



ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

“PROPUESTA DE TÉCNICAS DE ASEGURAMIENTO DE APLICACIONES WEB DESARROLLADAS EN JAVA”

PAÚL XAVIER PAGUAY SOXO

**Tesis presentada ante la Escuela de Postgrado y Educación Continua de
la ESPOCH, como requisito parcial para la obtención del grado de
Magister en Interconectividad de Redes.**

RIOBAMBA – ECUADOR

2013

CERTIFICACIÓN

El Tribunal de Tesis certifica que:

El trabajo de investigación titulado: “**PROPUESTA DE TÉCNICAS DE ASEGURAMIENTO DE APLICACIONES WEB DESARROLLADAS EN JAVA**”, de responsabilidad del Ingeniero Paúl Xavier Paguay Soxo ha sido prolijamente revisado y se autoriza su presentación.

Tribunal de Tesis:

Dr. Ms.C. Juan Vargas
PRESIDENTE

Ing. Ms.C. Danilo Pastor
Ramírez
DIRECTOR

Ing. Ms.C. Gloria Arcos
Medina
MIEMBRO

Ing. Ms.C. Daniel Haro
Mendoza
MIEMBRO

**Escuela Superior Politécnica de Chimborazo
Riobamba, Marzo del 2013**

DERECHOS INTELECTUALES

Yo, Paúl Xavier Paguay Soxo, soy responsable de las ideas, doctrinas y resultados expuestos en esta Tesis y que el patrimonio intelectual generado por la misma pertenece exclusivamente a la Escuela Superior Politécnica de Chimborazo.

FIRMA
CÉDULA: 060272477-5

AGRADECIMIENTO

A través de la presente quiero expresar mi sincero agradecimiento a todas aquellas personas que han colaborado en la consecución de esta meta.

En primer lugar a mis padres, hermanas, familiares, amigos incondicionales y a los profesores del programa que comparten sus conocimientos y experiencias en especial a los Másters Danilo Pástor, Gloria Arcos, Daniel Haro y Gonzalo Allauca por sus acertados comentarios y sugerencias.

DEDICATORIA

Este trabajo de postgrado lo dedico a todos mis seres queridos en especial a Dorian, Carlos, Blanca, Jimena, Karla, que son mi soporte para continuar día a día.

INDICE DE ABREVIATURAS

A

ACL Access Control List (Lista de control de accesos)

B

BD Base de Datos

C

CID Confidencialidad, Integridad, Disponibilidad

D

DOS Denief of Service (Deniego de Servicio)

DBMS Sistema de Gestión de Base de Datos

DER Diagrama Entidad Relación

E

EPEC Escuela de Postgrado y Educación Continua

ESAPI Enterprise Security API

ESPOCH Escuela Superior Politécnica de Chimborazo

I

IP Internet Protocol

H

HTML HyperText Markup Language («lenguaje de marcado de hipertexto»)

J

JSP Java Server Page (Servidor de Páginas JAVA)

M

MAC Medium Acces Control (Controlador de Acceso al Medio)

O

OