perciben o sienten tienen sus sistemas, se puede identificar el nivel de seguridad que tiene el software y si es necesario la intervención del mismo.

También se tomarán como referencia las experiencias propias en el área de desarrollo de software y la de colegas que están en el mercado.

10.3. METODOLGIA DE DESARROLLO

Para la elaboración de la guía de las buenas prácticas para el desarrollo de software, se definen por fases dentro del proyecto que se describen a continuación:

10.3.1. Recolección y documentación.

Dentro de esta etapa se levantará todo el inventario mediante la investigación, consulta recopilación de las buenas prácticas para el desarrollo de software de las fuentes descritas anteriormente en el capítulo 18 y toda la bibliografía propuesta.

10.3.2. Consultas y recolección de Datos Secundarios.

Se realizará la recolección de datos secundarios sobre la aplicación de estándares de codificación seguro de software cuestionario de expertos y autores especialistas en el tema de desarrollo de software seguro, el cual será insumo y referencia dentro de la construcción del documento.

10.3.3. Análisis y clasificación de la información

Una vez que se tenga el inventario de todo el material, se procederá a clasificar las buenas prácticas dentro del contexto de la seguridad de la construcción del software seguro.

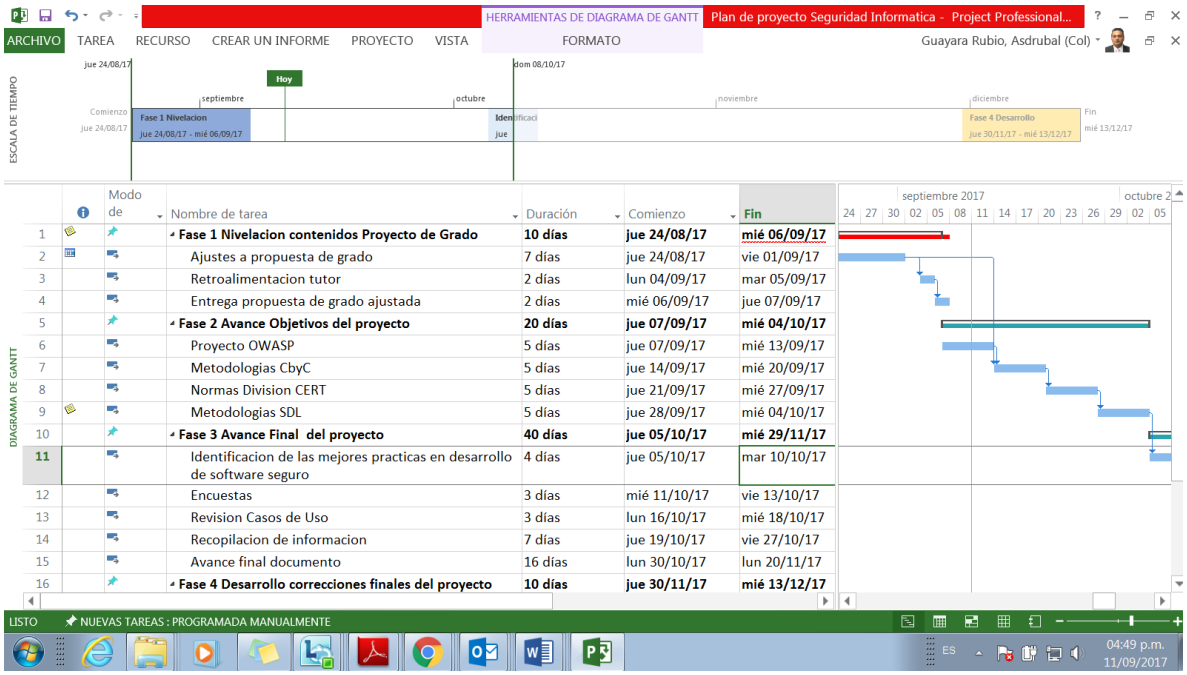
10.3.4. Construcción de la Guía.

Tomando como referencia la información clasificada, se inicia la documentación de los la información apoyados en algunos casos de uso que tienen que ver con buenas prácticas y estándares para codificación de software seguro.

11. CRONOGRAMA

Fig 1. Cronograma Proyecto de Grado Asdrubal Guayara

Figura 1. Cronograma Proyecto de Grado Asdrubal Guayara



Fuente: El autor

12. DIAGNOSTICO DE LOS PRINCIPALES PROBLEMAS DE SEGURIDAD EN EL DESARROLLO DE SOFTWARE.

Es un hecho que cada día más los sistemas de información se ven afectados por los innumerables ataques de hackers y hace que las organizaciones se vean abocados a implementar mayores controles a los accesos a los sistemas.
El autor David Leblanc[13] dice “La división de seguridad Nacional del ciberespacio del Departamento de seguridad Nacional de Estados Unidos hizo alusión que el 95% de los errores de seguridad de software proviene de 19 errores comunes de programación”

En el ámbito nacional tenemos un estudio de la donde se hace evidente que en las fábricas de desarrollo de software no se tiene incluidos controles de seguridad por ejemplo en un estudio adelantado por estudiantes de la UNIMAR:
“… se concluye con una preocupante cifra cuyo promedio de evaluación es de 0+, interpretado como un mínimo y muy limitado uso de prácticas de seguridad en construcción, en cuanto a la práctica de evaluación de la amenaza, ninguna empresa (0 %) compara ni clasifica las amenazas con los mecanismos de protección y control, lo cual indica que aun antes de desarrollar el software, ya están dejando brechas abiertas para que se presenten más adelante incidentes de seguridad” [14]

Otra falencia que se observa es la baja penetración del tema de capacitación de los profesionales en técnicas para implementar seguridad a nivel del desarrollo del software:

“Una práctica de seguridad que preocupa por su baja calificación es la de formación y orientación perteneciente al aspecto de gobierno, obteniendo una calificación de 0+, indicando que hay un muy escaso conocimiento o uso esta práctica, solo el 12.5 % de las empresas entrenan a sus desarrolladores en concientización sobre seguridad y lo han corroborado con una evaluación para comprobar dichas

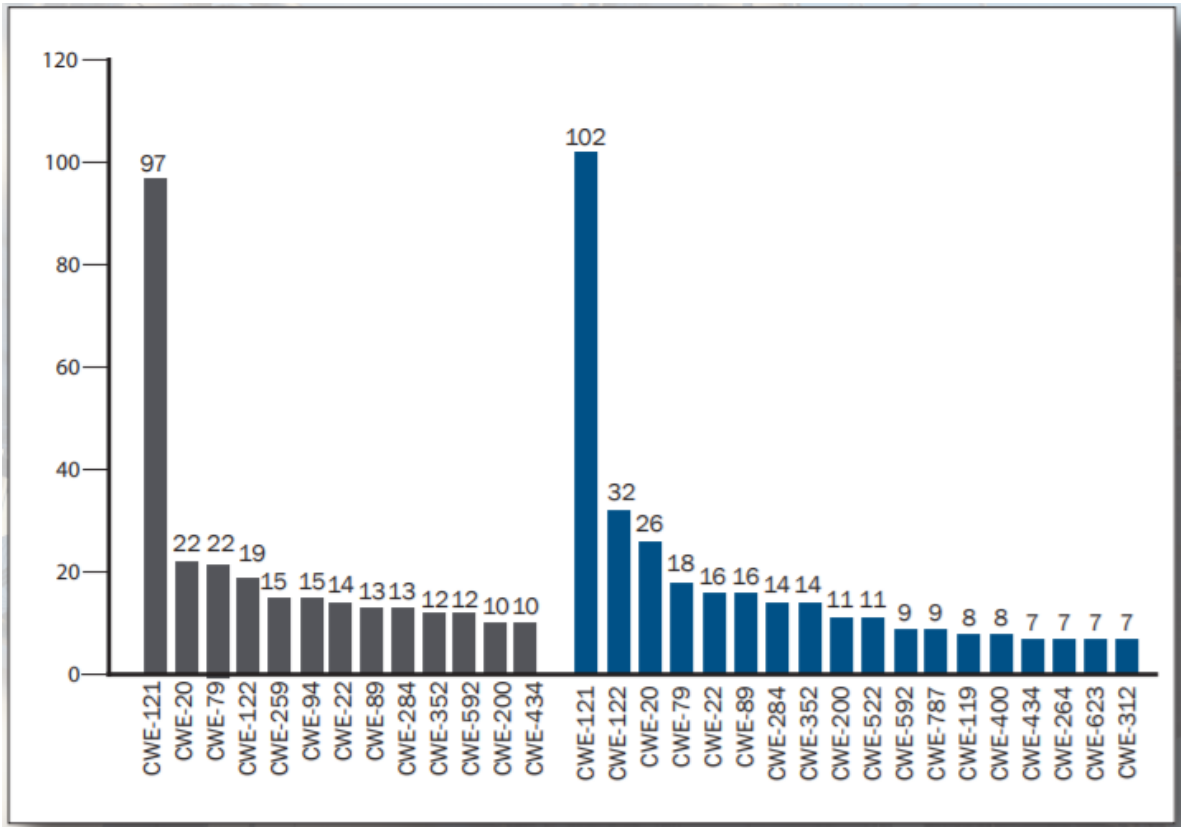
[13] Howard Michael y Leblanc David.(2007). 19 puntos críticos sobre seguridad de software Editorial: McGraw-Hill Interamericana, enero 2007. Pag. 19.

[14]LORA,J y DELGADO W. (2016) Diagnóstico de uso de normativas de seguridad en fábricas de construcción de software en la ciudad de pasto{en línea}. Recuperado de:http://www.umariana.edu.co/ojs-editorial/index.php/libroseditorialunimar/article/view/980/902

habilidades, se destaca que el 62 % de las empresas entrena y orienta a los diferentes roles en el proceso de desarrollo” [15]

En cuanto a fallas por vulnerabilidades: según el reporte de ICS-CERT del año 2016, indica la forma en que se tipificaron las vulnerabilidades que se presentaron con mayor frecuencia, tal como lo indica el grafico de la Figura 2, donde las principales vulnerabilidades fueron: CWE79: Cross Site Scripting, CWE121 corresponde a desbordamiento de buffer basado en overflow, CWE20: Error de entrada, CWE122 corresponde a desbordamiento de buffer basado en pila[16].

Figura 2 Most frequently assigned CWEs assigned by ICS-CERT in FY and CY 2016



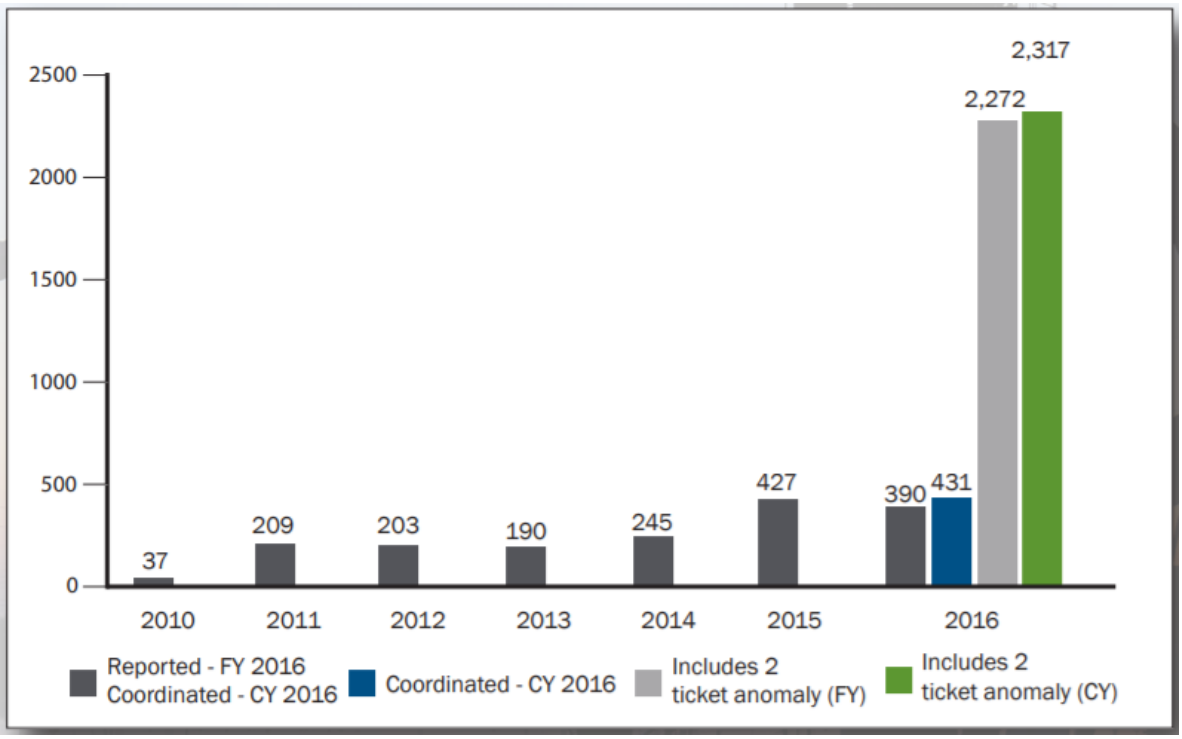
Fuente:https://ics-cert.us-cert.gov/sites/default/files/Annual_Reports/NCCIC_ICS-CERT_FY%202016_Annual_Vulnerability_Coordination_Report.pdf

[15] LORA,J y DELGADO W. (2016) Diagnóstico de uso de normativas de seguridad en fábricas de construcción de software en la ciudad de pasto{en línea}. Recuperado de:http://www.umariana.edu.co/ojs-editorial/index.php/libroseditorialunimar/article/view/980/902.

[16]ICS-CERT.(2016). Annual Vulnerability Coordination Report.{En linea}.Recuperado de: https://ics-cert.us-cert.gov/sites/default/files/Annual_Reports/NCCIC_ICS-CERT_FY%202016_Annual_Vulnerability_Coordination_Report.pdf

De igual manera en ese mismo reporte y tal lo que indica el grafico de la Figura 3 generado por el reporte de ICS-CERT, muestra el comportamiento ascendente de las vulnerabilidades desde el año 2010 hasta el año 2016.[17]
Este comportamiento evidencia una notoria falencia en la calidad del desarrollo del software y ausencia de controles para evitar los huecos de seguridad en el mismo.

Figura 3. Vulnerabilities reported to ICS-CERT since FY 2010.



Fuente: https://ics-cert.us-cert.gov/sites/default/files/Annual_Reports/NCCIC_ICS-CERT_FY%202016_Annual_Vulnerability_Coordination_Report.pdf

12.1. FALLA DESBORDAMIENTO DE ENTEROS

Desde siempre históricamente los errores originados por punto flotante han sido una constante desde las primeras versiones de programas de computadora.
El origen del problema se da básicamente con las operaciones binarias dado que no siempre el resultado es el esperado.

[17] ICS-CERT.(2016). Annual Vulnerability Coordination Report.{En linea}.Recuperado de: https://ics-cert.us-cert.gov/sites/default/files/Annual_Reports/NCCIC_ICS-CERT_FY%202016_Annual_Vulnerability_Coordination_Report.pdf

Algunos lenguajes han introducido tipos de punto como “Variant” es el caso de las versiones más resientes de Visual Basic, con el ánimo de solucionar estas falencias.

La mayoría de los lenguajes son afectados por esta situación. Unos errores generados por desbordamiento de enteros pueden generar mucho daño en el sistema incluso errores de seguridad manifestados en obtención de privilegios sobre el mismo y ejecuciones de código que pueden llegar a afectar la disponibilidad del sistema. En general en cualquier sistema de información que realice cálculos matemáticos con entradas que introduzcan los usuarios y que no sean validadas, son susceptibles a experimentar esta falla. Algunas de las recomendaciones que se dan para tratar de minimizar este problema es la utilización de datos numéricos sin signo cuando no sea estrictamente necesario. [18]

12.2. USO INAPROPIADO DE SSL Y TLS.

Algunos programadores usan en sus desarrollos para implementar seguridad en red, utilizan protocolos de conexiones de rede seguras (en vez de Web) reemplazando el protocolo TCP por la utilización SSL (capa segura de conectores) (Secure Socket Layer), junto a la capa de transporte TLS (Transport Layer Security). Esto puede genera problemas serios en la autenticación dado que un hacker puede intervenir, interceptar o introducirse en alguna conversación y no ser detectado o acceder a través de certificado confiable o un certificado robado por ellos. Se recomienda que la autenticación se realice adecuadamente en el servidor y en lo posible realizar validación del certificado, es decir que sea una validación minuciosa campo por campo, en especial el DN y AltName de tipo DNSname. Se recomienda en lo posible el uso de HTTPS.[19]

12.3. FALLA EN EL MANEJO DE ERRORES.

Las excepciones o condiciones de error son fundamentales cuando se quiere dar un buen soporte al software y facilitarle las cosas al usuario, sin embargo, si este manejo no se realiza de manera adecuada, puede generarse resultados

[18] Howard Michael y Leblanc David.(enero 2007). 19 puntos críticos sobre seguridad de software Editorial: McGraw-Hill Interamericana. p. 17.
[19] Ibid., Pag. 128.

inesperados e inclusive generar una vulnerabilidad en el software, como por ejemplo denegación de servicio. En Los lenguajes de programación como Java y cuando se manejan de manera inadecuada el control de errores, el programa puede abortar generando esto una denegación de servicio que a su vez se convierte en un problema de seguridad. Hay que tener en cuenta que se puede presentar esta situación si el programador omite errores, da mala interpretación a los errores, manejo incorrecto de excepciones.[20] Por otro lado el exceso en el manejo de excepciones tanto las conocidas como las no conocidas, por ejemplo en Java o .net utilizando try { ….. } catch (Excepción) {manejo de error por cualquier excepción..} …. Pueden llegar a enmascarar u ocultar un error del software, el cual puede ser muy difícil de detectar o depurar.[21]

12.4. ERRORES DE FORMATO

Esta vulnerabilidad del software, se presenta cuando el programador deja sin validar las entradas que el usuario diligencia por ejemplo en un front-end, esto puede llegar a generar una vulnerabilidad que el atacante puede aprovechar para escribir en memoria. Esto es un poco sofisticado pero el hacker conoce que puede evadir las protecciones de pila o utilizar las direcciones de memoria de acuerdo a su necesidad. En ambientes como Linux, también se puede presentar cuando las cadenas se utilizan desde una ubicación que no es confiable. Debemos recordar que en el sistema operativo Windows maneja las cadenas ya se en el ejecutable o en las librerías DLLs. El error se origina porque básicamente el programador utiliza las cadenas que el usuario introduce y las cuales no son validadas adecuadamente por el software; los lenguajes más vulnerables son C y C++. Se sugiere en lo posible nunca asignar las entradas de los usuarios a operación de formateo.[22]

12.5. INYECCIÓN DE SQL.

Actualmente es una deficiencia muy habitual e igualmente preocupante puesto que muchos de los sistemas que la padecen manejan información sensible de clientes y su información confidencial los cuales normalmente están en base de datos.

---

[20] Howard Michael y Leblanc David.(enero 2007). 19 puntos críticos sobre seguridad de software Editorial: McGraw-Hill Interamericana. p. 74.
[21] Ibid., Pag. 78.
[22] Howard Michael y Leblanc David.(enero 2007). 19 puntos críticos sobre seguridad de software Editorial: McGraw-Hill Interamericana. p. 23.

Las empresas que manejan información confidencial de clientes, es de obligatorio cumplimento la implementación de controles de acceso eficaces. Sin distingo de lenguaje que se utilice para acceder a las bases de datos, todos son susceptibles de este tipo de ataques. Los atacantes utilizan la conformación de consultas aprovechando que en los sistemas las Consultas sql se diseñan en Concatenación de cadena; otra criticidad son los procedimientos almacenados, los cuales manejan parámetros de entrada. La recomendación a los desarrolladores es que utilicen otro método diferente a la concatenación, por ejemplo, utilizar instrucciones sql con parámetros; adicionalmente se debe validar la entrada utilizando expresiones regulares y en lo posible se debe resguardar la conexión a la BD en una ubicación fuera del sistema de información, a modo de un archivo que tendrá la seguridad del registro en caso de contar con sistema operativo Windows. Muy importante no utilizar de manera similar a las comillas dobles y sencillas. Una recomendación final es suprimir el acceso directo a las tablas de la BD. por parte de los usuarios del sistema.[23]

12.6. DESBORDAMIENTO DE BUFFER

Este problema ocurre cuando dentro de un programa el desarrollador permite escribir fuera del rango que búfer que se asigna. Cabe destacar que el impacto de una debilidad de esta naturaleza pueda tumbar el sistema o puede el sistema quedar a merced del atacante. Dependiendo del ámbito de la aplicación que falla, si es un programa que atiende servicios de red, seguramente esto será provechado por ejemplo por un gusano informático, que en el pasado producto de un ataque de un gusano por problemas de desbordamiento de buffer casi Colapsa el internet a finales de los ochentas. Los lenguajes de programación con los cuales es más común cometer estos errores son C y C++. Los programas más recientes java, c# evitan el acceso directo al manejo de la memoria y hacen menos posible que ocurra desbordamiento.[24] Como sugerencia para tratar de mitigar un la aparición de esta vulnerabilidad se recomienda, tener actualizada la librería de versiones de las funciones que presentan mayor problema, como por ejemplo: strcpy,strcat, sprintf; adicionalmente revisar los finales de los ciclos antes de acceder a la pila. Otro punto importante es utilizar la implementación de pila que ofrecen varios fabricantes.[25]

[23] Howard Michael y Leblanc David.(enero 2007). 19 puntos críticos sobre seguridad de software Editorial: McGraw-Hill Interamericana. p. 55, 61.
[24] Ibid., Pag. 3.
[25] Ibid., Pag. 13,14.

12.7. SCRIPT EN SITIO CRUZADO (XSS).

Es un hueco de seguridad básicamente afecta a usuarios de sistemas web vulnerables, los cuales a través de la transferencia de script entre clientes o computadores que tienen acceso a web con vulnerabilidad, pueden comprometer la información privada o financiera de sus clientes. Normalmente los atacantes utilizan las Cookies para estos fraudes. Detallando un poco más esta afectación: a través de la entrada de usuario que puede ser una consulta a una página web, la aplicación Web recibe una petición que realiza el atacante, pero éste envía algo que el servidor web no esperaba y que en últimas es el ataque disfrazada en una consulta deformada. Este ataque también es conocido como “ataque xss”.Se debe tener precaución con los desarrollos WEB que requieren retroalimentación, dado que esta es la vulnerabilidad que los atacantes utilizan para que la cadena de datos que se ingresen, sean manipulados por los atacantes.[26]

13. ESTADO ACTUAL DE LA CONSTRUCCIÓN DE SOFTWARE.

En la actualidad se cuentan con muchas metodologías para la construcción de software seguro que buscan minimizar los errores en las aplicaciones de software y los daños de imagen y financieros que una vulnerabilidad genera a las empresas, como lo indica Joaquim Brito en su trabajo sobre metodologías de software seguro: “En una organización se pierden aproximadamente $6.6 millones de dólares por cada brecha de seguridad de TI, estas actividades ilícitas tienen ganancias de hasta 114 billones de dólares anualmente y 431 millones de víctimas anuales, es decir, 14 víctimas de cibercrimen cada segundo (Norton, 2012).”[27]

Dentro de esas metodologías ellas tenemos Correctness by Construction (CbyC), Security Development Lifecycle (SDL), Metodología, OWASP ASVS, las cuales serán abordadas en este trabajo.

---

[26] Ibid., Pag. 84.

[27] Brito Carlos Joaquín. (Diciembre 2013). Metodología para el software Seguro. {En Linea}. Recuperado de http://recibe.cucei.udg.mx/revista/es/vol2-no3/pdf/computacion05.pdf. Pag 2.

14. GUIA DE BUENAS PRÁCTICAS PARA CONSTRUCCION DE SOFTWARE SEGURO.

Las mejores prácticas para construir software seguro las ofrecen las siguientes metodologías: OWASP ASVS, metodología SDL, Metodología Correctness by Contruction CbyC y Estándares de da división CERT del Instituto SEI (Software Ingineering Institute) de la universidad Carnegie Mellon y sobre las cuales vamos a abordar a continuación en esta monografía; sin embargo, existen otras como SAMM y BSIMM que también tienen bastante aceptación, pero no son del alcance de este trabajo.

Según lo publicado por el NIST (National Institute of Standard and Tecnology), es más costoso corregir el código ya en producción, (en número que puede llegar a 30 veces), que identificarlo y corregirlo en etapas iniciales del desarrollo del software.[28]

Una vulnerabilidad se puede enmarcar como una falla de programación, configuración o diseño, que le facilita a un atacante intervenir un sistema de información para su lucro provecho o causar daño.

Dentro de las fallas de implementación se pueden incluir cross-site scripting, inyección sql, desbordamiento de buffer, formatos entre otros. Las fallas de diseño son utilizadas para perpetrar un ataque, ejemplo Telnet, el cual no tiene controles efectivos en un ambiente de ataques y por último los fallos de configuración, el ejemplo típico son las instalaciones por default, las cuales dejan funcionalidades que no se requieren y que pueden ser presa de los atacantes. [29]

[28] UNIVERSIDAD INTERNACIONAL DE LA RIOJA. (2017) El problema de la seguridad en el Software. Pág. 10.

[29] UNIVERSIDAD INTERNACIONAL DE LA RIOJA. (2017) El problema de la seguridad en el Software. Pag. 12.

15. METODOLOGIA SDL (SECURITY DEVELOPMENT LIFECYCLE).

Esta Metodología es de propiedad de Microsoft Corporación tiene como finalidad incluir controles de seguridad en todas las fases del desarrollo de software haciendo mayor énfasis en la privacidad.

El Objetivo final de SDL es a partir de la practicidad y tomando como un "todo" el proceso del desarrollo de software, disminuir la cantidad y la calidad de las vulnerabilidades de los productos de software (inicialmente los de Microsoft, pero esto esta metodología se ha compartido a todos sus clientes y socios estratégicos dedicados al desarrollo de software), disminuyendo la posibilidad de ataques al software por parte de piratas informáticos.[30]

Un segundo objetivo de esta metodología es minimizar el impacto de las vulnerabilidades que continúen con el sistema o el software. Es claro que las vulnerabilidades no se pueden disminuir a cero y cada día se genera nuevos ataques explotando las vulnerabilidades que deben irse atendiendo paulatinamente en un trabajo constante de los grupos de trabajo de seguridad de la informática.

Esta metodología incorpora los conceptos de ciclo de vida de desarrollo de seguridad de Microsoft SDL, que no es más que incluir controles de seguridad dentro de los ciclos de desarrollo de software.

Es de destacar que esta metodología en ningún momento puede ser reemplazo de los procedimientos de seguridad de TI en cada organización, esta metodología aplica para el desarrollo de software.

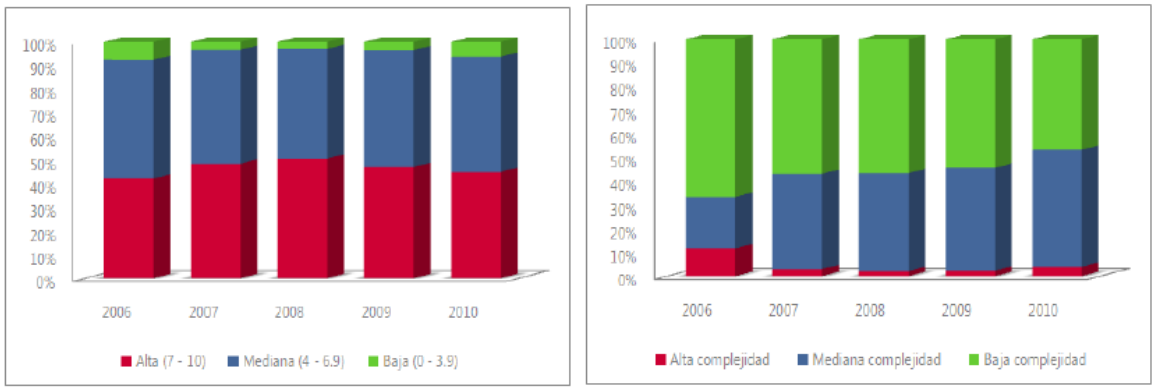
15.1. ANTECEDENTES Y EVOLUCION DE LA METODOLOGIA SDL.

[30] Microsoft Corporation. (2016). Implementación simplificada del proceso SDL de Microsoft.{En línea}. Recuperado de https://www.microsoft.com/es-ES/download/confirmation.aspx?id=12379

El error es inherente a todo lo que el ser humano realiza y el desarrollo de software no se escapa de esta regla y en la medida que se implementen controles y metodologías para el desarrollo de software seguro disminuye la probabilidad de vulnerabilidades en el software.
La tendencia a nivel mundial de las divulgaciones de vulnerabilidades del software es creciente, aunque algunas vulnerabilidades se pueden catalogar como de baja incidencia si existe un porcentaje que se puede considerar de alta incidencia.

En la figura 4 se observan un cuadro donde se reflejan los reportes de vulnerabilidades en la industria del software entre los años 2006 y 2010.

Figura 4 izquierda: Divulgaciones de Vulnerabilidades por gravedad en toda la industria del Software 2006-2010; derecha: Divulgaciones de Vulnerabilidades por complejidad de acceso 2006-2010



Fuente: Informe de progreso del proceso SDL, Microsoft 2004-2010

Entre los años 1990-2000 antes de la adopción de SDL, los procesos de seguridad que incluían en la compañía eran independiente en cada grupo de desarrollo y cada uno según su criterio individual implementaba controles mínimos de seguridad; en ese entonces tenía prelación la funcionalidad de los aplicativos a la seguridad. La seguridad en el desarrollo del software estaba en segundo y tercer plano.

En la década de los noventas y a principios del 2000 se presentaron varios ataques a cargo de virus y software malicioso, como fue el caso de “Melissa” y “Code Red” entre otros, que llevaron a los desarrolladores a cuestionarse a cerca de la seguridad del software e incorporaron conceptos como el STRIDE y el análisis de causa raíz, que fueron soluciones puntuales y no atendían estructuralmente el problema de tener software no seguro.[31]

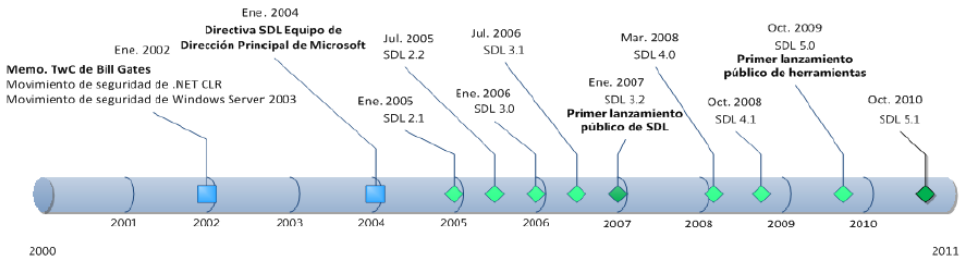
[31] Microsoft Corporation. (2010). Implementación simplificada del proceso SDL de Microsoft. {En Linea}. Recuperado de https://www.microsoft.com/es-ES/download/confirmation.aspx?id=12379

En el 2000 Microsoft adquirió el software de seguridad PREfix, para lo cual se incluyó dicha funcionalidad al sistema operativo Windows 2000.
En 2002 en CLR (.NET Framework Common Language Rountime), los desarrolladores dieron prioridad a construir código seguro sobre las funcionalidades del software y el resultado fue la detección y corrección de una infinidad de errores de seguridad.

Finalmente las directivas de Microsoft entendieron que el tema debía atenderse estructuralmente, como lo dice el informe de Microsoft : “los líderes de seguridad de Microsoft entendieron que un “movimiento” antes del lanzamiento del producto no sería tan eficaz como la integración de la seguridad en el diseño y desarrollo de los productos”[32], y en año 2004 la Dirección de Microsoft aprobó como de obligatorio cumplimento la implementación de SDL como directiva de seguridad para todos los productos de la compañía. Desde ahí a la actualidad ha tenido varias versiones, pero solo hasta el año 2007 fue lanzado al público, hasta entonces era solo de cumplimento interno de la compañía.

La figura 5 muestra en time line de la evolución de la metodología de SDL desde sus inicios hasta su madurez en 2010.

Figura 5: Línea del tiempo evolución de proceso SDL Microsoft.



Fuente: Informe de progreso del proceso SDL, Microsoft 2004-2010

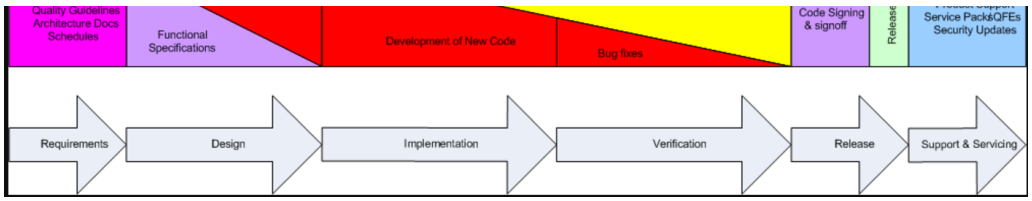
15.2. MODELO DE OPTIMIZACIÓN DE LA METODOLOGÍA SDL.

[32] Microsoft Corporation. (2016). Implementación simplificada del proceso SDL de Microsoft. {En Linea}. Recuperado de https://www.microsoft.com/es-ES/download/confirmation.aspx?id=12379

El proceso estándar para el desarrollo de software, se muestra en la figura 4 y es el aceptado en el mercado (incluso por Microsoft). Este sugiere procesos secuenciales como Levantamiento de requerimientos, diseño, Implementación, Verificación y pruebas, liberación y finalmente soporte.

En la figura 6 se observan las etapas de la metodología SDL de Microsoft.

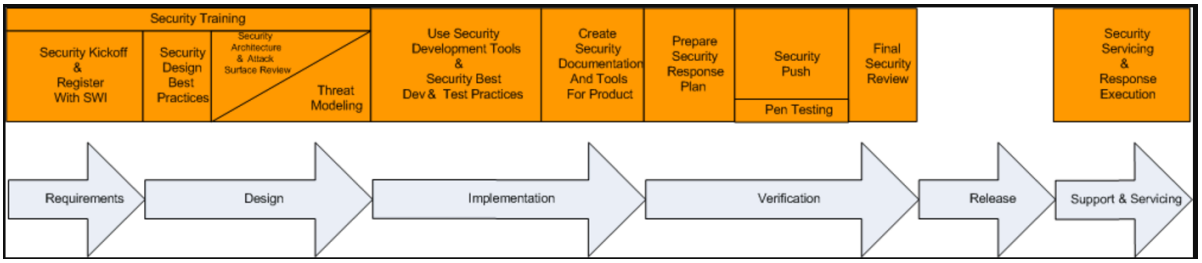
Figura 6. Estándar de Microsoft para procesos de desarrollo



Fuente: https://msdn.microsoft.com/en-us/library/ms995349.aspx

En la figura 7 se muestra el modelo mejorado que incluye la metodología SDL para el proceso de desarrollo de software en Microsoft.

Figura 7. Mejoras de SDL al proceso de desarrollo de software de Microsoft.



Fuente: https://msdn.microsoft.com/en-us/library/ms995349.aspx

La inclusión de seguridad en el ciclo de vida del desarrollo de software lo ha incluido Microsoft[33] , teniendo como definición el SD3+C, lo cuales son: seguro en diseño, definición, distribución y comunicaciones. Estas son los detalles de los principios SD3+C (seguros).

[33] Microsoft Corporation . The Trustworthy Computing Security Development Lifecycle {En Linea}.https://msdn.microsoft.com/en-us/library/ms995349.aspx

Por diseño: Se hace referencia a la protección de los datos y el software en sus definiciones de arquitectura, diseño e implementación del aplicativo. Por Definición: el software no es 100% libre de vulnerabilidad y eso lo saben los arquitectos, por esa razón existe un umbral en cada desarrollo donde se sabe que van a existir fallas de seguridad. Una forma de minimizar estos riesgos es por ejemplo recomendada habilitar los programas para ejecutarse con sus mínimos privilegios a nivel de los usuarios que lo ejecutan.

Por distribución: es incluir en el paquete de software que se distribuye, utilidades o ayudas para el usuario final tenga acceso a información de seguridad que debe tener en cuenta al momento de instalación, para el caso de administradores, o de ejecución, para el caso de usuarios finales.[34]

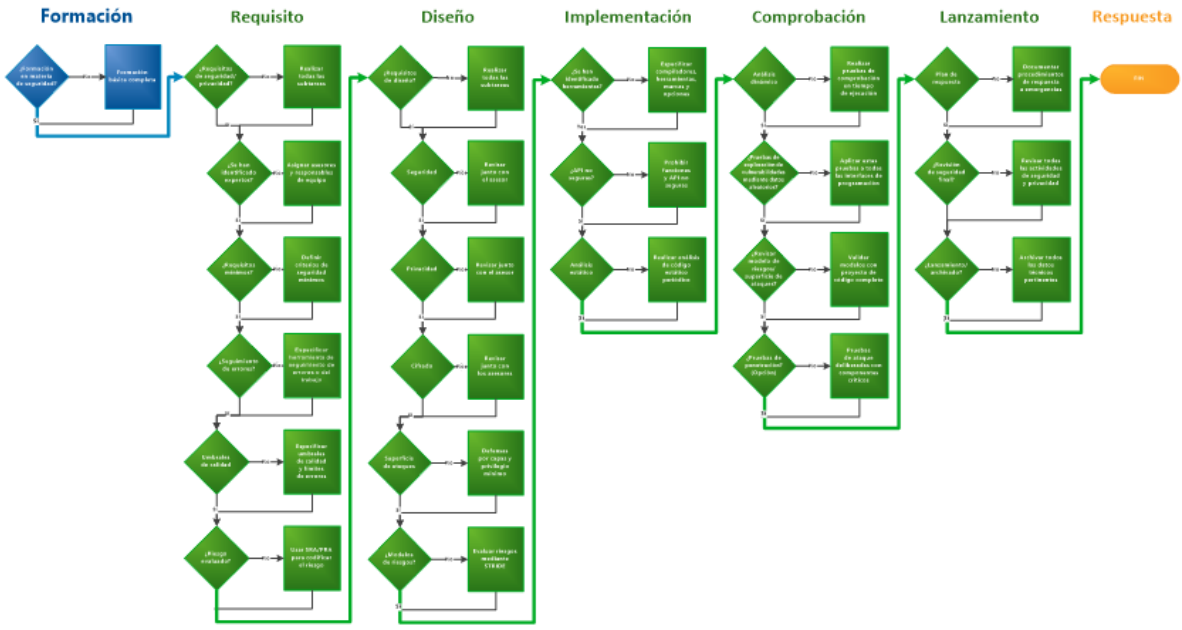
Por comunicaciones: debe existir a nivel de los programas una interacción clara y concisa al usuario o administrador del sistema, para los casos que el aplicativo se vea vulnerado o presente algún ataque que comprometa la integridad de la información.

15.3. FASES O PROCEDIMIENTOS DE LA METODOLOGIA SDL.

Se pueden tener 2 versiones de la metodología, la 1ª corresponde a una menos flexible y maneja un enfoque a proyectos de desarrollos grandes; la 2ª es más flexible y es la recomendada para aplicaciones que se fundamentan en WEB.

[34] Microsoft Corporation. The Trustworthy Computing Security Development Lifecycle {En Linea}.https://msdn.microsoft.com/en-us/library/ms995349.aspx

Figura 8. Ilustración de la implementación simplificada de la Metodología SDL



Fuente: Security Developtment LifeCycle Microsoft Corporation

Es también llamado: requisitos de información. Uno de los objetivos es asegurar que todos los miembros del equipo de desarrollo tengan un nivel adecuado dependiendo del rol que desempeñe dentro del grupo. Que se disponga adicionalmente de una capacitación adecuada de acuerdo a la especialidad de cada ingeniero. Lo ideal es que la capacitación se encuentre alineada con todos los conceptos de tecnología y de seguridad informática de punta y privacidad de la información. Con el cumplimiento de la mencionado anteriormente, se garantiza la

Los Miembros del grupo de desarrollo deben capacitarse anualmente en temas de seguridad de software en temas como: diseño seguro, definición de disminución de superficies de ataque, defensa en profundidad, configuraciones seguras default y privilegios.

Modelos de Riesgo: todo lo concerniente a implicaciones, modelos y restricciones de los modelos de riesgo.

Código seguro: temas de codificación segura como son la saturación de buffer, errores aritméticos, xss, inyection sql, criptografía.

Pruebas de seguridad: en las pruebas abarcando la garantía de seguridad; privacidad: tipos de datos confidenciales, pruebas privacidad, pruebas de confidencialidad, riesgos.

El objetivo es obtener una línea base en cuanto a conocimientos de seguridad de todo el grupo de trabajo del proyecto de desarrollo de software, en caso de requerir para algunos miembros conceptos avanzados se pueden abordar temas adicionales como: diseño de arquitectura e interfaces de confianza, vulnerabilidad y mitigación de amenazas.

15.3.1. Fase de Requisitos

15.3.1.1. Procedimiento 2 Requisitos de Seguridad.

Cuando se piensa en desarrollar un aplicación o solución de software se debe pensar desde el inicio en la seguridad y privacidad, es decir en que el sistema va a ser seguro. Si se fija desde el inicio del proyecto hitos que tengan en cuenta los prerrequisitos de seguridad y privacidad, permitirá al equipo de desarrollo identificar vulnerabilidades y tener un control más detallado del desarrollo que redunde en definitiva en tener un producto más seguro y que respete la privacidad de la información.

El equipo de desarrollo debe en esta fase planear cómo será la integración de seguridad en el proceso de desarrollo del software, inclusión de puntos de control de seguridad que se llevaran a cabo en la construcción del código.Esta planeación no garantiza que, durante el desarrollo del proyecto, se deban cambiar algunas cosas en el camino.

Adicionalmente durante esta fase el equipo de desarrollo se contacta con el analista de seguridad, que este caso debería ser el responsable de la implementación de la metodología Security Development Lifecycle de Microsoft.[35]

Lo que se espera del Analista de Seguridad es que preste asesoría o consultaría en todo lo relacionado con: planes y recomendaciones de seguridad, recursos necesarios para la inclusión de la seguridad en el desarrollo del proyecto y las definiciones de salida o liberación o lanzamiento del producto de software.

[35] Microsoft Corporation. Michael Howard. (Marzo 2005).The Trustworthy Computing Security Development Lifecycle {En Linea}.https://msdn.microsoft.com/en-us/library/ms995349.aspx. pag. 10.

15.3.1.2. Procedimiento 3 Umbrales de calidad y los límites de fallas.

Estas definiciones se utilizan para saber cuáles son los parámetros mínimos de calidad para la seguridad y privacidad del sistema. El objetivo principal de conocer estos límites es poder en la medida que se identifique algún “hueco” de seguridad o vulnerabilidad, ir trabajando en la corrección de las fallas y errores de seguridad. La calidad en un proyecto de desarrollo de software depende de una buena definición de límites de errores el cual se utiliza para identificar el nivel de los huecos de seguridad del producto de desarrollo de software.

Cada grupo dentro del equipo de desarrollo debe identificar los requisitos de seguridad que debe incluir dentro de cada fase del proyecto, sin embargo, es posible que en la fase de amenazas se debe actualizar estos requisitos, dependiendo de la evolución y requerimientos del negocio.

15.3.1.3. Procedimiento 4 Valoración de riesgos de privacidad y seguridad.

Se requieren y son fundamentales para identificar a que modulo del software se le debe hacer una revisión mucho más integral. Estas revisiones son: antes de la liberación se debe identificar que parte del software requieren implementar prototipos de riesgos**; a**ntes del lanzamiento los módulos del software que requieren inspección del diseño de seguridad; antes del lanzamiento se identifican a que módulos del programa se les debe realizar pruebas de pentest previamente acordado. Este procedimiento incluye pruebas adicionales de seguridad para disminuir el riesgo de seguridad y pruebas mediante exploración con aleatoriedad de datos. Identificar el nivel impacto sobre la privacidad de la información, el cual puede será alto, moderado o bajo, para el caso del alto se tiene que la funcionalidad del software incorpora el manejo de datos de información personal, por lo tanto es al que hay que darle mayor atención.[36]

[36] Microsoft Corporation. (2016). Implementación simplificada del proceso SDL de Microsoft. {En Linea}. Recuperado de https://www.microsoft.com/es-ES/download/confirmation.aspx?id=12379https://msdn.microsoft.com/en-us/library/ms995349.aspx. Pag. 10.

15.4. SEGUNDA FASE PROCEDIMIENTO DE DISEÑO.

El diseño es una de las etapas más importantes porque aquí se pueden incluir todos los puntos claves para la seguridad y privacidad que debe tener el sistema.

15.4.1. Procedimiento 5 Requisito de Diseño.

Lo fundamental de esta fase de la metodología es identificar los requerimientos de seguridad del software a desarrollar. Dentro los principales puntos a tener en cuenta se tienen:

Especificar arquitectura y diseño de seguridad: de vital importancia esta definición dado que se deben incluir cuáles serán los pilares de diseño, el tema del software a utilizar, cual es el más apropiado para soportar los posibles ataques o explotación de vulnerabilidades al futura solución, por ejemplo evitar que el aplicativo tenga dependencia circular o manejo de mínimos privilegios para acceder al sistema.[37]

Documentar: se debe documentar la extensión de ataque al software, pues claramente no se podrá llegar al 100 por ciento de efectividad en la seguridad del software, por lo que se recomienda que se exponga toda la funcionalidad del aplicativo solo con los privilegios mínimos y que abarquen el mayor número de usuarios del software. Tener una extensión definida donde los ataques se pueden presentar, permite al grupo de ingenieros encargados de la seguridad, tener un panorama claro para identificar los puntos débiles por donde pueden llegar ataques maliciosos.[38]

Realizar modelo de Amenazas: es importante que se realice esta actividad, pues con este modelamiento se puede identificar que módulos del software con más susceptibles de riesgo de ataque y mitigar este riego, por ejemplo implementando medidas de seguridad como es el caso del cifrado. También ayuda a identificar las particularidades de seguridad que se requiere para cada módulo del software. Especificación de principios de publicaciones adicionales: cuando en el desarrollo se detecten vulnerabilidades, estas deben ser corregidas y probadas y no esperar al lanzamiento para que se reporte la vulnerabilidad y tener que enviar

[37] Ibid., Pag. 11.
[38] Microsoft Corporation. (2016). Implementación simplificada del proceso SDL de Microsoft. {En Linea}. Recuperado de https://www.microsoft.com/es-ES/download/confirmation.aspx?id=12379https://msdn.microsoft.com/en-us/library/ms995349.aspx

actualizaciones sobre la versión, pues esto generara desconfianza en los clientes y usuarios del software.[39]

La metodología recomienda tener claridad en los conceptos características seguras y de seguridad. La característica segura hace referencia al diseño de seguridad, ejemplo validación de datos, criptografía; y las características de seguridad definen funcionalidad de del software que tiene connivencia para la seguridad, ejemplo: dispositivos de hardware como cortafuegos o protocolos de autenticación. Se debe en esta etapa como requisito fundamental, realizar las definiciones en materia de seguridad y privacidad, así como las definiciones de criptografía y deben ir alineadas a las especificaciones funcionales del sistema.

15.4.2. Procedimiento 6 disminución del área de ataques.

Mientras se tenga el área o superficie de ataque reducido, el riesgo es mejor porque los atacantes van a tener un menor frente para acceder a encontrar vulnerabilidades en el sistema, es decir que la disminución de esta superficie es directamente proporcional al riesgo de ataques. Para conseguir una efectiva reducción del área de ataque se deben definir perfectamente la política de privilegios mínimos o implementar la seguridad a nivel de capas.

15.4.3. Procedimiento 7 Definición Modelo de Riesgo.

Estos modelos son especialmente útiles en entornos que se conocen existe un mayor riesgo de que se vea afectada la seguridad. Los modelos de riesgo involucran a todo el equipo de trabajo y facilitan a los especialistas o analistas de seguridad identificar los problemas de seguridad que se pueden presentar en los módulos del sistema. [40]

La metodología recomienda implementar el modelo de riesgo basado en la clasificación STRIDE: suplantación de identidad, modificación de data, repudio, develación de información, denegar servicio y aumento de privilegios, en inglés: “ Spoofing identity, Tampering with Data, Repudiaton, Information disclousure, Denial of Service, Elevation of privilege”.

---

[39] Ibid., Pag. 2.
[40] Microsoft Corporation. (2010). Implementación simplificada del proceso SDL de Microsoft. {En Linea}. Recuperado de https://www.microsoft.com/es-ES/download/confirmation.aspx?id=12379

La idea de cómo utilizar este método consiste en tomar el sistema y fraccionarlo en porciones que tengan consistencia y se analiza cada ítem para identificar si puede ser blanco de amenaza, luego se hacen sugerencias de como disipara estas amenazas y se aplican correctivos. El proceso se repite recurrentemente hasta que quede un mínimo de amenazas que estén dentro del umbral para manejarlas en el sistema. Suplantación de identidad: Requiere de una política eficaz de autenticación para evitar que usuarios se hagan pasar por otros o incluso atacantes se hagan pasar por usuarios válidos.[41]

Modificación de data: ocurre más a menudo en aplicaciones de entornos de internet, dado que se permite casi cualquier entrada por parte de los usuarios, lo cual puede ser aprovechado por atacantes maliciosos a través de inyection sql o XSS. Cuando el sistema realice operaciones con los datos ingresados aumenta el riesgo porque puede estar vulnerando la base de datos dependiendo del tipo de ataque que intrusos hayan realizado sobre el software.[42]

Repudio: el objetivo es que garantizar el no repudio de los usuarios dentro del sistema. Sin embargo, dependiendo del tipo de aplicación se puede dar menos o mayor margen de maniobra al usuario, pues no se puede comparar un sistema bancario o de bolsa con un sistema que provee solo información de consulta, pues en el primer caso si existiera un error del usuario él podría imputar el error al sistema, y si no se tiene un tracking adecuado para el seguimiento de la transacción, el software debe asumir perdida (monetarias). En el segundo escenario no habría pérdida económica.[43]

Develación de información: los sistemas que permitan extraer información de las bases de datos, son claramente un foco de inseguridad que puede acarrear consecuencias desastrosas (hablando en términos económicos, dañar la reputación de la empresa o pérdida de la confianza de proveedores y clientes). Como ejemplos de esta situación podríamos tener: aplicación web corriendo sobre tabletas o celulares, donde se podría presentar el robo de claves y credenciales, correos

[41]Segu-Info. (2010). Que es Stride?.{En Linea}. Recuperado de: https://blog.segu-info.com.ar/2010/03/que-es-stride.html.
[42] Gonzalez Pablo (2014). El modelado de Amenzas.{En Linea} Recuperado de: https://www.flu-project.com/2014/06/el-modelado-de-amenazas-parte-iii.html.
[43] Gonzalez Pablo (2014). El modelado de Amenzas.{En Linea} Recuperado de: https://www.flu-project.com/2014/06/el-modelado-de-amenazas-parte-iii.html.

electrónicos o cualquier información de carácter privado y financiera de los usuarios con el clásico crackeo a través de ataques de fuerza bruta.[44]

Denegación del servicio: para aquellas aplicaciones de uso de mucha concurrencia, se recomienda al momento del diseño, ser muy cuidadosos y minimizar al máximo casos de ataques de denegación del servicio. Ejemplos de cómo evitarlos seria no tener cálculos muy complicados, no hace búsquedas sin índices o sobre archivos muy grandes, entre otros.

Aumento de privilegios: se debe contar con una política y adecuado módulo de gestión para asignación de privilegios de los usuarios del sistema. Se debe impedir a toda costa que se produzca un elevamiento de privilegios por fuera de los parámetros y políticas definidas.

15.5. TERCERA FASE IMPLEMENTACIÓN.

Todo el equipo de desarrollo del proyecto debe publicar las listas de las herramientas que has sido aprobada indicando que temas de seguridad se han revisado o incluido, con el objetivo que todos los grupos del equipo puedan utilizar las implementaciones de seguridad que se han incluido en ellas.

En el ámbito del desarrollo actual hace necesario evaluar cuales funciones API representan amenazas por ser inseguras y no deberían utilizarse en los proyectos de desarrollo. La idea es poder reemplazarlas por alternativas más seguras.

Adicionalmente se cuenta con el Análisis estático el cual tiene que ver con la revisión que se debe hacer al código fuente, el cual debe cumplir con las políticas de seguridad y debe hacerse periódicamente. Sin embargo, se recomienda que se complemente la revisión estática con herramientas o manualmente. Pero es potestad de los grupos de ingenieros del grupo de desarrollo determinar con que

[44] Gonzalez Pablo (2014). El modelado de Amenzas.{En Linea} Recuperado de: https://www.flu-project.com/2014/06/el-modelado-de-amenazas-parte-iii.html.

periodicidad se deben realizar los procedimientos de análisis estático, balanceando le dinámica productiva del grupo y la seguridad del software.[45]

15.6. CUARTA FASE COMPROBACIÓN.

Dentro de esta fase se realizan utilizando pruebas a todo nivel principalmente enfocadas en la privacidad y seguridad del sistema, identificando en la revisión del código fuente posibles huecos de seguridad.

15.6.1. Procedimiento 11 Análisis Dinámico de los programas.

La funcionalidad de los programas se debe probar en esta etapa en ejecución y determinar si están consistente versus el diseño inicial, El resultado de esta etapa sería los inconvenientes de cualquier tipo ya sea de privilegio de usuarios, de seguridad del software, incluso de performance y memoria. Una de las herramientas usadas y recomendadas por la metodología SDL es AppVerifier e incluyendo otras técnicas como fuzz testing pruebas de exploración de las posibles vulnerabilidades.[46]

15.6.1.1. Desbordamiento de Buffer (BufferOverflows).

El desbordamiento de buffer se presenta cuando el buffer está definido de un tamaño fijo y el tamaño de los datos excede esta capacidad del buffer.
Los principales riesgos se presentas por datos inconsistentes, software malicioso en ejecución, falencias en la validación de las entradas, se puede presentar en la pila. Comúnmente se tiene mitos a cerca de los desbordamientos que se debe revaluar, por ejemplo: los desbordamientos de búfer solo afectan los ambientes Microsoft. [47]

Los desbordamientos de búfer no se presentan en software basado en Java o .NET.

---

[45] Microsoft Corporation. (2010).Fase 3 Implementacino.{En linea}.Recuperado de: https://msdn.microsoft.com/en-us/library/windows/desktop/cc307416.aspx
[46] Microsoft Corporation. (2010).proceso SDL.{En Linea}. Recuperado de: https://msdn.microsoft.com/en-us/library/windows/desktop/cc307418.aspx
[47] Microsoft Corporation. (2010). Implementación simplificada del proceso SDL de Microsoft. {En Linea}. Recuperado de https://www.microsoft.com/es-ES/download/confirmation.aspx?id=12379.

Existen muchísimas aplicaciones a los cuales no les afecta los desbordamientos de búfer. Para el riesgo de sobrescribir las estructuras de control:

```
/* Funcion Segura */
void FuncionSegura(char * cadena1)
{
        char Buffer{32};
        /* Copia  cadena1 en el bufer */
        strcpy(Buffer,cadena1);
}
```

Se tienen definido un buffer de 32 byte, pero en caso de una entrada maliciosa a través de cadena1 por ejemplo repitiendo 50 veces el carácter “R”, excedería el buffer y se presentaría desbordamiento de buffer. El desbordamiento se puede presentar cuando es atacada la pila ejemplo:

```
/* Funcion Segura */
void FuncionSegura(char * cadena2)
{
        /* Se asigna un espacio de almacenamiento de 32 bytes en
memoria*/
        Cadena2 * Buffer = (cadena2 *)malloc(32);
        /* copia cadena2 en el buffer */
        strcpy(Buffer,cadena2);
```

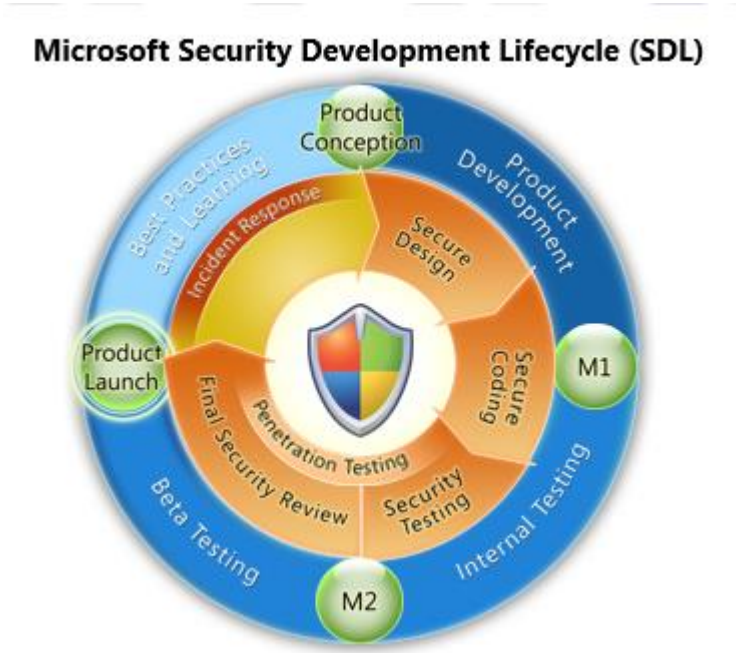
En la situación del caso del ejemplo anterior si se repite 50 veces el carácter “R”, como entrada, excede lo definido para el buffer y se presenta el desbordamiento. Para minimizar el riesgo de desbordamientos de búfer Microsoft SDL recomienda:

Reducir la superficie de ataque y el privilegio mínimo, el cual disminuye el número de vectores de desbordamiento de buffer susceptibles de explotación. La reducción de privilegio mínimo reduce la afectación de un desbordamiento de buffer que ya haya sido explotado.
Buscar funciones riesgosas y determinar el origen de los datos. Se debe hacer una revisión de código fuente de manera manual para ir específicamente a las vulnerabilidades, en este caso a la de desbordamiento de buffer. Utilice bibliotecas y clases más seguras. Por ejemplo (StrSafe, Safe CRT, STL)/ GS, NX), son más resistentes al desbordamiento de buffer.

Prueba de Fuzz: método para pruebas que permite identificar vulnerabilidades de seguridad originadas en su mayoría de las veces por entradas maliciosas. Los sistemas operativos actuales vienen con controles y prevención del desbordamiento de buffer, sin embargo, esto no garantiza que no ocurra, por eso el ingeniero de desarrollo de software deben estar muy pendientes y en lo posible utilizar la metodología que ofrece SDL para dicho control[48]

En la figura 9 se visualiza el ciclo de vida de desarrollo de software de la metodología SDL.

Figura 9. SDL Security Developtment Lifecycle contención Desbordamiento de Buffer



Fuente: Practica SDL 11 Análisis dinámico https://www.microsoft.com/en-us/SDL/process/verification.aspx

15.6.1.2. Atenuación de ataques XSS Script Cruzados.

XSS es un riesgo de seguridad el entorno de la seguridad de informática y que se presenta básicamente en los aplicativos en internet. Se configura desde que a las páginas web se insertan las entrados en páginas dinámicas y que se envían al navegador y sin filtrar caracteres especiales. Esto es aprovechado por los atacantes

[48] Microsoft Corporation. (2010). Automated Penetration Testing with White-Box Fuzzing. {En Linea}. Recuperado de https://msdn.microsoft.com/library/cc162782.aspx

insertan código malicioso en forma de scripts en las páginas dinámicas y pueden ejecutarse dentro de su navegador, y engaña al usuario apoderándose de información crítica y privada para realizar fraudes y hackeos.

15.6.2. Procedimiento 12 Pruebas de Exploración de Vulnerabilidades.

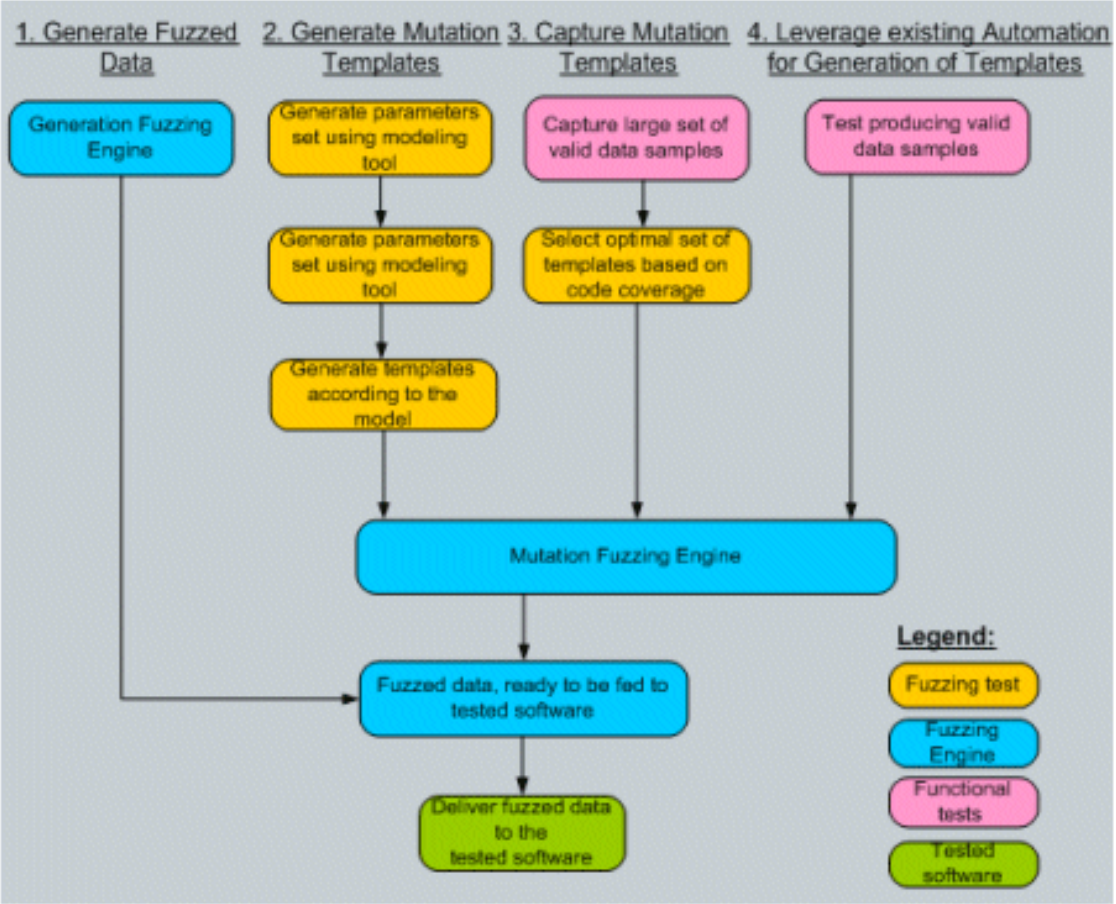
Esta prueba consiste y teniendo como base las especificaciones funcionales del software, en introducir al sistema de manera intencional errores de manera controlada utilizando un análisis dinámico para facilitar que se identifique si existen puntos vulnerables dentro de la aplicación. En estas pruebas debe estar involucrado el especialista en seguridad de la empresa o compañía el cual decidirá si se debe ampliar el espectro de las pruebas o se debe modificar el alcance de las mismas.

Fuzzing es una técnica utilizada para determinar desbordamiento de buffers, excepciones fuera de control, denegación de servicio, fugas de memoria entre otras. Están son algunas de las clasificaciones de fuzzing:
Corrupción de paquetes de datos aleatorios no conociendo el esquema de la data.
Corrupción de paquetes de datos conociendo la estructura de la data.
Envío de datos mal conformados sin revisión.
Generación de información borrada de forma automática.
Buffer fuzzing, corresponde a una mutación de plantilla de fuzzing. [49]

El método fuzzing valida las entradas y sus límites de tolerancia. Los límites que consideran incluyen lo siguiente:Archivo descargado de la web, emails,Conexiones de red,Tuberías, Drivers,Objetos ActiveX.

[49] Microsoft Corporation. (2010). Automated Penetration Testing with White-Box Fuzzing. {En Linea}. Recuperado de https://msdn.microsoft.com/library/cc162782.aspx

Figura 10. Plantillas de mutación método Fuzzing,



Fuente: https://msdn.microsoft.com/library/cc162782.aspx

Para la selección de las plantillas de mutación se debe identificar las igualdades para los escenarios de prueba. Se puede hace manual. Para analizadores de sintaxis un poco más elaborados se tienen como base métodos de automatización para crear plantillas para fuzzing.

Generación de plantillas basadas en modelos por parejas: Las pruebas basadas en modelo lo que buscan es un diseño óptimo para realizar pruebas de sistemas o Software. El modelo requiere

15.6.2.1. Pruebas basadas en modelos por parejas.

Con este modelo se pueden definir parámetros para el analizador, utilizando la herramienta de pares con lo es PICT, el cual se encarga de generar los parámetros individuales de cada plantilla. Dentro de las formas más comunes para implementar pruebas bajo este esquema tenemos: prueba en línea, fuera de línea y manuales. La generación en línea utilizar una herramienta se conectar al sistema y realiza las pruebas de manera dinámica.

15.6.2.2. Pruebas basadas de plantillas de ambientes de producción.

Las plantillas de muestras con ayuda por ejemplo de un sniffer, luego ejecute el software que se está probando utilizando cada una de las plantillas como entrada estas entradas y finalmente se deben catalogar de acuerdo a la cobertura de funcionalidades del software.

15.6.3. Procedimiento 13 Verificar y revisar superficie de ataque.

El objetivo de esta verificación es poder alinear y corregir si las premisas de las especificaciones funcionales se han salido de curso, es decir si difieren de las originalmente planteadas dentro de las fases de diseño del proyecto. Es de vital importancia que una vez finalizado la construcción del código se verifiquen el riesgo y la superficie de ataque. Con esta verificación se garantiza que los cambios y modificaciones al diseño se tengan en cuenta y las posibles nuevas posibilidades de ataque que se generen como producto de estos cambios.[50]

[50] Microsoft Corporation. (2010). Implementación simplificada del proceso SDL de Microsoft. {En Linea}. Recuperado de: https://msdn.microsoft.com/en-us/library/windows/desktop/cc307418.aspx

15.7. QUINTA FASE LANZAMIENTO.

15.7.1. Procedimiento 14 Plan Respuesta a Incidentes.

Para cada liberación de aplicaciones que están bajo la metodología SDL tienen que considerar un esquema de soporte y respuesta a errores e incidentes. Incluso los programas a los cuales no se les identifico vulnerabilidades y que pueden posteriormente y en el momento de la liberación estar bajo amenaza. El plan de Soporte o atención de incidentes debe considerar:

Un grupo de trabajo o un plan de soporte que responda desde todos niveles: comercial, comunicaciones, ingeniería o administrativos que vendrían siendo los primeros niveles de soporte en caso de escalamiento de una incidencia de seguridad. Soporte 7 x 24 con personal especializado y con conocimiento para atender un escalamiento de un error o evento que comprometa la seguridad del sistema. También se debe contar con un plan de servicios de seguridad con información detallada de proveedores para código que sea legado, versiones, archivos fuentes, nombres de archivos.[51] Se recomienda definir un cuestionario de privacidad como respuesta al incidente: Las siguientes preguntas, aunque no son mandatarias puedes ser un punto de referencia: ¿Cuál es el Nombre del Proyecto?, ¿en qué fecha el usuario tendrá acceso al sistema?, ¿Qué persona o rol es responsable de la privacidad?.

Se debe tener una evaluación inicial para poder determinar el nivel de impacto de seguridad en la privacidad e identificar el grado de riesgo del software.

Para determinar el impacto del riesgo a la privacidad se debe realizar el siguiente cuestionario:
El sistema guarda información de identificación personal SI, NO.
En caso afirmativo, deja la información local o lo transfiere a una BD 1 o 2.
Se monitorea continuamente al usuario SI, NO.
Tiene privilegios para instalar software nuevo SI, NO.
Transmite datos anónimos SI, NO.

Principales preguntas del cuestionario para evaluar el modelo de Amenazas:

Detalle que usuarios del sistema almacenan o transmiten información personal.

[51] Microsoft Corporation. Michael Howard. (marzo 2005). The Trustworthy Computing Security Development Lifecycle {En Línea}.https://msdn.microsoft.com/en-us/library/ms995349.aspx

Describa la justificación para guardar este tipo de información.
Describa el software que instala en los equipos.
Detalle como los usuarios tienen acceso a información pública.
Describa el control que se tiene a la función.

15.7.2. Entorno de respuesta de impactos de privacidad.

A continuación, se resume los principios que se deberían tener presentes para responder a un incidente, ataque o violación de privacidad y los escalamientos de los mismos, manejar las comunicaciones entre las áreas implicadas en el incidente e identificar los puntos a mejora para evitar que se repitan.

La prioridad de los impactos en los ataques a la privacidad se puede definir por los siguientes criterios:

Ataque, robo o violación a data o información privada de cliente o empresa.
No cumplimiento de los requerimientos de seguridad.
Demandas originadas por ataques a la privacidad.
Solicitudes del ente regulatorio productos de ataques maliciosos
Los medios de comunicación frente a un ataque de la empresa.

Cuando se recibe una notificación o correo informando de un incidente de seguridad, se debe tener la información precisa de los empleados o terceros especialistas en el tema y que deben dar soporte de segundo y tercer nivel. Cuando se escale este tipo de incidentes se debe ser concreto en el escalamiento, es decir, se debe tener claro la fuente que origina la notificación, el impacto del incidente, valides de la fuente, resumen y time line para atender esta solicitud y se debe hacer un resumen consistente y claro del problema.

15.7.3. Procedimiento 15 Revisión de Seguridad Final.

Es de vital importancia que se realice previo al lanzamiento del software, una revisión final de todos lo que se definió como procesos de seguridad para el sistema.

Esto debe incluir los datos de las personas que dan soporte de seguridad al sistema, para los sistemas propios de la compañía, sistemas legados y proveedores.

15.7.4. Lanzamiento y publicación.

El analista que cumple con el rol gestor de seguridad, debe garantizar a través del cumplimiento de la atapa anterior, es decir, la de revisión de seguridad final, que el proyecto ha superado todos los prerrequisitos de seguridad, posterior a esta certificación, se debe almacenara o archivar absolutamente toda la información que ayude para el mantenimiento futuro del sistema, entre estos entregables tenemos: Manuales de especificaciones funcionales y no funcionales, código fuente, los archivos bin, especificaciones del modelo de seguridad, documentación del sistema, planes de emergencia y continuidad , Información de licencias con los proveedores.[52]

15.8. TAREAS DE SEGURIDAD OPCIONALES.

Cuando el software que se libera a producción se va a ejecutar en ambientes críticos, se debe escalara al analista de seguridad los puntos adicionales para que el análisis de seguridad del sistema lo contemple.

15.8.1. Análisis Manual del código Fuente.

Esta es una actividad adicional y manual que la metodología SDL la considera voluntario según lo decida el grupo de seguridad del proyecto, pues esta labor solo la puede realizar el personal del grupo de seguridad que tenga un conocimiento amplio del tema. Esta revisión se apoya en herramientas, pero siempre hay la necesidad de realizar análisis, pues los objetivos de esta actividad es hallar los huecos o puntos críticos de seguridad del software.

15.8.2. Pruebas de Penetración.

[52] Microsoft Corporation. Proceso SDL (2012).{En linea}. Recuperado de: https://msdn.microsoft.com/en-us/library/windows/desktop/cc307420.aspx

Estas pruebas las debe realizar expertos en la materia, el cual consiste en ataques simulando a los intrusos malintencionados y cuyo objeto principal es poder detectar huecos de seguridad en software que sean originados por la falta de validaciones y controles en el software.

Se debe tener muy en cuenta que los inconvenientes o hallazgos que resulten de las pruebas de penetración deben ser solucionados antes del visto bueno para la liberación del software a producción.

Es importante acudir a toda la literatura y casos de uso que existen en la web a cerca de problemas de seguridad encontrados a sistemas del mismo tipo del cual estuviésemos probando, es posible que estas consultas nos ayuden a identificar y subsanara errores que ya se hayan presentado en este tipo de software.[53]

15.8.3. Otras Condiciones de la Metodología SDL.

15.8.4. Análisis de Causa Raíz.

Es un punto muy importante como ítem a tener en cuenta cuando puesto que con el análisis de causa raíz lo que se busca es a partir de un error o hueco de seguridad que no ha sido tipificada, identificar la causa o la esencia que lo origino, por ejemplo: omisión por parte del desarrollador de software que evidencia un error humano, un error de definición por parte del proyecto, un software entre otros.[54]

La información recolectada ayudara para que en adelante errores o huecos de seguridad del mismo tipo se puedan prevenir y permitan a la metodología SDL enriquecerse.

15.8.5. Actualizar el proceso.

La recomendación es que se haga una actualización del proceso SDL por lo menos una vez al año, tomando como base los análisis de causa raíz, los ajustes a las

[53] Microsoft Corporation. Michael Howard. (marzo 2005). The Trustworthy Computing Security Development Lifecycle {En Linea}.https://msdn.microsoft.com/en-us/library/ms995349.aspx
[54] Ibid., Pag. 16.

definiciones de la metodología, los avances tecnológicos, tomando en cuenta que las metodologías que se implementa para asegurar el software no son fijos y por el contrario deben ser muy dinámicos de acuerdo a como se mueva el entorno de la tecnología y las especializaciones de los atacantes. [55]

15.9. PROCESO DE VERIFICACIÓN DE SEGURIDAD DEL SOFTWARE.

Las empresas u organizaciones que se dedican al desarrollo de software, tienen especial cuidado en temas de verificación del software y en lo que tiene que ver con seguir con los pasos que se describen en la metodología SDL. La validación de seguridad de un software involucra muchos actores: se debe tener un almacenamiento central donde se tenga todos los elementos recaudados de la metodología SDL, por ejemplo: Escritos, modelos, diseños de seguridad. Debe existir una verificación para que el acceso al sistema solo sea por usuarios autorizados. Para que el diagnostico diagnóstico sea el adecuado, los hallazgo o errores que se están revisando y evaluando debe ser clasificado en el contexto correctamente y su información debe ser consistente.[56]

[55] Microsoft Corporation. Michael Howard. (marzo 2005). The Trustworthy Computing Security Development Lifecycle {En Linea}.https://msdn.microsoft.com/en-us/library/ms995349.aspx
[56] Microsoft Corporation. Michael Howard. (marzo 2005). The Trustworthy Computing Security Development Lifecycle {En Linea}.https://msdn.microsoft.com/en-us/library/ms995349.aspx

16. METODOLOGIA CbyC (CORRECTNESS BY CONSTRUCTION).

En la actualidad la mayoría de las aplicaciones corporativas ya se encuentran disponibles en la Internet y dada la globalización de la WEB ha abonado el terreno para que los ataques se encuentren en aumento y se dirijan en alto porcentaje a este segmento (Segmento Corporaciones). Lo anterior ha generado en los investigadores y desarrolladores una conciencia para que todos los desarrollos basados en WEB se blinden. Para ello cuentan con la metodología "Correctness by Construction) CbyC", metodología desarrollada por Praxis High integrity System.[57]

El principal objetivo de esta metodología es desde el inicio del desarrollo generar sistemas con calidad y fundamentados como obligatoriedad de la seguridad. Los principios básicos de esta metodología son: que los errores sean muy difíciles de introducir en el sistema de información.

Eliminar o suprimir los errores muy rápidamente una vez se ha notificado de la existencia de uno. Hacer pruebas de aptitud que se involucre en todos los módulos del sistema de información que está en proceso de desarrollo.

Algo muy importante de esta metodología es que permite la detección de errores muy rápido después de la salida a producción o justo antes de la salida, todo esto porque los procesos involucrados en esta metodología lo permiten.

La metodología CbyC utiliza estándares que no dan pie para la confusión al contrario de las herramientas de programación mayormente utilizadas que cuyos resultados son aleatorios. Esta metodología se basa en desarrollo incremental, la disminución de tiempos en la construcción del software y la no repetición, por ejemplo: se debería desvincular la definición del sistema, del diseño del aplicativo, porque mientras la definición del sistema busca el cómo se construye el software,

[57] Brito Carlos Joaquín. (diciembre 2013). Metodología para el software Seguro.{En Linea} Recuperado de http://recibe.cucei.udg.mx/revista/es/vol2-no3/pdf/computacion05.pdf. Pag 4.

el diseño se encarga y tiene que ver con temas seguridad, protección y productividad.

16.1. FASES METODOLOGIA CbyC.

Debido a que la metodología une métodos tradicionales de programación con metodologías agiles debe utilizar codificación exacta junto con un desarrollo incremental, el cual permite hacer un tracking del desarrollo y poder verificar y proceder a la retroalimentación sobre el producto que se está desarrollando.

Figura 11. **Fases Metodología CbyC**

Fuente: Amey Peter (2006) Núcleo de CbyC

16.1.1. Fase Requerimientos.

Esta fase es una de las más importantes dado que los requerimientos son completos y detallados, el objetivo del software será el esperado. Se deben indicar aquí los requerimientos funcionales y no funcionales; el usuario debe incluir detalladamente lo que necesita incluyendo diagramas para ser más explícito acerca de lo que necesita, debe incluir un análisis de impacto de las amenazas que implica cada uno de los ítems a solicitar. Dentro de los requisitos también se incluye el modelamiento de la política de seguridad del sistema, pues contribuye a que el objetivo de

seguridad no se ambiguo y se desvié del original y que las validaciones previas a la generación de los entregables sean más exhaustivas. [58]

16.1.2. Fase de Especificación del Software

En esta fase se deben incluir las condiciones de errores y se debe visualizar el comportamiento del software como un “caja”, no se debe incluir información detallada de información sobre el software, se deben especificar temas funcionales y no funcionales, así como realizar la solicitud de seguridad dependiendo de cada una de las amenazas; incluso detallando las condiciones de error.[59]

16.1.3. Fase Diseño de Alto Nivel.

En esta fase se detalla la arquitectura del software y los temas no funcionales como lo es la seguridad que se van a tener en cuenta en el aplicativo.
También se detallan todos los módulos que componen la solución de software con sus funcionalidades como por ejemplo estructura del proceso o esquema de base de datos.

16.1.4. Fase Especificaciones de los Módulos.

En esta fase se deben definir de manera encapsulada cada uno de los módulos del sistema de información o programa a desarrollar y teniendo siempre en cuenta toda la información entregada por los usuarios. Se debe considerar las posibles fallas del software por lo tanto los módulos deberían ser fácilmente intervenidos en caso de algún incidente.[60]

---

[58] Amey Peter.(Mayo 2013). Cbyc Correctness by construction.{En Linea}. Recuperado de: https://www.us-cert.gov/bsi/articles/knowledge/sdlc-process/correctness-by-construction

[59] Amey Peter.(Mayo 2013). Cbyc Correctness by construction.{En Linea}. Recuperado de: https://www.us-cert.gov/bsi/articles/knowledge/sdlc-process/correctness-by-construction

[60] Amey Peter.(Mayo 2013). Cbyc Correctness by construction.{En Linea}. Recuperado de: https://www.us-cert.gov/bsi/articles/knowledge/sdlc-process/correctness-by-construction

16.1.5. Fase de Pruebas.

Sin lugar a dudas en todo desarrollo las pruebas unitarias y de usuario final son claves para garantizar que el producto quede los cercano a estar libre de errores.
Para la realización de las pruebas se debe contar con un documento de prueba donde estén completamente especificadas, incluyendo pruebas de límites y los resultados esperados. En CbyC las pruebas son integrales a nivel del sistema propiamente dicho y basándose en especificaciones y no se realizan prueba por unidad.

16.1.6. Fase Construcción del software.

El modelo CbyC[61] soporta el esquema de Team Software process, es decir el TSP el cual cuyo objetivo central es integrar al grupo o equipo de desarrollo de software para que se trabaje pensando en que tienen un foco único para mejorar los productos de software y que tiene como valor agregado la disminución de la tasa de errores en el mismo.
El TSP, es un proceso básicamente para Equipos de ingenieros que desarrollan software y les permite realizar productos con mucha calidad.

Figura 12. ***Procesos del desarrollo TSP***

Fuente: http://alejandrogomeztsp.blogspot.com/

[61] AMEY, Peter.(Mayo 2013). Cbyc Correctness by construction.{En Linea}. Recuperado de: https://www.us-cert.gov/bsi/articles/knowledge/sdlc-process/correctness-by-construction

El TSP entro a cubrir las falencias del PSP (Personal Software process), el cual se enfocó a procesos industriales y individualizaba cada proceso y solo enfatizaba en el desarrollo del software en lo que tiene que ver con el diseño y pruebas.

Figura 13. ***PSP vs TSP***

Fuente: http://alejandrogomeztsp.blogspot.com/

Para solventar esta situación el ingeniero Watts Humprey desarrollo TSP[62], que nos solo consideraba los postulados del PSP, sino que también incorporó los requisitos, pruebas integrales y la documentación entre otros. Muy importante la inclusión de roles dentro de los equipos de desarrollo como: Líder, Gestor de desarrollo, Gestor de pruebas, Administrador de requerimientos. Con modelo TSP los ingenieros pueden detectar tempranamente un error de software y corregirlo debido a que los ciclos de prueba son más cortos y rápidos. Tras la formación del equipo de trabajo se debe poner en marcha la gestión del mismo. Todo proyecto de desarrollo bajo este esquema se compone de los siguientes ciclos: Lanzamiento, estrategia, plan, requisitos, diseño, implementación, pruebas, liberación.

[62] GOMEZ,David.(noviembre 2009). Team Software Process.{En Linea}. Recuperado de: http://alejandrogomeztsp.blogspot.com.co

17. FUNDACION OWASP.

OWAS es un Proyecto de seguridad en aplicaciones WEB cuyo conglomerado de integrantes tiene la comunidad abierta a todos los desarrolladores.

La Fundación OWAS es una Fundación sin animo de lucro que ayuda a combatir las vulnerabilidades del software para que sea más Seguro.

Uno de los principales objetivos de la fundación OWASP[63] es capacitar a los profesionales de servicios de información IT sobre los impactos que trae las vulnerabilidades sobre la seguridad del software web, de las empresas y las personas.

Para mitigar este impacto, la fundación tiene algunos métodos que ayudan de la mejor manera a protegerse o prevenir un ataque a las vulnerabilidades.

Otra de la recomendación de la fundación OWAS es estar muy atentos y apoyarse en algunas herramientas adicionales que provee la fundación como lo son:

- El OWASP[64] Cheat Seat, el cual corresponde a una guía para la seguridad de sistemas basados en WEB.
- Guía de Pruebas OWASP y la guía de revisión de código OWASP.
- Guía ASVS, la cual es una guía para implementar controles de seguridad en aplicaciones.

[63] OWASP. (enero 2018) Welcome to Owasp. {En línea}. Recuperado de: https://www.owasp.org/index.php/Main_Page

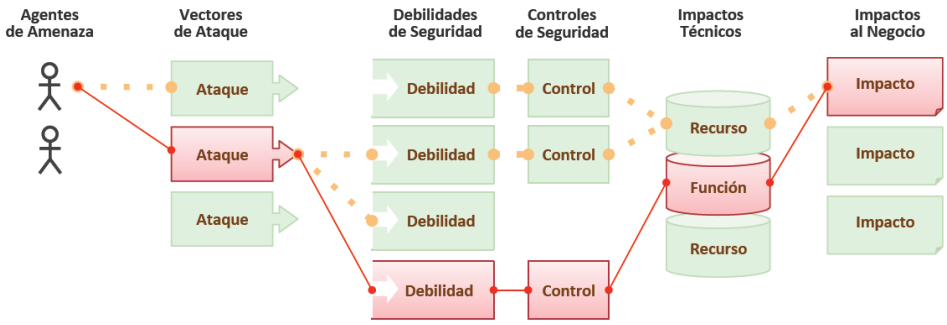
[64] OWASP. (2017) Owasp Top 10 2017 Los 10 Riesgos más críticos en Aplicaciones Web. {En línea}. Recuperado de: https://www.owasp.org/images/5/5e/OWASP-Top-10-2017-es.pdf. Pag. 1

17.1. RIESGO DE SEGURIDAD DE APLICACIONES

Sin temor a la equivocación se puede afirmar: los sistemas de información y las aplicaciones de software son el objetivo inicial de los atacantes, es por eso que toma mucha importancia la seguridad en las aplicaciones y a nivel de la construcción del software es decir en su desarrollo. En primer término, se debe tener mucha claridad sobre cual vía tomara el hacker para realizar su ataque. Los efectos que puede producir un ataque pueden ser de cualquier índole desde no afectar a que el negocio cierre sus puertas.

Por lo mencionado anteriormente, es que se debe prestar mucha atención a los diferentes agentes de amenazas, vector de ataque y los huecos de seguridad.[65]

Figura 14. ***Riesgo Seguridad Aplicaciones***



Fuente: Owasp Top 10-2017 Los Riesgos más Críticos en Aplicaciones Web.

Los riesgos que define el top10 del OWASP, van alineados al tipo de agresión o vulnerabilidad o la clase de afectación que generen al sistema.[66]

[65] OWASP. (2017) Owasp Top 10 2017 Los 10 Riesgos más críticos en Aplicaciones Web. {En línea}. Recuperado de: https://www.owasp.org/images/5/5e/OWASP-Top-10-2017-es.pdf. Pag. 3
[66] Ibid., Pag. 6.

17.2. OWASP ASVS VERIFICACIÓN DE SEGURIDAD DE APLICACIÓN.

Es un proyecto estándar de OWASP que tiene como objetivo proporcionar los criterios para probar la seguridad técnica de las aplicaciones web y da una guía de requisitos para realizar un desarrollo seguro. Este estándar es abierto y su alcance llegar hasta los controles técnicos y de entorno y los requisitos tiene como pilar los siguientes objetivos, confianza, control de seguridad, control de seguridad en contratos. [67]

17.2.1. Requisito de Modelado.

Desde la planeacion y el modelo arquitectonico del sistema se exigen seguridad modelo; estos requisitos tienen que ver con la arquitectura y diseño de la aplicación.[68]

En la figura 15 el proyecto ASVS define los puntos clave de seguridad en la fase de diseño y modelo de amanezas[69]

[67] OWASP. (marzo 2018).OWASP Application security verification estándard proyect {En línea}. Recuperado de: https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project.

[68] OWASP. (marzo 2018).Application security verification estándard proyect {En línea}. Recuperado de: https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project.

[69] OWASP. (marzo 2018). ASVS V1 Arquitecture {En línea}. Recuperado de: https://www.owasp.org/index.php/ASVS_V1_Architecture

Figura 15. Requisitos de seguridad fase de diseño

#	Descripción
1.1	Todos los componentes de la aplicación se identifican y se sabe que son necesarios.
1.2	Los controles de seguridad nunca se aplican solo en el lado del cliente, sino en los puntos finales remotos respectivos.
1.3	Se ha definido una arquitectura de alto nivel para la aplicación y todos los servicios remotos conectados y se ha abordado la seguridad en esa arquitectura.
1.4	Los datos considerados sensibles en el contexto de la aplicación están claramente identificados.
1.5	Todos los componentes de la aplicación se definen en términos de las funciones comerciales y / o funciones de seguridad que proporcionan.
1.6	Se ha producido un modelo de amenaza para la aplicación y los servicios remotos asociados que identifica posibles amenazas y contramedidas.
1.7	Todos los controles de seguridad tienen una implementación centralizada.
1.8	Los componentes están segregados entre sí a través de un control de seguridad definido, como la segmentación de red, las reglas de firewall o los grupos de seguridad basados en la nube.
1.9	Existe un mecanismo para aplicar actualizaciones de la aplicación.
1.10	La seguridad se aborda en todas las partes del ciclo de vida de desarrollo de software.
1.11	todos los componentes de aplicaciones, bibliotecas, módulos, marcos, plataforma y sistemas operativos están libres de vulnerabilidades conocidas
1.12	Existe una política explícita sobre cómo se administran las claves criptográficas (si las hay) y se impone el ciclo de vida de las claves criptográficas. Idealmente, siga un estándar de gestión de claves como NIST SP 800-57.

Fuente: https://www.owasp.org/index.php/ASVS_V1_Architecture

17.2.2. Requisito de autenticacion.

El objetivo de este control es identificar la identidad del emisor en una comunicación, en la figura 16 se describen controles de autenticación cuyo objetivo es: asegurar que accedan solo usuarios autorizados, validando credenciales. Otros puntos claves son: Garantizar protección al lado del servidor, no completar automáticamente credenciales, Garantizar la falla segura de los controles de autenticación.[70]

[70] OWASP. (marzo 2018). Application security verification estándard proyect {En línea}. Recuperado de:https://www.owasp.org/index.php/ASVS_V2_Authentication

Figura 16. Requisitos de verificación de autenticación

#	Descripción
2.1	Verifique que todas las páginas y recursos estén protegidos por la autenticación del lado del servidor, excepto aquellos destinados específicamente a ser públicos.
2.2	Verifique que la aplicación no complete automáticamente las credenciales, ya sea como campos ocultos, argumentos URL, solicitudes Ajax o en formularios, ya que esto implica el almacenamiento de contraseñas de texto plano, reversible o descifrable. Las nonces de tiempo aleatorio son aceptables como bases de datos, como para proteger formularios de cambio de contraseña o formularios de contraseña olvidada.
2.6	Verifique que todos los controles de autenticación fallen de forma segura para garantizar que los atacantes no puedan iniciar sesión.
2.7	Los campos de verificación de entrada de contraseña permiten o fomentan el uso de frases de contraseña, y no evitan las contraseñas largas o las contraseñas altamente complejas que se ingresan.
2.8	Verifique que todas las funciones de identidad (por ejemplo, contraseña olvidada, contraseña, correo electrónico, token de 2FA, etc.) tengan los controles de seguridad, como mecanismo de autenticación principal (por ejemplo, formulario de inicio de sesión).
2.9	Verifique que la funcionalidad de cambio de contraseña incluya la contraseña anterior, la nueva contraseña y una confirmación de contraseña.
2.12	Verifique que todas las decisiones de autenticación se puedan registrar, sin almacenar identificadores o contraseñas de sesión confidenciales. Esto debería incluir solicitudes con los metadatos relevantes necesarios para las investigaciones de seguridad.
2.13	Verifique que las contraseñas de la cuenta tengan un hashing de una manera con un salt, y que haya suficiente factor de trabajo para derrotar la fuerza bruta y los ataques de recuperación de hash de contraseñas.
2.16	Verifique que todos los datos de la aplicación se transmiten a través de un canal cifrado (por ejemplo, TLS).
2.17	Verifique que la función de contraseña olvidada y otras rutas de recuperación no revelen la contraseña actual y que la nueva contraseña no se envíe en texto claro al usuario. En su lugar, se debe usar un enlace de restablecimiento de contraseña de una sola vez.
2.18	Verifique que la enumeración de información no sea posible mediante el inicio de sesión, el restablecimiento de contraseña o la funcionalidad olvidada de la cuenta.
2.19	Verifique que no haya contraseñas predeterminadas en uso para el marco de aplicación o cualquier componente utilizado por la aplicación (como "admin / password").
2.20	Verifique que la anti-automatización esté implementada para evitar las pruebas de credenciales violadas, el forzamiento bruto y los ataques de bloqueo de cuenta.
2.21	Verifique que todas las credenciales de autenticación para acceder a servicios externos a la aplicación estén encriptadas y almacenadas en una ubicación protegida.
2.22	Verifique que la contraseña olvidada y otras rutas de recuperación utilicen un TOTP u otro token de software, inserción de dispositivo móvil u otro mecanismo de recuperación fuera de línea. El uso de SMS ha sido desaprobado por NIST y no debe utilizarse.
2.23	Verifique que el bloqueo de la cuenta esté dividido en estado de bloqueo suave y fijo, y estos no son mutuamente exclusivos. Si una cuenta se bloquea temporalmente de forma suave debido a un ataque de fuerza bruta, esto no debe restablecer el estado de bloqueo.
2.24	Verifique que si se requieren preguntas secretas, las preguntas no violen las leyes de privacidad y sean suficientemente fuertes para proteger las cuentas de la recuperación maliciosa.

Fuente: https://www.owasp.org/index.php/ASVS_V2_Authentication

17.2.3. Requisito gestion de session.

En un sistema basado en WEB es fundamental controlar y mantener el estado de un usuario que interactúa con el sistema, es decir la gestione de sesión se puede definir como el grupo de controles que manejan la relación o interacción de un usuario con una aplicación basada en la WEB. Otros controles importantes: validar que las sesiones se invaliden cuando el usuario cierre la sesión; expiración de sesiones de acuerdo a una franja de tiempo; Verificar que el ID de sesión nunca exponga la URL, mensajes de alertas o warning, errores; verificar las autenticaciones y re-autenticaciones generen una nueva sesión; verificar que se dé la finalización de sesiones para cuando exista cambio de contraseña. [71]

17.2.4. Requisito Control de Acceso.

Se deben garantizar en este control que los usuarios que accedan al sistema poseen credenciales verdaderas para hacerlo y que estos usuarios tienen definidos

[71] OWASP. (marzo 2018). Application security verification estándard proyect {En línea}. Recuperado de: https://www.owasp.org/index.php/ASVS_V3_Session_Management

claramente roles y privilegios los cuales deben tener un nivel de seguridad alto para evitar ser mal utilizados por atacantes maliciosos.[72]

17.2.5. Validacion E/S.

Sin lugar a dudas la fragilidad de seguridad más común de las aplicaciones web es la escasa o mala validación de las entradas que provienen de clientes o de dominios. Esta debilidad es la causa de todo el resto de vulnerabilidades de seguridad como son, inyection SQL, desbordamiento de buffer, ataques de localización.[73]

En la figura 17 se listan las validaciones de E/S de este control y define una premisa clave para la seguridad del sistema: las entradas de un usuario externo o fuente externa nunca se presumirá que es confiable, se debe dar el tratamiento de no confiable y aplicar las validaciones correspondientes.

Figura 17. Registro de verificación de E/S

#	Descripción
5.3	Verifique que las fallas de validación de entrada del lado del servidor den como resultado el rechazo de la solicitud y se registren.
5.5	Verifique que las rutinas de validación de entrada se apliquen en el lado del servidor.
5.6	Verifique que la aplicación utilice un mecanismo de control de validación de entrada centralizado.
5.10	Verifique que todas las consultas de la base de datos estén protegidas mediante el uso de consultas parametrizadas o el uso adecuado de ORM para evitar la inyección de SQL.
5.11	Verifique que la aplicación no sea susceptible a la inyección LDAP o que los controles de seguridad impidan la inyección LDAP.
5.12	Verifique que la aplicación no sea susceptible a la inyección de comandos del sistema operativo o que los controles de seguridad impidan la inyección de comandos del sistema operativo.
5.13	Verifique que la aplicación no sea susceptible a Inclusión de archivo remoto (RFI) o Inclusión de archivo local (LFI) cuando se usa contenido que es una ruta a un archivo.
5.14	Verifique que la aplicación no sea susceptible a inyección de XPath o ataques de inyección de XML.
5.15	Verifique que todas las variables de cadena colocadas en HTML u otro código de cliente web estén correctamente codificadas contextualmente de forma manual o que utilicen plantillas que codifican contextualmente de forma automática para garantizar que la aplicación no sea susceptible a ataques reflejados, almacenados o DOM Cross-Site Scripting (XSS).
5.16	Verifique que la aplicación no contenga vulnerabilidades de asignación de parámetros masivos (AKA unión variable automática).
5.17	Verifique que la aplicación tenga defensas contra los ataques de contaminación de parámetros HTTP, particularmente si el marco de trabajo de la aplicación no hace distinción sobre la fuente de los parámetros de solicitud (GET, POST, cookies, encabezados, entorno, etc.)
5.19	Verifique que todos los datos de entrada estén validados, no solo los campos de formulario HTML, sino todas las fuentes de entrada, como las llamadas REST, los parámetros de consulta, los encabezados HTTP, las cookies, los archivos por lotes, las fuentes RSS, etc. utilizando la validación positiva (lista blanca), luego formas de validación menores, como la lista gris (eliminando cadenas malas conocidas) o rechazando entradas malas (lista negra).
5.20	Verifique que los datos estructurados estén fuertemente tipados y validados contra un esquema definido, incluidos los caracteres permitidos, la longitud y el patrón (por ejemplo, números de tarjetas de crédito o teléfono, o validar que dos campos relacionados son razonables, como validar suburbios y códigos postales o postales).
5.21	Verifique que los datos no estructurados se desinfecten para imponer medidas de seguridad genéricas, como los caracteres permitidos y la longitud, y que se eviten los caracteres potencialmente dañinos en un contexto determinado (por ejemplo, nombres naturales con Unicode o apóstrofes, como ね or O'Hara)
5.22	Verifique que todas las entradas de HTML no confiables de los editores WYSIWYG o similares se desinfecten adecuadamente con una biblioteca de sanitizador HTML o una función de marco.
5.24	Verifique que cuando los datos se transfieren de un contexto DOM a otro, la transferencia utiliza métodos seguros de JavaScript, como el uso de innerText o .val para garantizar que la aplicación no sea susceptible a los ataques DOM Cross-Site Scripting (XSS).
5.25	Verifique al analizar JSON en navegadores o bases de datos JavaScript, que JSON.parse se utiliza para analizar el documento JSON. No use eval () para analizar JSON.

Fuente https://www.owasp.org/index.php/ASVS_V5_Input_validation_and_output_encoding

[72] OWASP. (2017).Access control verification {En línea}. Recuperado de: https://www.owasp.org/index.php/ASVS_V4_Access_Control.
[73] OWASP. (2017).(agosto 2017). Input validation and output encoding. {En línea}.Recuperado de: https://www.owasp.org/index.php/ASVS_V5_Input_validation_and_output_encoding

17.2.6. Criptografia

Se debe garantizar que el sistema cumpla con los siguientes requerimientos: que al momento de falla los procesos de criptografía, éstos fallen adecuadamente y realizar el tratamiento de los errores correctamente. Gestión adecuada para el acceso a las claves; validar si la generación aleatoria de números está acorde a los requerimientos del módulo criptográfico. Aplicación correcta del ciclo de vida de vida de password; política para manejo de contraseñas.[74]

17.2.7. Manejo de Errores

El manejo y registro de errores da una herramienta muy útil para los administradores como apoyo a la resolución de incidentes. Se debe garantizar que no se incluya en los mensajes de error, datos confidenciales que pueden ser aprovechadas por los usuarios maliciosos. Se recomienda no incluir en los mensajes de error información confidencial en caso de no requerirse. Asegurarse que los registros tengan un tiempo límite de permanencia.[75]

17.2.8. Proteccion de datos.

Se definen como puntos muy importantes y de mayor relevancia a nivel de protección de datos como lo son: confidencialidad, integridad y disponibilidad los cuales deben estar soportado por un sistema y servidor de confianza; cuando por alguna razón se guarde información privada de clientes o de otra índole en dispositivos tecnológicos llámese pc, Smartphone o tabletas, se debe garantizar que la información quede encriptada y no debe ni intervenirse ni divulgarse.[76]

---

[74] OWASP.(agosto 2017) Error handling.{En línea}.Recuperado de: https://www.owasp.org/index.php/ASVS_V8_Error_Handling.
[75] OWASP.(agosto 2017) Cryptography.{En línea}.Recuperado de: https://www.owasp.org/index.php/ASVS_V7_Cryptography
[76] OWASP. (agosto 2017) Data protection.{En línea}.Recuperado de: https://www.owasp.org/index.php/ASVS_V9_Data_Protection.

Como se mencionó al inicio de este párrafo se debe asegurar que un sistema de información por lo menos debe cumplir como mínimo los requerimientos de seguridad: Confidencialidad, que tiene que ver con la no divulgación de información privada o sensible sin autorización; Integridad: proteger los datos de los ataques maliciosos y la disponibilidad, la cual dicta que la información debe estar siempre disponible a usuarios autorizados.[77]

17.2.9. Comunicaciones.

Para este requisito de seguridad lo que se debe garantizar una ruta confiable para el certificado de seguridad valido a nivel de capa de transporte. Utilizar protocolos cifrados y seguros al momento de realizar las conexiones externas e internas. Manejo adecuado de las claves públicas a nivel productivo y de backup. Verificar encabezados de HTTP, y utilizar os algoritmos de cifrado y protocolos potentes. [78]

17.2.10. Código Malicioso.

Se debe procurar poder encapsular o separar la actividad maliciosa para que no afecte la operación del sistema de información en su operación general. Se debe tener especial cuidado en que los sistemas de información sobre todo de proveedores de terceros, no tenga las llamadas puertas traseras, huevos de pascua o fuente oculto, dado que los hackers se pueden aprovechar de estos componentes para perpetrar su fechoría.[79]

17.2.11. Errores Logica de Negocio.

En tema de lógica de negocio se debe tener especial cuidado en los temas de ataques automáticos, que pueden darse por ejemplo envío continuado de información, o adicionar muchos contactos de manera incremental. En la lógica de negocio se ha considerado no dejar pasar por alto el tema de los ataques maliciosos

[77] OWASP.(agosto 2017) Data Protection.{En línea}.Recuperado de: https://www.owasp.org/index.php/ASVS_V9_Data_Protection.
[78] OWASP.(agosto 2017) Communications.{En línea}.Recuperado de: https://www.owasp.org/index.php/ASVS_V10_Communication.
[79] OWASP.(agosto 2017). Malicious code.{En Linea}.Recuperado de: https://www.owasp.org/index.php/ASVS_V13_Malicious_Code

que pueden derivar en suplantaciones, modificación de información, repudio o incremento de privilegios entre otros.[80]

17.2.12. Archivos y Recursos.

Especial cuidado se recomienda tener con archivos no confiables o de los cuales se desconoce su procedencia, es decir se debe enviar a un sitio diferente a la ruta principal y se deben suprimir los permisos. Otras consideraciones son: los caminos o direccionamientos solo deben estar definidos por las rutas que se especifiquen en las listas blancas, para casos que no estén allí se debe generar una notificación de alerta para que se proceda a aislar estos recursos; Incluir escaneos con antivirus para archivos sobre los cuales se desconozca su procedencia o sean sospechosos. Los servidores de aplicaciones deben tener incorporados controles para no permitir el acceso desde software remoto que esté fuera del dominio del servidor de aplicaciones.[81]

17.2.13. Configuracion, API y IOT.

Para el tema de API y Servicios Web que sean orientados REST o SOAP, deben contar con una autenticación óptima tanto para las sesiones como para los WS propiamente dichos, con verificación y validación de las entradas y los respuestas que se generan entre niveles. Se debe validar que XML o JSON estén completamente validados antes de dejar pasar la entrada de datos. Procura que los WS basados en REST estén blindados contra la falsificación de solicitudes utilizando por ejemplo cookie de doble vía, encabezados de solicitud de envío.[82]

En lo que tiene que ver con la configuración del sistema, se recomienda tener las bibliotecas y parámetros actualizados y cada archivo de cada aplicación debería estar en la carpeta correspondiente a cada aplicativo, borrando archivos y subdirectorios que no se requieran, así como contar con una configuración defecto pero con la suficiente seguridad, controlando adecuadamente los cambios que los usuarios realicen sobre la configuración, es decir deben permitir cambios pero no

[80] OWASP.(agosto 2017). Business logic flaws. {En línea}.Recuperado de: https://www.owasp.org/index.php/ASVS_V15_Business_Logic_Flaws
[81] OWASP. (agosto 2017). Files and Resourses. {En línea}.Recuperado de: https://www.owasp.org/index.php/ASVS_V16_Files_and_Resources
[82] OWASP. (agosto 2017).API. . {En línea}.Recuperado de: https://www.owasp.org/index.php/ASVS_V18_API

dejar vulnerabilidades abiertas producto de ellos. Garantizar que la comunicación e integración entre el servidor y la BD estén encriptados.[83]

En el internet de las cosas, cada dispositivo debe utilizar seguridad a nivel de la capa de transporte y los datos que se almacenen se deben hacer de forma segura y encriptados; las interfaces o puertos USB deben estar deshabilitados. Verificar que existan controles de seguridad para evitar la ingeniería inversa y permitir que los controles a nivel tecnológico de los fabricantes estén habilitados.[84]

17.3. OWASP Top 10 Vulnerabilidades

Existe también dentro de la fundación OWASP[85] un top 10 de vulnerabilidades de seguridad para aplicaciones Web que se viene actualizando de acuerdo a cómo evolucionan las amenazas y vulnerabilidades. La última versión es del año 2017,

Dentro de los principales riesgos en seguridad de sistemas de información definidos por OWASP[86] se encuentran:
Problemas de Inyección (A1).
Perdida de autenticación y Gestión de Sesiones (A2).
Exposición de Información Sensible (A3).
Entidades Externas XML (XXE) (A4).
Pérdida de control de acceso (A5).
Configuración de Seguridad Incorrecta. (A6).
Comando de secuencias de sitios cruzados XSS (A7).
Deserialización insegura (A8).
Utilización de componentes con vulnerabilidad conocidas (A9).
Insuficientes Registros y monitoreo (A10).

[83] OWASP. (agosto 2017).Configuration. En línea}.Recuperado de: https://www.owasp.org/index.php/ASVS_V19_Configuration
[84] OWASP. (agosto 2017). . {En línea}.Recuperado de: https://www.owasp.org/index.php/ASVS_V20_Internet_of_Things
[85] OWASP. (2017).(agosto 2017) The open Web Aplication Security Project, Los 10 riesgos más críticos en aplicaciones Web.{En línea}.Recuperado de : http://www.owasp.org/index.php/Top_10
[86] Ibid., Pag. 7.

17.3.1. Inyección.

Cuando data que no es de procedencia conocida, proveniente de un atacante, es trasmitida como una consulta a SQL, LDAP u OS, y genera una falla conocida como Inyección y donde el objetivo del atacante es burlar al interprete para acceder fraudulentamente a información clasificada y privada. [87]

Los huecos de seguridad tipificadas como Inyección, normalmente se presentan cuando los analistas de software desarrollan querys dinámicos sobre las BD, donde el usuario es el que incluye los inputs

Para determinar que un sistema tiene vulnerabilidad a la inyección, se debe inspeccionar que las consultas que las son interpretadas por deben tener una brecha que los separe de la información o data que no es confiable.

17.3.1.1. Amenazas de Aplicación

Una amenaza puede provenir de e cualquier usuario, que puede ser persona que conoce o no del sistema de información y puede transmitir datos poco fiables al sistema con el ánimo de beneficiarse a través de un ataque deliberado. Los hackers proceden con el ataque tomando basándose en cadenas de texto muy triviales buscando réditos en el destino engañando el intérprete de sintaxis. El origen de un ataque de inyección puede ser cualquier fuente de datos.Se configura un ataque por inyección cuando una fuente envía cadenas de datos maliciosos al traductor de sintaxis. Los Errores de inyección son a menudo encontrados en código con herencia, como por ejemplo en querys SQL, LDAP, Xpath o No Sql o también instrucciones del SO, examinadores de XML o SMTP. Contrariamente a lo que se cree, este tipo de vulnerabilidad no es fácil detectarla en pruebas, solo cuando se revisa minuciosamente el código fuente se pueden detectar las anomalías. El impacto puede ser impredecible ante un ataque de inyección; puede presentarse

[87] OWASP. (2017).(agosto 2017) The open Web Aplication Security Project, Los 10 riesgos más críticos en aplicaciones Web.{En línea}.Recuperado de : http://www.owasp.org/index.php/Top_10.

perdida de información, daño de datos o incluso hasta el “secuestro” del sistema por parte del atacante.[88]

17.3.1.2. Recomendaciones

Los datos son la vida del sistema de la empresa, si un dato es dañado, robado o secuestrado, el prestigio de la empresa estaría en juego y toda la actividad de negocio estaría seriamente amenazada y en riesgo para el futuro de la empresa.
Las recomendaciones para combatir los ataques por inyección, se pueden dividir en Primaria y Secundarias.

17.3.1.3. Defensas Primarias.

17.3.1.3.1. Consultas Parametrizadas.

Los querys o consultas parametrizadas no son difíciles de desarrollar y lo primero que los analistas de software deben tener en cuenta es la definición del SQL y después la construcción de los parámetros que pasan a la consulta. Esta metodología hace que el motor de base de datos pueda identificar claramente que es Dato y que es código independizando al usuario y su origen de información.
Cuando un hacker intenta vulnerar el sistema se encuentra con que no puede realizar su maniobra por la disposición del software el cual no permiten que el intruso guie la consulta, por ejemplo, en el caso más típico si intenta inyectar el ID de Pedro ‘o’ 2’=’2, el sistema no se vulnera dado que el el query parametrizado intentaría buscar toda la cadena.[89]
A continuación, se despliega una muestra del manejo de una implementación de Java de consulta paramétrica sobre una base de datos.

```
String nombrecliente = request.getParameter ("nombreCliente");
 // Se debe validar la entrada para identificar  ataques
 String Consulta = "SELECT Saldo_clientes FROM Tabla_clientes WHERE
nombre_Cliente =?";
 PreparedStatement pstmt = connection.prepareStatement (consulta);
 pstmt.setString (1, nombreCliente);
```

[88] OWASP. (2017).(agosto 2017) The open Web Aplication Security Project, Los 10 riesgos más críticos en aplicaciones Web.{En línea}.Recuperado de : http://www.owasp.org/index.php/Top_10
[89] OWASP. (2017) OWASP Cheat Sheet: Injection Prevention. {En línea}. Recuperado de: https://www.owasp.org/index.php/Injection_Prevention_Cheat_Sheet.

```
ResultSet resultado = pstmt.executeQuery ();
```

17.3.1.3.2. Procedimientos Almacenados.

No necesariamente los procedimientos almacenados son inmunes al ataque por Inyección, pero si se realiza su implementación teniendo en consideración algunos estándares, por ejemplo, evitando utilizar Sqls dinámicos, se podría tener un resultado muy parecido al de las implementaciones por consultas parametrizadas, dicho de otra manera, los programadores deben tener en cuanta cuando construyen las consultas con parámetros los deben diseñar para que se parametricen de forma automática. La gran diferencia con las consultas paramétricas es que los procedimientos almacenados, se definen y almacenan en la base de datos y es invocado desde un aplicativo. Ambas técnicas tienen el mismo comportamiento para evitar el ataque de inyección. Para el caso específico de la herramienta SQL Server, para el manejo de Store Procedure en el servidor, tiene existen 3 perfiles de lectura, escritura y propietario, pero los procedimientos requieren permisos de ejecución por ende los administradores asignan privilegios de propietarios a todos los SP, generando un riesgo inherente a los accesos a través de los mismos porque los hackers podrían entrar al servidor a través de un SP y tener acceso total con las concedidas consecuencias. Se recomienda que para acceder a las consultas SQL, se deben manejar con variables y en lo posible con Store Procedures. Se deben evitar las consultas dinámicas. Se pueden utilizar herramientas para comprobar la vulnerabilidad del sistema. Para prevenir un ataque por inyección se recomienda tener claramente separado la data no fiable de los comandos. El uso de Api en vez de intérprete de comandos es una opción que sirve como método de prevención. [90]

El siguiente es un ejemplo donde se realiza desarrollo en Java a través de procedimiento almacenado para procesar una consulta en la DB, el procedimiento sp_ObtenerSaldocuenta, se debe definir en la base de datos.

```
String nombrecliente = request.getParameter ("nombreCliente");
 try {
      LlamadaSentencia sentencia = connection.prepareCall("{call
sp_ObtenerSaldoCuenta(?)}");
      sentencia.setString(1, nombrecliente);
      ResultSet resultados = sentencia.executeQuery();
```

[90] OWASP. (2017) OWASP Cheat Sheet: Injection Prevention. {En línea}. Recuperado de: https://www.owasp.org/index.php/Injection_Prevention_Cheat_Sheet..

```
      // … conjunto de Resultados
 } catch (SQLException er) {
      // … Seccion de manejos de errores
 }
```

17.3.1.3.3. Verificación Entrada listas blancas

Para los casos en que las consultas requieran por ejemplo nombres de tablas, columnas es conveniente utilizar la validación de listas para la entrada del usuario, pero en el caso que estas mismas consultas utilicen los parámetros para crear nombre de tablas o columnas, es necesario realizar una reingeniería del sistema porque no es aconsejable llevar a cabo este manejo.[91]

En el siguiente ejemplo se valida el nombre de una tabla.

```
String NombreTabla;
 switch(PARAMETRO):
   case "Value1": tableName = "EjemploNombreTabla";
                  break;
   case "Value2": tableName = "EjemploNombreTabla";
                  break;
    ...
   default      : throw new InputValidationException("Valor Errado o
no esperado para el nombre de la tabla
```

17.3.1.3.4. Escape de entradas

En la eventualidad de no poder utilizar ninguno de los anteriores defensas, se puede acudir a la técnica para dar escape del input que esa digitando el usuario previo al envío de la consulta. El modo de funcionamiento de esta defensa empieza desde que el manejador de la base de datos permite un juego de caracteres para cada tipo de consulta el Sistema manejador de DB identificara claramente la entrada y el código SQL. Se deben utilizar sintaxis de escape para las Apis que no estén parametrizadas. En caso de que las Api parametrizadas no esté disponibles se puede hacer uso del codificador OWASP de java que proveen librerías de escape. El Proyecto codificador de Java es una herramienta que ayuda a través de la codificación contextual, a que el código java no se vea afectado por inyection con

[91] OWASP. (2017) OWASP Cheat Sheet: Injection Prevention. {En línea}. Recuperado de: https://www.owasp.org/index.php/Injection_Prevention_Cheat_Sheet.

Cross Site Scripting(XSS) el cual se puede insertar a través de script que engañan por su apariencia de legalidad. Solo se debe descargar el codificador-1.2.1.jar y dar la instrucción “import org.owas.encoder.Encode” y listo ya se puede contar con la herramienta y sus librerías. Si se va a utilizar intérpretes se debe considerar la siguiente recomendación: validar las entradas de información por ejemplo validar contra el tamaño del campo, reglas del negocio, tipos de dato y todas las validaciones que permitan filtrar posibles ataques antes de almacenar o visualizar la información. Rechazar los datos que no surtan las validaciones y no intentar corregir o arreglar datos errados. En cuanto a los privilegios se recomienda dar los mínimos necesarios para acceder a la BD. Cuando se presente error, no se debe detallarlos porque estos son analizados y utilizados por el agresor. Especial atención a las funciones Exec() cuando se esté utilizando procedimientos almacenados. [92]

No considerar el uso de consultas dinámicas. Utilizar funciones de escape más fuertes. Se recomienda por ejemplo usar mysql_real_escape_string() en caso de que se esté usando base de datos MYSQL. No utilizar funciones de reemplazo como lo es str_replace(“””,””).

17.3.1.4. Ejemplos adicionales por ataque.

Cuando un sistema utiliza datos que no dan confianza cuando invocan un query a través de SQL>

Figura 18. ***Ejemplo 1 Ataque A1***

```
string query = " Select * from tablaClientes where
Identificacion = '" + request.getParameters ("ident")
+ "'";
```

Fuente: Autor

O utilizando un Entorno de desarrollo como Hibérnate:

[92] OWASP. (2017) OWASP Cheat Sheet: Injection Prevention. {En línea}. Recuperado de: https://www.owasp.org/index.php/Injection_Prevention_Cheat_Sheet.

```
Consulta HLQuery=sesión.CreateQuery (from tablaClientes where
Identificacion = '" + request.getParameters ("ident")+ "'")
```

En los 2 casos el hacker modifica a través de cualquier navegador Web, el parámetro ident por ‘o' 1 '=' 1, así:

```
http://ataque.com/VistadeCuentas?ident= 'or' 1 '=' 1
```

El sentido en las consultas se modifica y puede devolver todas las filas de la tabla de clientes. En algunas ocasiones los ataques pueden llegar a cambiar la información en la base de datos.

17.3.2. Pérdida de Autenticación y Gestión de Sesiones.

La manera más fácil de poder comprobar a través de un mapeo que un sistema tiene esta vulnerabilidad o realizando un control de privilegios de acceso a los datos por parte de los usuarios autorizados. Por otro lado, se debe también tener el mismo control para cuando la petición llegue desde una función que no sea pública. [93]

17.3.2.1. Amenazas de Aplicación.

Los huecos de seguridad son comúnmente explotados por los hackers en los programas de autenticación o de sesión, los cuales no son desarrollados con los suficientes controles de seguridad. Estos ataques pueden llegar a comprometer contraseñas o los usuarios pueden ser suplantados para la configurar el robo de información. Se debe cuestionar si los privilegios que tienen los usuarios son los adecuados y no están por fuera de sus perfiles de actualización de la base de datos, esta labor se debe hacer en conjunto con el DBA y el especialista de Seguridad.[94]

Los intrusos que llegan en un momento dado a poder acceder al sistema, lo hacen con perfiles autorizados obviamente cambiando los parámetros para poder acceder a privilegios de usuarios autorizados.

[93] OWASP. (2017) Owasp Top 10 2017 Los 10 Riesgos más críticos en Aplicaciones Web. {En línea}. Recuperado de: https://www.owasp.org/images/5/5e/OWASP-Top-10-2017-es.pdf. Pag. 6.
[94] OWASP. (2017) Owasp Top 10 2017 Los 10 Riesgos más críticos en Aplicaciones Web. {En línea}. Recuperado de: https://www.owasp.org/images/5/5e/OWASP-Top-10-2017-es.pdf. Pag. 8.

La falta de validación por parte de los APIS al momento de la petición de acceso de un usuario no autorizado, es la causa más común para que se viole la seguridad sobre el acceso al sistema. Normalmente los nombres de las funciones y los nombres del Localizador Uniforme de Recursos, son fáciles de vaticinar por los intrusos dado que ellos utilizan técnicas para identificar estas deficiencias.

Estad deficiencias pueden ser tan vulnerables que dado el caso pueden comprometer seriamente la seguridad de los datos o la base de datos, llegando incluso al robo o secuestro de la información. El impacto puede ser tan grande como lo podamos imaginar, pues si un intruso mal intencionado se apropia o roba la información de los clientes de una empresa o entidad, la reputación y la información financiera y privada de los mismos queda a merced del hacker y por ende el futuro de la compañía queda comprometido.

17.3.2.2. Recomendaciones.

Para prevenir esta vulnerabilidad, se deben tener en cuenta los siguientes aspectos: Que el cifrado de las claves de acceso de los usuarios estén debidamente resguardadas al momento de su almacenamiento. Evitar mediante protocolos de seguridad, que las llaves o claves sean fáciles de vaticinar por parte de los hackers. Que los identificadores de sesión no se expongan en la URLs. Que sea de conocimiento de los analistas desarrolladores de software una política robusta de gestión de sesión, donde se tengan todos los controles de autenticación, así como una herramienta para emular los accesos.
Evitar la suplantación de identidad a través de XSS.

Se recomienda que el nombre de usuario no distinga entre mayúscula y minúscula y deberían ser únicos, Las contraseñas deben ser de más de 10 caracteres que lo habitual, así serán más seguras. Las contraseñas deben tener la siguiente composición un carácter Mayúscula, 1 carácter minúscula, al menos 1 carácter numérico y al menos un carácter considerado como especial (&/$#). Como minino 10 caracteres como máximo 128; no repetir 2 caracteres de forma consecutiva y el sistema debe forzar a que los usuarios generen periódicamente diferentes topologías de contraseñas. Se debe permitir ingresar a los usuarios por lo menos 20 caracteres para la contraseña, evitando al máximo la práctica de truncamiento.

Definir una clara y consistente política de contraseñas, la cual debe estar disponible y publicada. En caso de requerirse se debe contar con un esquema de recuperación

de contraseñas. Las claves se deben transmitir utilizando un medio de transporte robusto, por ejemplo, la TLS (Transport Layer Sheet) o autenticación bidireccional, es decir involucra navegador y servidor y apoyados con el certificado de alguna autoridad Certificadora o CA.

Existen protocolos de autenticación contra un servidor y que no requiere incluir contraseña como el protocolo Oauth el cual utiliza token apoyado en unos procesos de autorización donde se identifica claramente el usuario utiliza el sistema independientemente de la plataforma de donde se conecta. La versión más recomendada es la 2.0 que trabaja con HTTPS. Otro protocolo de autenticación muy recomendado y utilizado con frecuencia es el Openld, el cual es un protocolo muy liviano que ofrece inicio de sesión único SSO, la que entrega identidad única de confianza, utilizando de intermediario al proveedor de Identidades. El protocolo SAML es el competidor de Openld y ofrece funcionalidades similares, la única diferencia con Openld es que SAML se basa en XML y es más flexible. [95]

El protocolo Openld tiene el segmento del mercado de consumo mientras que el protocolo SAML se enfoca especialmente en el segmento empresarial, por ejemplo, SAP ERP y SharePoint utilizan SAML versión 2.0[96]

Existen otros protocolos como el UAF y el U2F, el primero se enfoca en autenticación sin la contraseña y el segundo admite un segundo factor de autenticación. Los dos se fundamentan en un prototipo en criptografía de clave pública. UAF puede utilizar técnicas de seguridad como sensores, biometrías faciales y de voz, huellas. U2F por el contrario se enfoca en contraseñas con manejo de Token de hardware como por ejemplo con dispositivos USB, donde se almacenan las claves de autenticación que son utilizadas para la firma. Este protocolo es un aliado muy fuerte en la protección contra el ataque de pishing.

17.3.2.3. Ejemplos

Existen aplicaciones comerciales que permiten la Reescritura de URL, la cual es aprovechada incluyendo la identificación de la sesión dentro de ella. Es posible que

[95] OWASP. (2017) Owasp OWASP Cheat Sheet: Authentication. {En línea}. Recuperado de:.https://www.owasp.org/index.php/Authentication_Cheat_Sheet
[96] OWASP. (2017) Owasp OWASP Cheat Sheet: Authentication. {En línea}. Recuperado de:https://www.owasp.org/index.php/Authentication_Cheat_Sheet.

se vaya dentro del número de sesión toda la información de la transacción incluyendo la información financiera y datos personales. Cuando se utilizan un equipo de cómputo en un café internet por ejemplo y se cierra la pestaña del navegador, los datos del usuario continúan en él y es aprovechado por hackers. Robo de contraseñas que no están adecuadamente encriptadas.

17.3.3. Secuencia de Comandos en Sitios Cruzados (XSS).

Es un agujero de seguridad que posibilita a un tercero inyectar en sitios web que habitualmente visitan los usuarios (posibles víctimas), código fuente, normalmente en JavaScript para eludir los controles del sitio y poder robar información de la víctima. La amenaza puede provenir de individuos de cualquier índole que envíen información no real y no fiable a los sistemas de información, incluso usuarios del sistema.[97]

17.3.3.1. Vectores de ataque y Fragilidad de seguridad.

Los huecos de seguridad producidos por el Cross Site Scripting, se pueden producir cuando un sistema actualiza una web que ya ha sido controlada por un intruso. Se puede presentar a nivel de la aplicación servidor o aplicación cliente.
Los agresores a través de guiones de código fuente pueden aprovecharse de las debilidades de los intérpretes dentro de exploradores de la web. Otro vector de ataque puede provenir del interior del sistema, por ejemplo, usuarios corruptos que aprovechas sus privilegios para realizar los ataques. Existen 3 formas de ataque de XSS: El reflejado, almacenado y basado en DOM.

XSS[98] reflejado y almacenado son vulnerabilidades de inyección que afectan el servidor, es decir el código malicioso en la aplicación, pero, solo en el servidor cuando se realiza el procesamiento de la solicitud se genera la agregación del HTML. Mientras que el ataque XSS basado en DOM inyecta el código en la aplicación en tiempo de ejecución del cliente es decir en el navegador.

---

[97] OWASP. (2017) Owasp OWASP Cheat Sheet: Authentication. {En línea}. Recuperado https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
[98] OWASP. (2017) OWASP XSS (Cross Site Scripting) Prevention Cheat Sheet.{En línea}. Recuperado de: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet.

Todos los 3 tipos de ataques pueden llegar a tener una afectación técnica importante y de grados diferentes sobre las empresas porque en la práctica se deriva en enrutamiento de usuario, secuestro de explorador de la web utilizando malware, inserción de información maliciosa, desaparecer sitios web entre otros.

La afectación de la información de datos e información privada de los clientes, supone una exposición de la compañía muy seria, así como su continuidad en el mercado.

17.3.3.2. Recomendaciones.

Si los aplicativos de la empresa no tienen controles de escape de contexto sobre todo en la parte del servidor y reciben directamente la entrada del cliente, podemos afirmar que el sistema está expuesto a ataques de tipo XSS.

Utilizar las verificaciones de entrada del usuario y escapes adecuados en caso de fallos o intentos de ataque que se compongan de HTML en el lado del servidor asi:
De ninguna manera ingrese información vulnerable o que atente con la seguridad del sistema. Si estamos en un cliente o servidor basado en HTML o JavaScript se debe escapar antes de recibir los datos maliciosos.
Utilizar JSON como alternativa las reglas de Codificación y escape.
Utilizar las reglas basados en DOM.
Realizar periódicamente sobre todo es sistemas que se basan en ActiveX, Flash, Silverligth, pruebas de pentest y revisión manual del código para detectar este tipo de vulnerabilidad. Utilizar configuración del Proyecto OWAS Sanitzer (Desinfección), para implementar políticas de validaciones de información sobre HTML.[99]

Otras reglas adicionales que pueden ayudar a bloquear un ataque de XSS:
Por ninguna razón inserte información de dudosa procedencia y que no genere confianza;se debe en lo posible denegra todo y solo lo permitir lo que se tiene certeza en cuanto sitios permitidos. Dar Escape en aquellos escenarios que intentan insertar contenidos, atributos, código javascript no fiables. Dar Escape antes de insertar parámetros de URL no confiables.

[99] OWASP. (2017) OWASP XSS (Cross Site Scripting) Prevention Cheat Sheet.{En línea}. Recuperado de: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet.

17.3.3.3. Ejemplos.

Ejemplo de Html vulnerable.

```
element.innerHTML = "<HTML> Etiqueta";
document.writeln ("<HTML> Etiqueta");
```

Para evitar esta esta vulnerabilidad se recomienda utilizar el HTML y luego codificación JavaScript.

```
element.innerHTML = "<% = Encoder.encodeForJS
(Encoder.encodeForHTML (untrustedData))%>";
document.writeln ("<% = Encoder.encodeForJS (Encoder.encodeForHTML
(untrustedData))%>");
```

Escape JS (JavaScript).

```
  var x = document.createElement ("input");
  x.setAttribute ("nombre", "Nombre_Empresa");
  x.setAttribute ("value", '<% = Encoder.encodeForJS
(NombreEmpresa)>');
  var form2 = document.forms {0};
  form2.appendChild (x);
```

Esto evita que, si el nombre de la empresa tiene un carácter especial como por ejemplo “&”, el intruso puede aprovechar esta situación para perpetrar su ataque.

17.3.4. Entidades Externas XML (XXE).

Cuando se accedan a procesadores de XML obsoletos o que están con configuraciones defectuosas, pueden ser aprovechados por los atacantes para descifrar archivos internos, URL o rutas y ejecutar código remotamente y realizar ataque de denegación de servicio.[100]

17.3.4.1. Debilidades de Seguridad Vectores de Ataque.

Por default gran parte de los procesadores XML obsoletos dan la posibilidad de especificar una entidad externa y una URL que hace el procesamiento del XML.

[100] OWASP. (2013) Owasp Top 10 2017 Los 10 Riesgos más críticos en Aplicaciones Web. {En línea}. Recuperado de: https://www.owasp.org/images/5/5e/OWASP-Top-10-2017-es.pdf. Pag 10.

Existen herramientas de análisis de código fuente que permiten a los testers identificar esta vulnerabilidad. Esta vulnerabilidad facilita a los atacantes acceder a los datos de manera remota. Los impactos al negocio dependen del tipo de dato que se vulnera.

17.3.4.2. Afectación y Recomendaciones.

Los entornos a aplicaciones que son vulnerables a los ataques XXE son aquellos que utilizan entidades XML y se basan en SOAP versión menor a la 1.2. Las aplicaciones que se basan en SAML (the security Assertion Makup Language) para el procesamiento de entidades XML, también son vulnerables.[101]

Dentro de las recomendaciones para evitar XXE se tienen: utilizar formatos sencillos como JSON, actualizar las versiones de las bibliotecas XML; implementar listas blancas para la captura de información hacia el encabezado XML.Incluir validaciones a nivel del XSD para cuando se monten o se carguen archivos XML; utilizar herramientas de análisis de código fuente y como último recurso utilizar Cortafuegos de aplicaciones Web para el monitoreo e identificación de ataques XXE.[102]

Lo más efectivo para evitar ataque XXE es desactivas las entidades externas DTD así:

factory.setFeature ("http://apache.org/xml/features/disallow-doctype-decl", true);[103]

En language java:

```
xmlInputFactory.setProperty (XMLInputFactory.SUPPORT_DTD, false); // Deshabilita totalmente la funcionalidad entidades externas DTD para esta fabrica.
xmlInputFactory.setProperty ("javax.xml.stream.isSupportingExternalEntities", false); // DESHALIBILITA ENTIDADES EXTERNAS [104]
```

[101] Ibid., Pag. 10.
[102] OWASP. (2017) Owasp Top 10 2017 Los 10 Riesgos más críticos en Aplicaciones Web. {En línea}. Recuperado de: https://www.owasp.org/images/5/5e/OWASP-Top-10-2017-es.pdf. Pag 10.
[103] OWASP. (2017). XML External Entity (XXE) Prevention Cheat Sheet {En línea}. Recuperado de: https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet
[104] OWASP. (2017) XML External Entity (XXE) Prevention Cheat Sheet {En línea}. Recuperado de https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet#General_Guidance

17.3.5. Configuración Errada de Seguridad.

La base de tener una Seguridad adecuada Configuración tanto del sistema operativo como de los programas, servidor de aplicación o web, plataformas y demás componentes del sistema, claro está que se sobreentiende que se debe estar permanentemente actualizando y manteniendo esta configuración para garantizar que el sistema este lo más alejado de la configuración por default y blinde la seguridad de posibles ataques. Todo lo anterior también aplicar para el software base y de S.O.[105]

La vulnerabilidad se puede originar por ejemplo desde la pila del sistema, un servidor de Aplicaciones o WEB o repositorio de base de datos. Es responsabilidad de todo el grupo de desarrollo del proyecto y de los analistas de seguridad, velar porque todas las configuraciones del sistema queden de manera correcta e idónea. Se pueden apoyar en herramientas tecnológicas como los scanner para identificar errores o configuraciones defectuosas del sistema.

17.3.5.1. Amenazas y Vectores de ataque.

Debemos tener en cuenta que los ataques pueden provenir del exterior del sistema como usuarios anónimos, pero lo más desafiante es que también puede provenir del interior del sistema utilizando sus nombres de usuarios y credenciales internos. Los vectores identificados para esta vulnerabilidad son las cuentas predeterminadas por default, fallos no atendidos ni solucionados, directorios y archivos desprotegidos. El software y el sistema pueden estar vulnerable sin que los administradores del sistema se hayan percatado de ello generando un posible robo continuado de información lo que podría generar un costo enorme para la compañía.

17.3.5.2. Recomendaciones y ejemplos.

[105] OWASP. 20117. Access Control Cheat Sheet . {En línea}. Recuperado de: https://www.owasp.org/index.php/Access_Control_Cheat_Sheet

Podemos identificar si un sistema se encuentra desprotegido si al hacernos el cuestionario, alguna de sus preguntas resulta afirmativa: ¿Se tiene software desactualizado? Esto incluye software de Base, SO, servidores de DB, WEB. ¿Se han realizados configuración que no son requeridas? Como por ejemplo privilegios, permisos, puertos, servicios entre otros.

¿Tiene cuentas default desde la instalación del sistema sin modificaciones?
Para evitar ser presa de los atacantes por tener de manera inadecuada la configuración de seguridad se recomienda contar con ambientes de producción, desarrollo y pruebas, nivelados en cuanto a la configuración, con credenciales que se diferencien para cada uno de los ambientes. Esto garantizaría que cada vez que se libere una versión a producción la configuración de seguridad sea la misma que se probó en desarrollo o pruebas. Tener el personal competente para estar pendiente de las nuevas versiones de sistemas y software y realizar el mantenimiento y configuraciones al mismo. Disponer de un diseño de arquitectura modular donde se identifiquen claramente los componentes y las integraciones del sistema. Realizar de manera permanente auditoria y escaneos para identificar posibles nuevos Huecos De seguridad. Las claves de acceso deben guardarse en formato que no se pueda revertir, ejemplo SHA-256 o por el estilo. Se recomienda no almacenar información financiera por ejemplo de Tarjetas de Crédito, solo si el sistema está certificado PCI se estaría en capacidad de hacerlo. Se recomienda utilizar por ejemplo SSL, SSH como método de encriptación para no ser víctimas de intervención de los datos en el medio de la comunicación.[106]

El sistema debería permitir cambiar las password regularmente. Los siguientes son ejemplos de este tipo de vulnerabilidad: Si un administrador deja que el instalador del aplicativo o sistema genere por default algunas cuentas sin cambiar nada. Este escenario genera vulnerabilidad puesto que los atacantes pueden identificar fácilmente que hay cuentas configuradas default y procede con el ataque, apoderándose de las cuentas accediendo al sistema para robar la información.

En muchas ocasiones los servidores devuelven mucha información de tracking de los accesos de los usuarios, lo que genera una exposición muy vulnerable a esta información. Esta información adicional que los mensajes de error o de warning desde los servidores son aprovechados por los hackers para apropiarse del sistema.

[106] OWASP. 2017. Access Control Cheat Sheet . {En línea}. Recuperado de: https://www.owasp.org/index.php/Access_Control_Cheat_Sheet

17.3.6. Exposición Datos vulnerables o Sensibles.

Dentro del análisis que se debe realizar el especialista de seguridad informática debe considerar si el sistema es vulnerable a la exposición de datos sensibles como por ejemplo las claves de acceso, numero de cuentas bancarias, Tarjetas de crédito y en general la información personal de clientes o usuarios. Debe considerarse a quien le interesaría poseer la información de datos confidenciales de clientes y copias de seguridad de la información. Lo más habitual es que los intrusos primero se apropien de las claves, realizar ataques de hombre en el medio y no se enfoquen en tratar de descifrar criptografía. Los atacantes aprovechan el pobre mantenimiento de claves, como lo es la debilidad de las contraseñas, debilidad en los navegadores. Para identificar si el sistema está expuesto a esta vulnerabilidad debo tener en cuenta: ¿El sistema almacena y trasmite información sensible y privada en texto sin encriptar?; ¿Los algoritmos de encriptación del sistema son antiguos y obsoletos?;¿Se consideran débil la administración de contraseñas o la gestión de cambio de las mismas?[107]

Una falla de este tipo es considerada muy grave y generalmente comprometo toda la integridad de los datos y por ende de las compañías, dado que incluyen información personal como son número de registro de información privada, Tarjetas de Crédito. Esta información es muy apetecida por los intrusos por el valor que tienen los mismos. Esto puede genera un daño reputacional para la compañía y temas legales por demandas que pueden ser catastróficos. Esta vulnerabilidad es dinámica y creciente frente al tema de los ataques a la información crítica y sensible de las empresas, pero lo mínimo que se debería tener en cuenta para tratar de minimizar el riesgo de este tipo de ataques que puede provenir del exterior o el interior de la organización. Se debe garantizar la encriptación de datos sensible y privados ya sean estáticos o en tránsito. En lo posible no se debe guardar información privada que no sea necesaria. De Tenerse se debe borrar o desechar para que no sea objeto de robo.[108]

Garantizar el uso de claves fuertes y con una administración robusta. No permitir el “autocompletar” en las plantillas de ingreso de información privada y confidencial al

[107] OWASP. (2017) Owasp Top 10 2017 Los 10 Riesgos más críticos en Aplicaciones Web. {En línea}. Recuperado de: https://www.owasp.org/images/5/5e/OWASP-Top-10-2017-es.pdf. Pag. 9
[108] OWASP. (2017) Owasp Cheat Sheet: Transport Layer Protection. {En línea}. Recuperado de: https://www.owasp.org/index.php/User_Privacy_Protection_Cheat_Sheet

sistema. Un escenario en donde se evidencia una vulnerabilidad de exposición de datos sensibles, puede ser un sistema financiero de que tiene cifrado del número de tarjetas de crédito en una base de datos realizada automáticamente, obviamente el descifrado también debe ser automática. Este sistema si se ve abocado a un ataque por inyección SQL, se puede recuperar los números de las tarjetas por parte de los atacantes. Como método de solución, se sugiere el cifrado utilizando una clave publica en la base de datos y en el software back-end para descifrarlos con clave privada. Otras recomendaciones a modo general podrían ser: tener una configuración de servidor seguro, el cual debería incorporar las siguientes reglas; usar un protocolo fuerte para la transmisión de datos, ejemplo TLS; no mezclar contenido TLS y no TLS; Usar cookies seguras; Usar HTTPS; evitar el almacenamiento en cache de los datos sensibles, el manejo de clave pública; manejo de certificado del servidor y protocolos de cifrados muy fuertes.[109]

17.3.7. Falta Control nivel de Acceso.

Los sistemas de información normalmente no protegen las funciones de acceso de manera adecuada, sobre todo cuando se deja la responsabilidad del control a la configuración del sistema y éste está mal configurado o cuando se deja a los ingenieros de desarrollo ese trabajo, pero no le dan prioridad o lo pasan por alto.

17.3.7.1. Vectores de ataque y Fragilidad de Seguridad.

Los ataques como en otras vulnerabilidades pueden provenir de fuera o de adentro del sistema, es decir no solamente los intrusos pueden generar los ataques, usuarios internos con solo cambiar las urls y los externos por supuesto pueden apropiarse de funciones que no estén con protección, sobre todo las funciones administrativas. El impacto en la reputación puede llegar a ser muy alta en caso que se haga público el ataque y la vulnerabilidad. [110]

17.3.7.2. Recomendaciones y Ejemplos.

[109] OWASP. (2017) Owasp Cheat Sheet: Transport Layer Protection. {En línea}. Recuperado de: https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet

[110] OWASP. (2017) Owasp Top 10 2017 Los 10 Riesgos más críticos en Aplicaciones Web. {En línea}. Recuperado de: https://www.owasp.org/images/5/5e/OWASP-Top-10-2017-es.pdf. Pag. 11

Con el siguiente cuestionario se puede identificar que se tiene una vulnerabilidad de este tipo: ¿están incompletas las validaciones de autenticación en el servidor? ¿La GUI de usuario tiene funciones que no están autorizadas?.

Se pueden realizar análisis apoyándose en el proxy para identificar si las respuestas del servidor siguen siendo iguales con y sin modificaciones en el proxy. En caso afirmativo se tiene vulnerabilidad. El siguiente ejemplo pone en evidencia esta debilidad, en el caso que el atacante que no esté autenticado pueda llegar hasta una URL, ejemplo: http://servidorprueba/aplicacion/administracion_aplicacion, o en el caso de un usuario autenticado que pueda acceder a una página de administrador del sistema, sería una situación delicada que requeriría atención del personal de seguridad informática. Existen muchos controles de accesos, pero dentro de los principales podemos definir: Control de acceso RBAC: Este control define el acceso de acuerdo al rol de un usuario dentro de la compañía, para los que se define un nivel de acceso de acuerdo a su responsabilidad. Este método tiene la ventaja de poder basarse en el organigrama de las compañías dando una mayor facilidad de administrar y usar; está en concordancia con la directiva de seguridad "mínimos privilegios". Dentro de las desventajas tenemos: complejidad en el mantenimiento de los perfiles de cada usuario, dificultades cuando se debe manejar con directorio Activo y se imposibilita cuando el acceso es a partir de una base de datos. Control de Acceso DAC (Control de Acceso Discrecional): El acceso en este método va en función de la identidad y credenciales del usuario, como pueden ser los password o tokens a nivel de software o de hardware. Dentro de las ventajas se cuenta con sencillo de manejar y administrar, privilegios mínimos. La parte no tan positiva es que no se puede perder el mantenimiento de los roles y accesos, problemas con acceso por directorio Activo y por último se puede presentar en el tiempo otorgamiento de que permisos y accesos más de los que se requieren.[111]

Control de Acceso MAC: Este método garantiza la aplicabilidad de la política de seguridad de la información. Es recomendado para sistemas que requieren máxima seguridad. Ventajas: Accesos basado en sensibilidad de los objetos, se cumple al detalle el acceso mínimo, solo el administrador del sistema puede otorgar privilegios. Desventajas: implementación costosa y no es tan flexible.[112]

[111] OWASP. (2017) Owasp Access Control Cheat. {En línea}. Recuperado de: https://www.owasp.org/index.php/Access_Control_Cheat_Sheet#tab=Role_Based_Access_Control__28RBAC_29

[112] OWASP. (2017) Owasp Top 10 2017 Los 10 Riesgos más críticos en Aplicaciones Web. {En línea}. Recuperado de: https://www.owasp.org/images/5/5e/OWASP-Top-10-2017-es.pdf. Pag. 13

17.3.8. Falsificación de solicitudes entre sitios (2013).

Falsificación de solicitudes entre sitio o llamada CSRF (Cross Site Request Forgery), es un ataque cuyo propósito principal es incitar al usuario a ejecutar cosas no deseadas. Los tipos de ataques CSRF se hacen más impactantes de acuerdo al perfil de la víctima, si es un usuario con bajo perfil el impacto en la seguridad estaría en esa proporción, por el contrario, si es un usuario con privilegios de administrador el impacto sería mucho mayor. El éxito de este ataque se basa en la confianza que la web puede llegar a tener con el usuario, posible víctima del ataque; el hacker toma ventaja de esta situación y se apropia indirectamente de los privilegios del usuario, capitaliza el ataque y puede suplantar al usuario haciendo uso indebido de sus cuentas, en el ejemplo más crítico cuando el cliente está navegando en la página de su banco.[113]

Para considerar si un sistema puede sufrir un ataque de falsificación de solicitudes entre sitios, se debe revisar si las plantillas y formularios no poseen el token, o solicitar a través de un captcha para identificar si es un usuario real el que quiere enviar una solicitud.

17.3.8.1. Vectores de ataque y Fragilidad de Seguridad.

Cualquier persona o sitio o fuente puede montar contenidos en los navegadores de los usuarios y obligarlos a genera solicitudes a un determinado sitio WEB (quien sería el objetivo del ataque), donde el atacante genera solicitudes http maliciosas y no verdaderas para poder confundir a su víctima a través de XSS, si el usuario esta autenticado, se materializo el ataque. Un ataque de esta naturaleza puede afectar seriamente la información de los clientes generando un impacto comercial negativo y una directa afectación a la reputación de la empresa.[114]

17.3.8.2. Recomendaciones

[113] OWASP. (2017) Owasp Top 10 2017 Los 10 Riesgos más críticos en Aplicaciones Web. {En línea}. Recuperado de: https://www.owasp.org/images/5/5e/OWASP-Top-10-2017-es.pdf. Pag. 7.
[114] OWASP. (2017) Owasp Top 10 2017 Los 10 Riesgos más críticos en Aplicaciones Web. {En línea}. Recuperado de: https://www.owasp.org/images/5/5e/OWASP-Top-10-2017-es.pdf. Pag. 14.

Para evitar ser víctima de un ataque de esta naturaleza se sugiere incluir autenticación para las peticiones a través de un TOKEN (se recomienda incluirlo en un campo oculto en el cuerpo de solicitud de http y no incluido en url, porque lo expone peligrosamente) que no sea de fácil interpretación por cada solicitud de usuario al sistema y únicos por sesión, entre las técnicas más populares están:
Patrón de toquen sincronizado: Define que en el dado caso que una operación tenga un cambio de estado, se debe generar un token aleatorio y seguro como por ejemplo el token CSRF cuya característica es que se genere única por cada sesión de usuario, aleatoriedad y utilizar criptografía segura.

En caso de ser necesario, solicitar al usuario que se identifique de nuevo, utilizando por ejemplo captcha. Otras recomendaciones importantes para evitar el ataque CSRF, es: verificar el encabezado Standard: la idea es verificar que la solicitud provenga del mismo origen. Verificar la ficha de CSRF.
Token encabezado de cookie: este token se debe incluir oculto dentro de la trama y en caso de no superar las validaciones, se debe rechazar la solicitud.

Una alternativa para mitigar el ataque de CSRF es utilizar CSRFguard, la cual entrega una variación de patrón de token de sincronización para disminuir la posibilidad de ataques de falsificación de solicitudes entre sitios CSRF.

OWASP[115] nos da algunas directrices como coadyuvantes para prevenir del XSS:
La primera recomendación es: en lo posible denegar todo en cuanto sea posible, no se debe colocar datos que no sean de confianza y de ninguna manera se debe aceptar y ejecutar código JavaScript que provenga de una fuente no confiable.
La segunda recomendación es dar escape de HTML si desconoce o no hay confianza de los datos que se ingresaran.
La 3ª. Recomendación, es dar escapa de JavaScript antes de ingresar datos no confiables, sobre todo en código generado dinámicamente.
La 4ª. Recomendación es para valores de JSON, se debe garantizar que el encabezado content type que devuelve, sea application/json y no text HTML, para evitar que el browser vaya a ejecutar algún tipo de inyección.
5ª Regla: no codifique URL completas o relativas si la entrada no es de confianza.[116]

[115] OWASP. (2017) Owasp Cheat Sheet: XSS Prevention. {En línea}. Recuperado de: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
[116] OWASP. (2017) Owasp Cheat Sheet: XSS Prevention. {En línea}. Recuperado de: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet.

17.3.9. Utilizar Componentes con Vulnerabilidades.

Una realidad latente es que los desarrolladores utilizan bibliotecas de terceros que contienen vulnerabilidades que son ya públicas y conocidas.
Para identificar si un sistema es vulnerable a esta amenaza, debemos consultar las bases de datos y toda la información de proyectos que estén en curso.[117]

17.3.9.1. Vectores de ataque y Fragilidad de Seguridad.

Un atacante puede fácilmente detectar cuando un sistema es vulnerable solo con realizar un escaneo, utilizando un exploit exclusivo para cada víctima. La mayoría de los sistemas puedes tener esta vulnerabilidad dado que todos utilizan bibliotecas de proveedores que tienen vulnerabilidades y los analistas no dan suficiente prioridad a su revisión y actualización. Por esta vulnerabilidad se pueden presentar todas las posibilidades de ataque: inyección, XSS y control de acceso interrumpido entre otras. El impacto puede llegar a ser muy grande dependiendo de target del atacante y del ataque propiamente dicho. La forma de minimizar esta amenaza es en primer lugar no usar componentes que no se hayan desarrollado por el grupo del proyecto.[118]

Si definitivamente se deben utilizar librerías ajenas, se debe procurar realizar las actualizaciones a la última versión del software que seguramente deben estar blindados de los errores de seguridad. Los siguientes 2 ejemplos son típicos de esta vulnerabilidad:

Omisión de autenticación de apache: esta librería no entrega token de identificación, razón por la cual los atacantes podía acceder indistintamente servicios web con todos sus permisos. El otro es Código Remoto de Spring: El uso inadecuado de Lenguaje en las librerías de spring facilita que los atacantes se apropiaran del servidor usando cualquier tipo de código.[119]

---

[117] OWASP. (2017) Owasp Top 10 2017 Los 10 Riesgos más críticos en Aplicaciones Web. {En línea}. Recuperado de: https://www.owasp.org/images/5/5e/OWASP-Top-10-2017-es.pdf. Pag. 7.
[118] OWASP. (2017) Owasp Top 10 2017 Los 10 Riesgos más críticos en Aplicaciones Web. {En línea}. Recuperado de: https://www.owasp.org/images/5/5e/OWASP-Top-10-2017-es.pdf. Pag. 15.
[119] OWASP. (2017) Owasp Cheat Sheet: XSS Prevention. {En línea}. Recuperado de: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet.

17.3.10. Falta o Monitoreos insuficientes.

La usencia de Monitoreo sobre las aplicaciones es considerado como la acción que puede iniciar todos los ataques contra una vulnerabilidad en el sistema. Si no existe monitoreo los hackers pueden actuar a sus anchas sin ser detectados oportunamente. Podemos identificar que un sistema es vulnerable a esta incidencia cuando no se tiene registros de inicio sesión y sus fallos; también cuando no hay claridad de los mensajes de error; el sistema no puede detectar atacas en tiempo real y no existen alertas para prevenir o actual ante un incidente de seguridad.[120]

17.3.10.1. Recomendaciones

Las siguientes son las recomendaciones más relevantes para evitar ser víctima de este ataque: validar que se tenga registro de todos los inicios de sesión para el seguimiento y tracking al momento de presentarse un ataque. Garantizar que se tiene una trazabilidad de auditoria para las transacciones de alto impacto hacia el cliente. Se sugiere contar con un registro de aplicación coherente de acuerdo al sistema que se esté trabando; el objetivo de este registro va más allá de solo la traza de auditoria, con él se puede hacer seguimiento a un incidente de seguridad que se presente o que se haya presentado como apoyo al análisis forense. La fuente de este registro puede estar en múltiples contextos, por ejemplo, puede desde el firewall, del sistema cliente, desde el servidor de aplicaciones, sistema de base de datos incluso desde el sistema operativo. Los registros de eventos se deben alojar dependiendo del sistema, es decir si es un sistema de escritorio, se debería guardar en una carpeta con seguridad a nivel del S.O, en caso de una aplicación Web se debería guardar por ejemplo como archivos MIME y no estar disponibles para la web, y para el caso de sistemas de bases de datos se sugiere tener una cuenta exclusiva para guardar este registro. Lo que se debe registrar también depende del objeto del sistema, pero en general se puede registrar: Fallos en entradas y salidas de datos, fallos de autorización y administración de las sesiones, así como cambios de datos o en la configuración del sistema, también se pueden registrar transacciones que derivan una sospecha o que sean fraudulentas.[121]

---

[120] OWASP. (2017) Owasp Top 10 2017 Los 10 Riesgos más críticos en Aplicaciones Web. {En línea}. Recuperado de: https://www.owasp.org/images/5/5e/OWASP-Top-10-2017-es.pdf. Pag. 16

[121] OWASP. (2017) Owasp Cheat Sheet: XSS Prevention. {En línea}. Recuperado de: https://www.owasp.org/index.php/Logging_Cheat_Sheet#Purpose

18. ESTANDARES DE LA DIVISION CERT.

La División CERT del instituto SEI (Instituto de Ingeniería de Software), es la encargada de resolver los incidentes de ciber-seguridad y vulnerabilidades de seguridad del software nuevo y existente. La división CERT, también tiene dentro sus alcances el análisis de los huecos de seguridad, monitoreo de redes e investigación de las mejores prácticas de seguridad, así como coadyuvar a los fabricantes de software a solucionar vulnerabilidades de seguridad. La división CERT adicionalmente a detectar y reportar incidentes de seguridad también incursiona en seguridad cibernética, análisis forense, detección de vulnerabilidades y gestionar riesgos y **codificación segura**. [122]

Este trabajo se enfocará en el estándar de **codificación segura** de software para algunas de las reglas de mayor utilidad para la herramienta Java.

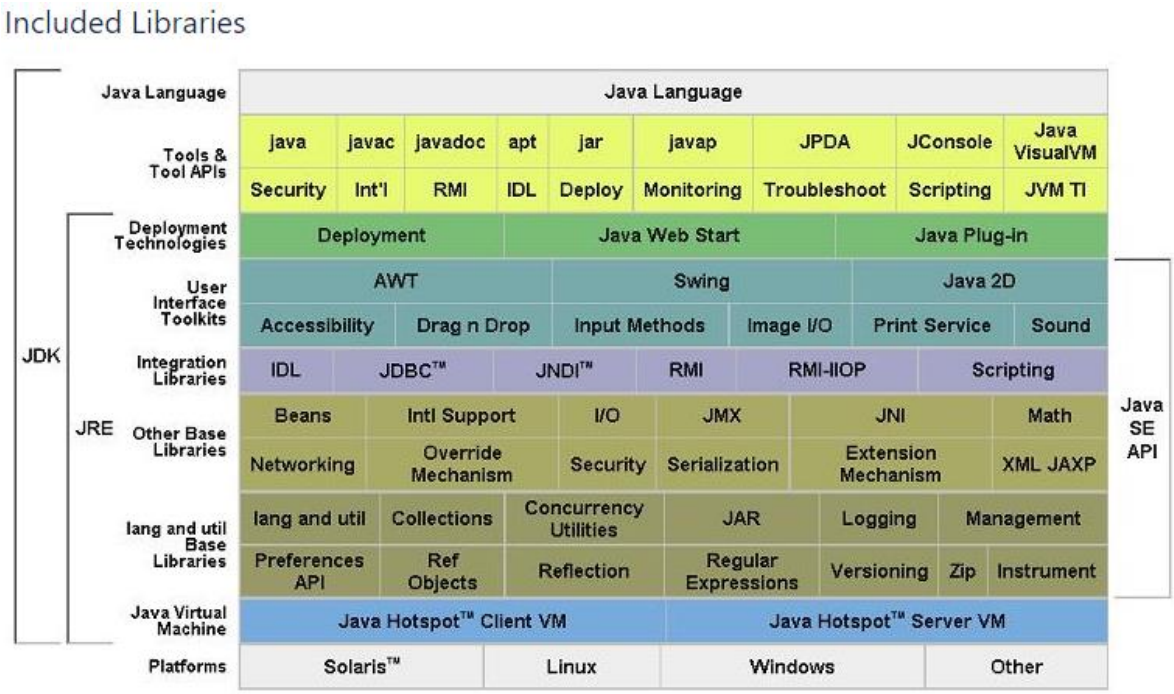
[122] Cornegie Mellon University. Software Engineering Institute (2018). {En Linea}. Recuperado de: https://www.sei.cmu.edu/about/divisions/cert/index.cfm.

18.1. SEI CERT CODIFICACION STANDARD PARA JAVA.

Corresponde a un trabajo que aun continua en proceso de construcción y no se cierra, dado la dinámica en los cambios y nuevas vulnerabilidades del software. [123]

Un elemento indispensable cuando se trata de programación segura, es contar con standard y reglas para la codificación segura del software, por eso el CERT cuenta con Oracle Secury Coding Estándar para Java. [124]

Figura 19 Librerias Incluidas en El CERT Oracle Secure Coding Standard para Java



Fuente: https://wiki.sei.cmu.edu/confluence/display/java/Rule%3A+Scope

18.1.1. Regla de Validación De entrada y depuración de Datos.

[123] MCMANUS José. (marzo 2018). SEI CERT Oracle Coding Standard for Java. {En línea}. Recuperado de: https://www.securecoding.cert.org/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java

[124]MOHINDRA Dhruv (febrero 2018).SEI CERT Oracle Coding Estándar for Java.{En linea}.Recuperado de:https://wiki.sei.cmu.edu/confluence/display/java/Rule.+Preface.

18.1.1.1. Prevención Injection SQL.

Estas vulnerabilidades se presentan en querys de aplicaciones de orígenes no confiables, lo que puede generar alteraciones en las consultas y robo de información de base de datos. La prevención más importante para evitar este ataque es la validación sobre los Store Procedure y los Query parametrizados, ya que por este medio es que se filtran la mayoría de las inyecciones.[125]

Por ejemplo, cuando un sistema autentica el logeo de usuarios en su sistema con la siguiente consulta:

```
SELECT * FROM db_usuario WHERE NombreUsuario='<USERNAME>' AND
                              Clave='<PASSWORD>'
```

Un intruso puede sustituir caprichosamente USERNAME, Y PASSWORD por cadena arbitraria:

```
validuser' OR '1'='1
```

Produciendo un dinamismo en la consulta de autenticación:

```
SELECT * FROM db_usuario WHERE NombreUsuario
='usuariovalido' OR '1'='1' AND Clave ='<PASSWORD>'
```

En caso que validuser es un usuario válido, esta línea de código insertaría una fila en la base de datos sin validación de la clave.
Otra variante seria:

```
SELECT * FROM db_usuario WHERE NombreUsuario='<USERNAME>' AND
Clave='' OR '1'='1'
```

La parte del OR siempre seria valida y devuelve el total de las filas de la tabla y su autenticación se daría sin necesidad de usuario y clave. Para contrarrestar esta situación, la Librería JDBC ofrece un API que realiza validación de datos SQL maliciosos. La clase java: java.sql.PreparedStatement, pero sin embargo sigue siendo vulnerable a la inyección mediante el nombre de usuario. La opción más aceptada seria utilizar la solución compatible PreparedStatement, incluyendo consulta paramétrica con una interrogación (?), para verificar el tamaño del nombre de usuario. [126]

---

[125] MOHINDRA Dhruv (febrero 2018). Prevent SQL injection. {En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection.
[126] MOHINDRA Dhruv (febrero 2018). Prevent SQL injection. {En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection.

```
public void doPrivilegedAction(
  String nombreusuario, char{} clave
) throws SQLException {
  Connection connection = getConnection();
  if (connection == null) {
    // Handle error
  }
  try {
    String passw = hashclave(clave);

    // Valida longitud de  nombreusuario
    if (nombreusuario.length() > 8) {
      // encapsula Error
    }

    String sqlString =
      "select * from db_usuario where nombreusuario=? and clave=?";
    PreparedStatement stmt = connection.prepareStatement(sqlString);
    stmt.setString(1, nombreusuario);
    stmt.setString(2, passw);
    ResultSet rs = stmt.executeQuery();
    if (!rs.next()) {
      throw new SecurityException("Nombre de Usuario o Clave
Incorrecta");
    }

    // Autenticación
  } finally {
    try {
      connection.close();
    } catch (SQLException x) {

    }
  }
}
```

18.1.1.2. Normalización De Cadenas antes de Verificarlas.

La estrategia de muchos sistemas es aceptar las cadenas no fiables, pero implementando validación de cadena de caracteres a través de las listas, la implementación de esta característica la realiza Java basado en Standares Unicode.

Por lo tanto y en conclusión para si el aplicativo decide realizar la validación de cadenas no fiables, debe primero normalizarlas antes de verificarlas.El siguiente ejemplo es compatible para la normalización de las entradas antes de verificarlas.[127]

```
cadena = Normalizer.normalize(cadena, Form.NFKC);
// Validacion
Pattern pattern = Pattern.compile("{<>}");
Matcher matcher = pattern.matcher(cadena);
if (matcher.find()) {
  // Se encontro en Lista Negra
  throw new IllegalStateException();
} else {
  // ...
}
```

Cuando los datos no confiables son detectados se activa la función `IllegalStateException()`.

Es muy riesgoso hacer primero la validación y luego la normalización, porque el atacante puede impedir las validaciones y los filtros de seguridad.[128]

18.1.1.3. Normalizar Nombres de Ruta Antes de Validación.

Los nombres de las rutas se pueden manejar absolutos o relativos e incluso pueden estar vinculados a otros con accesos directos, alias, otros enlaces etc., deben quedar completamente solucionados antes de realizar las verificaciones. Una forma de validar de manera óptima es dar al sistema privilegios a los archivos de la carpeta sobre la cuela se solicita acceso y de solo lectura a través de java.io.FilePermission:[129]

```
// Todos los archivos en  /ejemplo/java se pueden leer
grant codeBase "file:/home/programpath/" {
  permission java.io.FilePermission "/ejemplo/java", "read";
};
```

[127] SEACORD, Robert. (febrero 2018). Characteres and string. {En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487607

[128] SEACORD, Robert. (febrero 2018). Characteres and string. {En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487607.

[129] MOHINDRA Dhruv (febrero 2018). Canonicalize path names before validating them.{En Linea}.Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/FIO16-J.+Canonicalize+path+names+before+validating+them

18.1.1.4. No registrar Entrada sin Analizar

En ningún caso se debe ingresar datos en el sistema sin analizar, porque la probabilidad de que el sistema sea víctima de un ataque por inyección es directamente proporcional a la ausencia de esta validación. Adicionalmente una entrada de datos sin analizar puede genera perdida de información en el caso que el hacker utilice el límite de confianza para a través de inyectar código malicioso tome copia de cookies para acceder al sistema como administrador.[130]

```
if (loginSuccessful) {
  logger.severe("Usuario y Login satisfactorio para : " +
sanitizeUser(username));
} else {
  logger.severe("Usuario y Login Falla para : " +
sanitizeUser(username));
}
```

La desinfección del nombre del usuario se realiza a través del siguiente método.

```
public String sanitizeUser(String username) {
  return Pattern.matches("{A-Za-z0-9_}+", username))
      ? username : "Usuario no Autorizado";
}
```

18.1.1.5. Prevenga Inyección XML

El XML tiene como objetivo principal facilitar el almacenamiento y la transferencia de información; es independiente de la plataforma y tiene una versatilidad amplia, por esta razón es vulnerable para ataques por inyección, porque los datos del XML pueden ser sobrescritos por el usuario mal intencionado. [131]

En el siguiente ejemplo

---

[130] MOHINDRA Dhruv (febrero 2018). Do not log unsanitized user input {En Linea}.Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/IDS03-J.+Do+not+log+unsanitized+user+input

[131]SEACORD, Robert. (febrero 2018). Prevention XML injection.{En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/IDS16-J.+Prevent+XML+Injection.

```
<item>
  <description>Widget</description>
  <precio>10000.0</precio>
  <cantidad>5</cantidad>
</item>
```

El agresor puede enviar la siguiente instrucción para realizar cambios de cantidad por el precio y aprovecharse de esta vulnerabilidad:

```
1</cantidad><precio>1.0</precio><cantidad>
```

Y XML lo resuelve de la siguiente manera:

```
<item>
  <descripcion>Widget</descripcion>
  <pricio>10000.0</precio>
  <cantidad>1</cantidad><precio>1.0</precio><cantidad>1</cantidad>
</item>
```

La forma de solucionar adecuadamente este hueco de seguridad es a través de la siguiente implementación: [132]

```
import java.io.BufferedOutputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;

public class TiendaEnLinea {
  private static void CrearXMLStream(final BufferedOutputStream
outStream,
      final String cantidad) throws IOException, NumberFormatException
{
    // Escribir la cadena XML unicamente si la cantidad es un entero
sin signo (cantidad).
    int contador = Integer.parseUnsignedInt(cantidad);
    String xmlString = "<item>\n<descripcion>Widget</descripcion>\n"
        + "<precio>10000</precio>\n" + "<cantidad>" + contador +
"</cantidad></item>";
    outStream.write(xmlString.getBytes());
    outStream.flush();
  }
```

[132] SEACORD, Robert. (febrero 2018). Prevention XML injection.{En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/IDS16-J.+Prevent+XML+Injection.

```
}
```

18.1.2. Evitar Null en Instancia de Objeto.

No se recomienda la utilización del valor null en instancias donde los solicite un objeto, sobre todo en los siguientes escenarios: Llamada a un método de instancia de un objeto null. Cambios de algún campo de un objeto null. Incluir en el valor null simulando una matriz. Tratar los elementos null igual a un arreglo. La utilización indiscriminada de null en los objetos puede llegar a generar la interrupción del software o stream. En la siguiente implementación se garantiza la invocación isProperName con una referencia de string valido, en el método testString: [133]

```
public class Foo {
  private boolean isProperName(String caracter) {
    String names{} = caracter.split(" ");
    if (names.length != 2) {
      return false;
    }
    return (isCapitalized(names{0}) && isCapitalized(names{1}));
  }

  public boolean testString(String caracter) {
    if (caracter == null) return false;
    else return isProperName(caracter);
  }
}
```

18.1.3. Restricción objeto Object.equals ().

En el lenguaje JAVA los arreglos son objetos y se pueden incluir en el manejo de Objects.equal(), pero no es recomendable hacer comparaciones de matrices con este método, si un analista de software requiere comparar 2 matrices lo mejor es utilizar el método arrays.equals() o se puede utilizar los operadores de igualdad y

[133] SEACORD, Robert. (febrero 2018). Prevention XML injection.{En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/IDS16-J.+Prevent+XML+Injection.

de no igual: == y !=. Ejemplo de implementación no recomendada con sus implementaciones sugeridas:[134]

```
int{} arregloA = new int{20}; // Inicializa en cero
int{} arregloB = new int{20}; // Inicializa en cero
System.out.println(arregloA.equals(arregloB)); // genera Falso.
```

Implementaciones sugeridas:

```
int{} arregloA = new int{20}; // Inicializa en cero
int{} arregloB = new int{20}; // Inicializa en cero
System.out.println(Arrays.equals(arr1, arr2)); // genera Verdadero

int{} arregloA = new int{20}; // Inicializa en cero
int{} arregloB = new int{20}; // Inicializa en cero
System.out.println(arregloA == arregloB); // genera falso
```

18.2. Declaraciones e inicialización DCL.

18.2.1. No Usar Ciclos Inicialización de Clase.

La recomendación es que todos los programas que están más impactados por la seguridad no deberían realizar ciclos de inicialización de Clase.[135]

Ejemplo de código con inicialización de clase:

```
public class Ciclo {
  private final int saldo;
  private static final Cicle M = new Cicle();
  private static final int consignacion = (int) (Math.random() *
1000); // deposito

  public Ciclo() {
    saldo = consignacion - 20; // Resta tarifa
  }
```

[134]MOHINDRA Dhruv (febrero 2018). Do not use the Object.equals() method to compare two arrays.{En línea], Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/EXP02-J.+Do+not+use+the+Object.equals%28%29+method+to+compare+two+arrays

[135] MOHINDRA Dhruv (febrero 2018). Prevent class initialization cycles.{En linea}, Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/DCL00-J.+Prevent+class+initialization+cycles

```
  public static void main(String{} args) {
    System.out.println("La cuenta de saldo es: " + c.saldo);
  }
}
```

Aquí se observa que la variable M se inicializa antes de la inicialización en tiempo de ejecución de la variable Consignación, esto hace que el resultado siempre sea el mismo -20.

El siguiente ejemplo es Implementación con la solución compatible:

```
public class Cycle {
  private final int saldo;
  private static final int consignacion = (int) (Math.random()
* 100); // consignacion aleatoria
  private static final Cycle M = new Cycle();  // Inserta despues de
inicializar del campo requerido
  public Cycle() {
    saldo = consignacion  - 20; //
  }

  public static void main(String{} args) {
    System.out.println("El saldo de la cuenta es: " + c.balance);
  }
}
```

18.2.2. No use Identificadores de biblioteca estándar de java

No es recomendable utilizar o reutilizar las clases públicas de java, librerías y objetos en la biblioteca estándar de Java. Si un desarrollador de software reutiliza un nombre de algún identificador con un nombre semejante a una clase pública que este en la librería estándar de Java, los resultados pueden ser inesperados. Por ejemplo, una implementación que puede tener inconvenientes seria:[136]

```
class Vector {
  private int valor = 1;

  public boolean isEmpty() {
    if (valor == 1) {
```

[136] SVOBODA, David (junio 2015). Do not reuse public identifiers from the Java Standard Librar.{En linea}, Recuperado dehttps://wiki.sei.cmu.edu/confluence/display/java/DCL01-J.+Do+not+reuse+public+identifiers+from+the+Java+Standard+Library

```
      return true;
    } else {
      return false;
    }
  }
  // Other functionality is same as java.util.Vector
}

// import java.util.Vector; omitted
public class VectorUser {
  public static void main(String{} args) {
    Vector Vec = new Vector();
    if (v.isEmpty()) {
      System.out.println("Vector esta vacío");
    }
  }
}
```

La funcionalidad de este código pude ser la misma del java.util.Vector
, sin embargo, el resto de la funcionalidad de la misma utilidad generar una respuesta diferente a la que pretende el programador.

La implementación correcta seria:

```
class MyVector {
  //código o funcionalidad diferente
}
```

Esta implementación es compatible y no riñe con la clase vector definida en la librería de Java.

18.3. EXPRESIONES.

18.3.1. No ignorar las respuestas de los métodos.

Cuando existe un error inesperado o una falla o para el caso de informar el éxito de una transacción o cuando se requiere actualizaciones de diversa índole, los métodos casi siempre devuelven o retornan valores para informar estas incidencias. El no darle atención o descartar este tipo de respuesta puede generar un posible hueco de seguridad. El código fuente que a continuación se muestra da un ejemplo de no validación de respuesta, cuando por ejemplo se intenta eliminar un archivo y por alguna razón no se realiza la operación.[137]

[137]MOHINDRA Dhruv (febrero 2018). Do not ignore values returned by methods

```
public void borrarArchivo(){

  File ejemploArchivo = new File("ejemploArchivo.txt");
  // Aqui va lo que se quiere hacer con el archivo
  ejemploArchivo.delete();

}
```

Un ejemplo de código apropiado para evitar esta vulnerabilidad seria:

```
public void borrarArchivo(){

  File ejemploArchivo = new File("ejemploArchivo.txt");
  // Aqui va lo que se quiere hacer con el archivo
  ejemploArchivo.delete();
  if (!someFile.delete()) {
    //  Fallo el borrado del archivo
      System.out.println("El Archivo no pudo ser borrado");
  }
}
```

18.3.2. No usar nulo en instanciación de Objetos.

Se recomienda no usar el valor null en instancia donde se requiere objetos como, por ejemplo: invocar el método de instancia de un objeto null, actualizando objetos null, manejos de arreglos o matriz con null. Cuando se utiliza null en los casos anteriores se desencadena un NullPointerException el cual detiene la ejecución del software y se recomienda no intervenir o capturar esta excepción. En el siguiente ejemplo se evidencia la incidencia para el caso de la utilización del método isProperName() el cual retorna .T. si el argumento es un nombre valido .
Este método no funciona adecuadamente dado que se puede llamar con un argumento null , el cual genera una inconsistencia de puntero null.[138]

```
public boolean isProperName(String ejemplo) {
  String names{} = ejemplo.split(" ");
  if (names.length != 2) {
    return false;
  }
  return (isCapitalized(names{0}) && isCapitalized(names{1}));
}
```

.{En linea}, Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/EXP00-J.+Do+not+ignore+values+returned+by+methods

[138] MOHINDRA Dhruv (febrero 2018). Prevent SQL injection. {En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection.

La solución a la anterior incidencia seria la siguiente donde se utiliza un objeto en opcional, el cual puede devolver null, pero desde la versión 8 de java y que permite el manejo adecuado las inconsistencias generadas por el puntero null.

```
  public boolean isProperName(Optional<String> os) {
  String names{} = os.orElse("").split(" ");
  return (names.length != 2) ? false :
         (isCapitalized(names{0}) && isCapitalized(names{1}));
}
```

La clase opcional se pueden utilizar para acortar el código y facilitar su lectura y mantenimiento.

18.3.3. Comparación de matrices.

No se recomienda cuando se requiere comparar matrices utilizar el método object.equals(), puesto que solo compararía solo referencias de matriz, no el contenido de los arreglos. Para esta funcionalidad se sugiere a los programadores utilizar el método Array.equals().[139]
Para visualizar este comportamiento, a continuación, se muestra un código con la utilización no recomendada. [140]

```
int{} arreglo1 = new int{50};
int{} arreglo2 = new int{50};
System.out.println(arreglo1.equals(arreglo2));
```

El código que da la solución para esta tipología ejemplo seria el siguiente utilizando Arrays.equals().:

```
int{} arreglo1 = new int{50};
int{} arreglo2 = new int{50};
System.out.println(Arrays.equals(arreglo1, arreglo2));
```

O

```
System.out.println(arreglo1 == arreglo2)
```

18.3.4. Utilización de operadores de igualdad.

[139] MOHINDRA Dhruv (febrero 2018). Do not Use the Object.equals() to compare two array. {En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/EXP02-J.+Do+not+use+the+Object.equals%28%29+method+to+compare+two+arrays
[140] MOHINDRA Dhruv (febrero 2018). Do not Use the Object.equals() to compare two array. {En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/EXP02-J.+Do+not+use+the+Object.equals%28%29+method+to+compare+two+arrays

Los valores de las primitivas de caja no se pueden comparar directamente utilizando igual == o no igual!=, puesto la comparación se hace a nivel de referencia de objeto y no de valores de objeto. El siguiente código ejemplo utiliza el método compare (), para 2 primitivas de caja con argumentos, lo que se pretendía era la comparación del contenido de los objetos y no de sus referencias: [141]

```
public class Wrapper {
  public static void main(String{} args) {
    Integer x1 = 100;
    Integer x2 = 100;
    Integer x3 = 1000;
    Integer x4 = 1000;
    System.out.println(x1 == x2);
    System.out.println(x1 != x2);
    System.out.println(x3 == x4);
    System.out.println(x3 != x4);
  }
}
```

La mejor manera para comparar el contenido de los objetos de las primitivas es:

```
public class Wrapper {
  public static void main(String{} args) {
    Integer x1 = 100;
    Integer x2 = 100;
    Integer x3 = 1000;
    Integer x4 = 1000;
    System.out.println(x1.equals(x2));
    System.out.println(!x1.equals(x2));
    System.out.println(x3.equals(x4));
    System.out.println(!x3.equals(x4));
  }
}
```

Otro ejemplo podría ser con los constructores de la clase valores booleanos, este tipo de constructores devuelven variados objetos que estén instanciados, cuando el programador utiliza los operadores de igualdad de la referencia en vez de comparación de valores, puede generar resultados no esperado.[142]

```
public void EjemploOperadorIgualdad(){
  Boolean varbooleano1 = new Boolean("true");
  Boolean varbooleano2 = new Boolean("true");
```

[141] MOHINDRA Dhruv (febrero 2018). Do not Use the equality operators when comparing values of boxes primitives. {En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/EXP03-J.+Do+not+use+the+equality+operators+when+comparing+values+of+boxed+primitives

[142] MOHINDRA Dhruv (febrero 2018). Do not Use the equality operators when comparing values of boxes primitives. {En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/EXP03-J.+Do+not+use+the+equality+operators+when+comparing+values+of+boxed+primitives

```
  if (varbooleano1 == varbooleano2) {
    System.out.println("Esta comparación nunca es igual");
  }
}
```

El código adecuado para este manejo seria:

```
public void EjemploOperadorIgualdad (){
  Boolean varbooleano1 = true;
  Boolean varbooleano2 = true;

  if (varbooleano1 == varbooleano2) {   // siempre Seria igual
    System.out.println("Siempre es igual");
  }

  varbooleano1 = Boolean.TRUE;
  if (varbooleano1 == varbooleano2) {   // siempre es igual
    System.out.println("Siempre es igual");
  }
}
```

Los valores de caja True o False se pueden compararse utilizando los operadores de igualdad y no arrojara resultados inesperados.

18.4. Tipos y operaciones numéricas.

18.4.1. Prevenir Desbordamiento de Enteros.

El software como regla general no debería permitir que las operaciones matemáticas se extralimiten los valores que definen los límites de sus tipos de dato.

Los principales operadores de comparación que ofrecen como respuesta unos valores de tipo lógico son: Los operadores de comparación numérica <, <=, >, y >=.
Los operadores de igualdad numérica ==y!=.
Los que dan como resultado un tipo de dato Long son:
+ y -, *, /y %, &, ^y, *, /y %, *, /y %
En el siguiente ejemplo podemos evidenciar este comportamiento de desbordamiento y se puede presentar un resultado inesperado y producir un error.[143]

[143] MERTIKAS, Efstagios. (febrero 2018). Detected or prevent integer overflow. {En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/NUM00-J.+Detect+or+prevent+integer+overflow.

```
public static int MultipleAcumulado(int CuantaAnt, int NuevoValor, int es
cala) {
  // Se puede generar desbordamiento.
  return CuantaAnt + (NuevoValor * escala);
}
```

La corrección del código anterior seria:

```
public static int MultipleAcumulado
(int CuantaAnt, int NuevoValor, int escale) {
  return safeAdd(oldAcc, safeMultiply(newVal, scale
}
```

Esta opción utiliza los métodos para controlar desbordamientos; estos métodos son: SafeAdd(), SafeMultiply.

18.4.2. Evitar el uso de Operación Bit a Bit.

No se recomienda mezclar la operación que se realizan bit a bit con las operaciones aritméticas, porque los resultados pueden ser impredecibles. Las colecciones de bit son las indicadas para el manejo bit a bit (algunos operadores de bit son el unario ~ y los operadores binarios <<, >>, &, ^ y |).

Es muy común que se use indiscriminadamente esta combinación que genere problemas en las variables que almacenan el resultado y lo más delicado es que pueden compilar y "funcionar". Un ejemplo de este comportamiento no recomendado seria combinando operadores de desplazamiento con operaciones aritméticas [144]

```
int z = 70;
z += (z << 2) + 1;
```

La forma de solucionar el desplazamiento esta situación seria incluir la división para realizar el desplazamiento.

```
int z = -50;
z /= 4;
```

18.4.3. División por Cero.

[144] SVOBODA, David. (febrero 2016).do not perform bitwise and aritmetic operations.{En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/NUM01-J.+Do+not+perform+bitwise+and+arithmetic+operations+on+the+same+data.

La división por cero se puede presentar en operaciones de división y resta, en consecuencia, antes de realizar la operación se debe verificar el cero.
El siguiente es un ejemplo del comportamiento de esta tipología.
Esta operación puede generar el error división por cero porque no se tiene el control previo del divisor de la operación.[145]

```
long numeroA, numeroB, ResultadoSuma;

/* Se inicializa numeroA y numeroB  */

ResultadoSuma = numeroA / numeroB;
```

La forma de evitar esta vulnerabilidad es como se indica en el siguiente código:

```
long numeroA, numeroB, resultadoSuma;

 if (numeroA == 0) {
  // Se hace algo si hay error de divisor en cero
} else {
  resultadoSuma = numeroA / numeroB;
}
```

18.4.4. Tipos de datos sin firma.

En los lenguajes de programación C++ o C, se utiliza muy a menudo tipos de dato que no tienen firma, como por ejemplo los tipos de datos char de 16 bits. El manejo de datos como si estuvieran firmados, puede generar valores errados y llegar a genera perdida de información y posibles huecos de seguridad. Este manejo no se puede automatizar, se debe programar. Como se debe soportar r este manejo lo más aconsejable para el manejo en Java, es almacenarlos en un tipo de dato entero por ejemplo Long, el cual puede soportar todo el rango de valores. Por ejemplo El método readInt() asume valores firmados y retorna un entero firmado, luego se convierte en un tipo de dato Long con signo, por consiguiente se de incluir una máscara considerando el signo.[146]

---

[145] PAULSON, Jonathan. (febrero 2018).Ensure tan división and remainder operation do not result in divide-by-zero errors.{En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/NUM02-J.+Ensure+that+division+and+remainder+operations+do+not+result+in+divide-by-zero+errors

[146] MOHINDRA, Druv . (Junio 2015).Ensure tan división and remainder operation do not result in divide-by-zero errors.{En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/NUM03-J.+Use+integer+types+that+can+fully+represent+the+possible+range+of++unsigned+data

No soporta esta recomendación:

```
return is.readInt();
```

Código que soporta la recomendación:

```
return is.readInt() & 0xFFFFFFFFL;
```

18.4.5. Tipos de Datos coma flotante.

Para cuando se requiere cálculos con mucha precisión no se recomienda utilizar tipos de datos de coma flotante, porque puede generar resultados no esperados imprecisos, perdida de información. Si se llegasen a utilizar tipo de datos de punto flotante para un cálculo preciso, se debe tener en cuenta la tolerancia del error que se pueden permitir para el caso en particular. Código ejemplo que no cumple con la recomendación:[147]

```
double pesos = 1.00;
double centavos = 0.10;
int Cantidad = 6;
System.out.println(
  "Un Peso menos " + Cantidad + " centavos es  $" + (pesos - cantidad *
centavos)
);
```

El siguiente código soluciona si contiene la recomendación, utilizando el tipo BigDecimal la cual es más precisa en cuanto a los decimales.

```
import java.math.BigDecimal;

BigDecimal pesos = new BigDecimal("1.0");
BigDecimal centavos = new BigDecimal("0.1");
int cantidad = 6;
System.out.println ("Un peso menos  " + cantidad + " centavos es  $" +
    (pesos.subtract(new BigDecimal(cantidad).multiply(centavos) )) );
```

18.4.6. Evitar comparaciones con NaN.

[147] LONG, Fred. (febrero 2018).Do not use floating point numbers if precise computation is requerid.{En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/NUM04-J.+Do+not+use+floating-point+numbers+if+precise+computation+is+required.

No se recomienda hacer uso para las comparaciones directas para tipos de dato de punto flotante. (NaN significa not a number), en los casos de verificaciones de entradas si se utiliza NaN puede llegar a generar resultados no esperados.
Ejemplo de código no compatible donde se hace la comparación inmediata con NaN, la cual normalmente arroja resultado falso. [148]

```
Class public NaNComparison {
   public static void main (String {} args) {
     double y = 0.0;
     double resultado = Math.cos (1 / x); //Devuelve NaN si la entrada es
infinita
     if (resultado == Double.NaN) {// La comparación siempre es falsa!
       System.out.println ("el resultado es NaN  Esto nunca se imprime");
     }
   }
 }
```

La solución para este caso sería el siguiente código, el cual utiliza el método double para verificar si es un valor NaN.

```
clase pública NaNComparison {
   public static void main (String {} args) {
     double x = 0.0;
     double resultado = Math.cos (1 / x);  // Devuelve NaN cuando la
entrada es infinita
     if (Double.isNaN (resultado)) {
       System.out.println ("el resultado es NaN");
     }
   }
 }
```

En general el riesgo de no controlar este tipo de comparaciones, es que el resultado puede ser inesperado e impredecible.

18.4.7. Variables de punto flotante en ciclos.

A los programadores se les debe hacer claridad sobre que no es recomendable utilizar las variables de punto flotante no se deben utilizar como contadores en un Bucle. Los Tipos de punto flotante tienen las siguientes restricciones: Fracciones simples, decimales, dígitos con valores Grandes. Ejemplo de código con manejo

[148] MOHINDRA, Dhruv. (febrero 2018).Do not use floating as loop counter {En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/NUM07-J.+Do+not+attempt+comparisons+with+NaN.

inadecuado, donde se observa que se utiliza un contador de punto flotante el cual va incrementando muy poco en cada iteración generando resultados inesperados.[149]

```
for (float z = 0.2f; x <= 2.0f; z + = 0.2f) {
   System.out.println (z);
 }
```

Este código corrige las falencias del anterior:

```
for (int contador = 1; contador <= 10; contador + = 1) {
   double z = 200000000.0 + contador;
   / * ... * /
 }
```

Como se puede observar se utiliza un contador del ciclo de tipo entero y luego hace su transformación a float, garantiza con la definición del doublé su precisión.

18.4.8. Conversiones de tipo numérico.

La conversión de datos de tipo numérico a uno de menos capacidad puede genera perdida de datos, errores de información, interpretaciones inadecuadas y por ende un hueco de seguridad. Existen conversiones como por ejemplo de short a byte o char, de int a byte, de short a char, de float a byte entre otras. Este tipo de conversiones primitivas se permite siempre y cuando el rango de la de mayor capacidad este incluida en la de menor capacidad. En el siguiente ejemplo se evidencia una conversión de tipo entero a byte sin realizar la validación del rango.[150]

```
class Descartar {
  public static void main(String{} args) {
    int j = 256;
    EjemploConv(j);
```

[149] LONG, Fred. (febrero 2018).Do not use floating point numbers if precise computation is requerid.{En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/NUM09-J.+Do+not+use+floating-point+variables+as+loop+counters.

[150] SVOBODA, David. (febrero 2018).ensure convertions of numeric type to narrower types do not result in lost or misinterpreter data.{En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/NUM12-J.+Ensure+conversions+of+numeric+types+to+narrower+types+do+not+result+in+lost+or+misinterpreted+data.

```
  }
  public static void EjemploConv (int j) {
    byte c = (byte) j;  // c define valor  -256
  }
}
```

La forma adecuada para este manejo se da en el siguiente código, donde se evalúa antes de realizar la conversión si el rango es el adecuado para la conversión.

```
class Descartar{
  public static void EjemploConv (int j) {
    // Se comprueba si se esta dentro del rango de bytes
    if ((j < Byte.MIN_VALUE) || (j > Byte.MAX_VALUE)) {
      throw new ArithmeticException("Valor esta fuera del rango");
    }
    byte c = (byte) j;
    // Work with b
  }
}
```

18.5. Caracteres y Cadenas

18.5.1. Cadena de Ancho Variables.

Se recomienda a los programadores no conformar strings que incluyan caracteres fragmentados, es decir hacer conversión de caracteres de ancho variables a cadenas. La conformación de cadenas teniendo como base datos con caracteres parciales puede generar errores y pérdidas de información e inconsistencia de datos. En el siguiente Ejemplo no se considera el bucle de codificación de longitud variable y esto puede generar problemas en los bytes de sus extremos, lo que puedes desencadenar en malas interpretaciones de los datos. [151]

```
public EjemploCadenas int MAX_TAMANO = 2048;

public String readBytes(Socket socket) throws IOException {
  InputStream in = socket.getInputStream();
```

[151] SEACORD, Robert. (febrero 2018).Dont form string containing partial carácter from variable – width encoding .{En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/STR00-J.+Don%27t+form+strings+containing+partial+characters+from+variable-width+encodings.

```
  byte{} data = new byte{MAX_TAMANO +1};
  int offset = 0;
  int bytesRead = 0;
  String str = new String();
  while ((bytesRead = in.read(data, offset, data.length - offset)) !=
-1) {
    str += new String(data, offset, bytesRead, "UTF-8");
    offset += bytesRead;
    if (offset >= data.length) {
      throw new IOException("Hay muchas entradas");
    }
  }
  in.close();
  return str;
}
```

La forma adecuada para el manejo de cadenas de ancho variable seria como se presenta a continuación donde se hace en la lectura una conversión de bytes a char obviando los 2048 bytes. El procedimiento genera excepción en caso que el socket entregue más de 2048 caracteres en vez de 2048 bytes.

```
public EjemploCadenas int MAX_SIZE = 2048;

public String readBytes(Socket socket) throws IOException {
  InputStream in = socket.getInputStream();
  Reader r = new InputStreamReader(in, "UTF-8");
  char{} data = new char{MAX_TAMANO +1};
  int offset = 0;
  int charsRead = 0;
  String str = new String();
  while ((charsRead = r.read(data, offset, data.length - offset)) != -
1) {
    str += new String(data, offset, charsRead);
    offset += charsRead;
    if (offset >= data.length) {
      throw new IOException("Existen muchas entradas..");
    }
  }
  in.close();
  return str;
}
```

18.5.2. Manejo Configuración Regional.

No se recomienda a los programadores hacer dependientes de la configuración regional el manejo de los métodos, pues en caso que el equipo donde se ejecuta el programa no tiene definida una configuración regional, puede genera resultado inesperado y muy riesgosos para el usuario o compañía. Se acostumbra tener una configuración regional específica para manejar por ejemplo la moneda, claves, protocolos HTML y otras configuraciones, normalmente locate.ENGLISH. Si un programa se ejecuta con una configuración arbitraria genera un comportamiento no controlado del software, abriendo la posibilidad a un atacante que evite los filtros y controles que se incluyan a la entrada del sistema. Por ejemplo, en el siguiente código funciona para cuando es una configuración regional en inglés, pero en otras configuraciones que tienen el Latino dentro de su configuración, la "I" mayúscula se comporta como una I mayúscula pero conservando el punto de la i minúscula. [152]

```
public class EejmploCadenas {
  public static void main(String{} args) {
    System.out.println("Indice".toUpperCase());
  }
}
```

Este comportamiento tiene su mayor impacto en sitios Blogs Html los cuales utilizan scripts intermedios e inclusive posibles ataques con injection . Lo que se recomienda es primero realizar el filtro de todas estas vulnerabilidades antes de enviarlo a la web. En HTML las etiquetas manejan indiferentemente las mayúsculas y la minúsculas en el siguiente código se utiliza el método tag.toUppercase, el cual depende de la configuración regional para el manejo de las etiquetas, y se excluye las etiqueta marcadas con <SCRITP>, para este caso en la configuración de ingles quedaría "script" y para Europa Oriental quedaría "SCRİPT" y no funcionaría en ninguno de estos casos la exclusión de la etiqueta <SCRIPT>[153]

---

[152] SEACORD, Robert. (febrero 2018). Specify an appropriate locale when comparing locale-dependent data..{En línea}. Recuperado de:https://wiki.sei.cmu.edu/confluence/display/java/STR00-J.+Don%27t+form+strings+containing+partial+characters+from+variable-width+encodings

[153] SEACORD, Robert. (febrero 2018). Specify an appropriate locale when comparing locale-dependent data..{En línea}. Recuperado de:https://wiki.sei.cmu.edu/confluence/display/java/STR00-J.+Don%27t+form+strings+containing+partial+characters+from+variable-width+encodings

```
public static void processTag(String tag) {
  if (tag.toUpperCase().equals("SCRIPT")) {
    return;
  }
  // Aquí va el procesamiento de los tags
}
```

La forma de solventar esta situación se muestra en el siguiente código de ejemplo la cual incluye de manera predeterminada la configuración regional de Ingles, previamente antes de realizar la operación entre cadenas.

```
public static void processTag(String tag) {
  Locale.setDefault(Locale.ENGLISH);
  if (tag.toUpperCase().equals("SCRIPT")) {
    return;
  }
  // Aquí se procesan los tags
}
```

18.6. ORIENTACIÓN DEL OBJETO.

18.6.1. Accesibilidad a Campos.

Las propiedades Invariantes en lo posible la recomendación es que no se apliquen a campos públicos o a objetos mutables. Este comportamiento puede ser utilizado por los hackers para a través de la utilización de estos campos para realizar sus ataques. En lo posible y la recomendación final sería que estos campos sean declarados privados. El siguiente es un ejemplo de código muestra la vulnerabilidad, donde se tiene una variable entera donde se guardan la cantidad de campos adicionados o borrados a través de add o remove. TotaldeCampos es una variable publica puede ser alterado por el usuario o por un atacante, poniendo en riesgo la seguridad del sistema. [154]

```
public class Artefacto {
  public int TotaldeCampos; // Total de campos
```

[154] LONG, Fred. (noviembre 2017). Limited accecibility of field. {En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/OBJ01-J.+Limit+accessibility+of+fields.

```
  void add() {
    if (TotaldeCampos < Integer.MAX_VALUE) {
      TotaldeCampos ++;
      // cuerpo del código por verdadero
    } else {
      throw new ArithmeticException("Overflow");
    }
  }

  void remove() {
    if (TotaldeCampos > 0) {
      TotaldeCampos --;
      // cuerpos del código por verdadero
    } else {
      throw new ArithmeticException("Overflow");
    }
  }
}
```

La manera correcta para evitar esta vulnerabilidad seria declarando la variable TotaldeCampos como privada:

```
public class Artefactos {
  private int TotaldeCampos; // Se declara privada las variables.

  public int ObtenerTotalCampos () {
    return TotaldeCampos;
  }

  // Defunción y manejo para add y remove
}
```

18.6.2. Dependencias de Subclases.

Los ingenieros de desarrollo de software tienen la concepción que dividir la funcionalidad de los programas en múltiples clases o módulos para que el código sea reutilizable. Esto es cierto, pero se debe tener especial cuidado cuando se realiza mantenimiento o cambios al software, por ejemplo, cuando se realizan

cambios en una superclase se debe garantizar que se permanezcan las propiedades invariantes que dependen de las subclases. Si esto no es considerado por los desarrolladores, se abre una vulnerabilidad a la seguridad del sistema. En el siguiente ejemplo se muestra como es el comportamiento de esta vulnerabilidad: por ejemplo un programador puede desarrollar una subclase SubclaseCalendario que invoque la superclase calendar.after y asumir que este método de la superclase Calendar solo invocaría sus métodos sin los de anulación de la subclase.[155]

```
class SubclaseCalendario extends Calendar {
  @Override public boolean after(Object when) {
    if (when instanceof Calendar &&
        super.compareTo((Calendar) when) == 0) {
      return true;
    }
    return super.after(when);
  }

  @Override public int compareTo(Calendar otroCalendario) {
    return compareDays(this.getFirstDayOfWeek(),
                       otroCalendario.getFirstDayOfWeek());
  }

  private int compareDays(int currentFirstDayOfWeek,
                          int anotherFirstDayOfWeek) {
    return (currentFirstDayOfWeek > anotherFirstDayOfWeek) ? 1
           : (currentFirstDayOfWeek == anotherFirstDayOfWeek) ? 0 : -
1;
  }

  public static void main(String{} args) {
    SubclaseCalendario  cal1= new SubclaseCalendario ();
    cal1.setTime(new Date());
    cal1.set(Calendar.DAY_OF_WEEK, Calendar.SUNDAY);
    SubclaseCalendario  cal2 = new SubclaseCalendario ();
    System.out.println(cal1.after(cal2));
  }
```

[155] MOHINDRA, Dhruv. (noviembre 2016). Preserve dependencies in subclases when changing superclases. {En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/OBJ02-J.+Preserve+dependencies+in+subclasses+when+changing+superclasses.

```
 }
```

El código que es compatible y soporta esta vulnerabilidad, el cual utiliza el patrón de diseño denominado composición y reenvío, la cual incluye una nueva clase de reenvío que contiene campos privados, en otras palabras, seria composición en vez de la herencia. En este ejemplo el campo privado está en la clase ImplementaCalendario, instanciación de la superclase Calendario. También se incluye una clase ComponeCalendario que contiene los mismos métodos de la subclase calendario del anterior código de ejemplo tomado de Drup Mohindra: [156]

```
public class AdelantaCalendario {
  private final ImplementaCalendario x;

  public AdelantaCalendario(ImplementaCalendario x) {
    this.x = x;
  }

  ImplementaCalendario getCalendarImplementation() {
    return x;
  }

  public boolean after(Object when) {
    return x.after(when);
  }

  public int compareTo(Calendar otroCalendario) {
    // ImplementaCalendario.compareTo() will be called
    return x.compareTo(otroCalendario);
  }
}

class CalendarioCompuesto extends AdelantaCalendario {
  public CalendarioCompuesto(ImplementaCalendario ci) {
    super(ci);
  }

  @Override public boolean after(Object when) {
```

[156] MOHINDRA, Dhruv. (noviembre 2016). Preserve dependencies in subclases when changing superclases. {En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/OBJ02-J.+Preserve+dependencies+in+subclasses+when+changing+superclasses.

```
    if (when instanceof Calendar &&
        super.compareTo((Calendar)when) == 0) {
      // Devuelve verdadero si es el primer día de la semana
      return true;
    }
    return super.after(when);
  }

  @Override public int compareTo(Calendar otroCalendario) {
    return ComparaDias(
              super.getCalendarImplementation().getFirstDayOfWeek(),
              otroCalendario.getFirstDayOfWeek());
  }

  private int ComparaDias(int currentFirstDayOfWeek,
                          int anotherFirstDayOfWeek) {
    return (currentFirstDayOfWeek > anotherFirstDayOfWeek) ? 1
           : (currentFirstDayOfWeek == anotherFirstDayOfWeek) ? 0 : -
1;
  }

  public static void main(String{} args) {
    ImplementaCalendario cal1 = new ImplementaCalendario();
    cal1.setTime(new Date());
    // Fecha del último domingo (antes de ahora)
    cal1.set(Calendar.DAY_OF_WEEK, Calendar.SUNDAY);

    ImplementaCalendario cal2 = new ImplementaCalendario();
    CalendarioCompuesto x = new CalendarioCompuesto(cal1);
    // Se esperaría que imprima  por verdadero
    System.out.println(x.after(cal2));
  }
}
```

18.6.3. Devolución de Referencias.

Esta vulnerabilidad hace referencia a cuando se retorna referencias a elementos mutables de una clase, la cual puede comprometer seriamente la seguridad del software, porque un ataque malicioso puede quebrar el encapsulamiento. En conclusión, la recomendación que se hace a los programadores es que nunca

deben retornar referencias a clases mutables privadas. En el siguiente ejemplo se observa el método getdate() el cual retorna una única instancia privada del objeto date. Un atacante malicioso puede manipular el objeto privado date, dado que al devolver la referencia pone en riesgo la información.[157]

```
class MutableClass {
  private Date fecha;

  public MutableClass() {
    fecha = new Date();
  }

  public Date getDate() {
    return fecha;
  }
}
```

El código siguiente es el que corrige esta anomalía utilizando clone (), y esto se da porque cuando se devuelve la referencia a una copia, los atacantes no pueden modificar el estado. Para este ejemplo en particular devuelve un clon del objeto date desde el método getdate(), y no puede ser intervenido por un hacker porque está blindado por la subclase MutableClass.

```
public Date getDate() {
  return (Date)d.clone();
}
```

18.6.4. Salida y Entrada.

18.6.4.1. Información en el lado del cliente.

Frecuentemente cuando se desarrollan aplicaciones cliente-servidor, los ingenieros desarrolladores de software, dejan almacenada la información que se denomina

[157] [157] MOHINDRA, Dhruv. (febrero 2018). Do not return references to private mutable class member. {En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/OBJ05-J.+Do+not+return+references+to+private+mutable+class+members.

confidencial, incluyendo claves de usuario al lado del cliente. Esto puede generar un riesgo de seguridad en caso de que el cliente sea vulnerable a ataques.

Este riesgo se resuelve en el software web dejando cokies al lado del cliente y almacenando toda la información sensible en el servidor. La cokie es el método de comunicación entre el cliente y el servidor cada vez que el usuario se logea y esto hace que se proporcione una nueva cada vez que acceda al sistema disminuyendo el riesgo de vulnerabilidad.

La recomendación para esta vulnerabilidad es utilizar el paquete de java HttpServletRequest(), implementado funcionalidad de recordación de guardado de nombre de sesión y usuario y un string para la cokie, apoyado en httpsession para mantener el estado de sesión.[158]

19. CONCLUSIONES.

Al concluir esta monografía se puede afirmar que el ataque a los sistemas informáticos siempre apunta al software y sistemas de información y de no poseer un blindaje a nivel de software, es decir no tiene incluido seguridad dentro del código fuente del mismo, se verá impactado directamente a la organización poniendo en riesgo la información confidencial de los clientes de la compañía.

La codificación segura del software es fundamental para minimizar el riesgo de ocurrencia de Huecos de Seguridad en los sistemas.

Es fundamental que los profesionales dedicados al desarrollo de software, adopten metodologías de software seguro.

Con la implementación de Seguridad al momento del desarrollo del software, se obtendrá productos más estables y blindados contra ataques maliciosos.

[158]HALL, Ryan. (Noviembre 2017). Do not store unencrypted sensitive information on the client side.{En línea}. Recuperado de: https://wiki.sei.cmu.edu/confluence/display/java/FIO52-J.+Do+not+store+unencrypted+sensitive+information+on+the+client+side

Los ataques a los programas y sistemas son permanentes y más sofisticados, por lo tanto, se debe estar actualizando a los ingenieros de desarrollo en técnicas y metodologías para contrarrestar ese riesgo.

Otra conclusión es que es muy importante que la compañías que se dedican al desarrollo de software o fábricas de software, incorporen cualquier metodología para el desarrollo de software seguro, algunas tienen ventajas sobre otras y básicamente depende del uso y el entorno en que sean utilizadas.

A continuación, se presenta la Figura 20 la cual corresponde a un cuadro comparativo entre las metodologías para desarrollo de software seguro tratadas en este documento y se incluyen 2 metodologías que no son del alcance de esta monografía: SAMM[159] , BSIMM[160]. Podemos concluir que

Figura 20. ***Cuadro comparativo Metodologías para Desarrollo Software Seguro***

DOMINIO/METODOLOGIA	Microsoft SDL	CbyC	OWASP	SAMM	BSIMM
Entorno de Aplicación	Empresas, entornos personales y en internet	empresarial y personal	Entornos de internet	organizaciones pequeñas, medianas y grandes	Empresas
Abierto	NO	NO	SI	SI	NO
Documentación	Bastante documentación a nivel de proceso	NO	NO	SI	SI
Métodos Formalizados	NO	SI	SI	SI	SI
Desarrollo por iteraciones	NO	SI	NO	NO	SI
Tracking de requerimientos	NO	SI	SI	SI	SI
Revisa código	NO	SI	SI	SI	SI
Tiene Plan para mitigar ataques	SI	NO	SI	NO	SI
Políticas testing	SI	NO	NO	SI	SI
Modelo de Gobierno	NO	NO	NO	SI	SI

Fuente: http://recibe.cucei.udg.mx/revista/es/vol2-no3/pdf/computacion05.pdf

[159] SAMM. (2018) The Software Assurance Maturity Model. {En línea}. Recuperado de: http://www.opensamm.org.
[160] BSIMM. (2018) The Building Security In Maturity Model. {En línea}. Recuperado de: https://www.bsimm.com.

.

19.1. RECOMENDACIÓNES Y DIFUSION.

Hay que tener muy presente que esta guía abarca los temas más generales y comunes a nivel de seguridad en desarrollo de software, pero se recomienda a los desarrolladores estar actualizándose y consultando con mucha frecuencia ya sea en internet o referencias bibliográficas sobre las nuevas amenazas o vulnerabilidades dada su continua y rápida aparición y mutación.

Complementar la consulta de esta guía con información de temas de seguridad para desarrollo de software para de SmartPhones (SO Android y IOS), dado el auge y popularidad que en la actualidad tienen.

La difusión que se le debe dar a esta guía puede ser desde los repositorios que la Universidad posee para la consulta de los diferentes usuarios.

20. BIBLIOGRAFIA

BRITO, Carlos Joaquín. Metodología para el software Seguro. {En Línea}. Diciembre 2013. {15 noviembre 2017} recuperado de: (http://recibe.cucei.udg.mx/revista/es/vol2-no3/pdf/computacion05.pdf)

CROXFORD, Martin. Correctness by Construccion A Manifiesto for High-Integrity software. {En Linea}. Diciembre 2015. {25 noviembre 2017} recuperado de: http://static1.1.sqspcdn.com/static/f/702523/9277880/1288929538453/200512-Croxford.pdf?token=JaJHj9j4llTZgJa0IeFMf4Jal4Q%3D.

MCMANUS, José. SEI CERT Oracle Coding Standard for Java.{En Linea}. Noviembre 2017. {10 enero 2018} recuperado de: https://www.securecoding.cert.org/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java

MICROSOFT CORPORATION. Implementación simplificada del proceso SDL de Microsoft.{En Linea}.Febrero de 2010. {20 noviembre 2017} recuperado de: https://www.microsoft.com/es-ES/download/confirmation.aspx?id=12379

ASTEAUSUAIN, Fernando. Programación Orientada a Aspectos, análisis del paradigma. Buenos Aires,2002. 130p. Tesis de Licenciatura. Universidad Nacional del Sur. Recuperado de http://www.angelfire.com/ri2/aspectos/Tesis/tesis.pdf

RAMÍREZ AGUILERA, Jonathan.Implicaciones de seguridad de metodologías agiles de desarrollo de software. {En Linea}.2017. {25 Octubre 2017} recuperado de:http://repository.libertadores.edu.co/bitstream/handle/11371/1163/ramirezjonathan2017.pdf?sequence=2

HOWARD Michael y LEBLANC David. 19 puntos críticos sobre seguridad de software. McGraw-Hill Interamericana, enero 2007. 305 p.

OWASP. The open Web Aplication Security Project, Los 10 riesgos más críticos en aplicaciones Web.{En línea}.Agosto 2017. {Marzo 2018} recuperado de: http://www.owasp.org/index.php/Top_10

AMEY, Peter. CBYC Correctness by construction. {En Linea}.14 de mayo de 2013. {25 Octubre 2017} recuperado de: https://www.us-cert.gov/bsi/articles/knowledge/sdlc-process/correctness-by-construction

CIBERSEGURIDAD.net. Falsificación de sitios cruzados XSS. {En línea}.22 de agosto 2017. {Febrero 2018} recuperado de: https://cyberseguridad.net/index.php/453-falsificacion-de-peticiones-en-sitios-cruzados-cross-site-request-forgery-csrf-ataques-informaticos-v

HOWARD,Michael y LIBNER, Steve.The trustworthy Computing Security Development lifecycle.{En Linea}.Marzo 2005. {15 de abril de 2018}.Recuperado de: https://msdn.microsoft.com/en-us/library/ms995349.aspx.

ICS-CERT. Annual Vulnerability Coordination Report Industrial Control Systems Cyber Emergency Response Team. {En Línea}. 2016.{5 de Mayo de 2018} recuperado de: https://ics-cert.us-cert.gov/sites/default/files/Annual_Reports/NCCIC_ICS-CERT_FY%202016_Annual_Vulnerability_Coordination_Report.pdf

CORNEGIE MELLON UNIVERSITY. Software Engineering Institute (2018). {En Línea}. 2018.{25 de mayo de 2018}. Recuperado de: https://www.sei.cmu.edu/about/divisions/cert/index.cfm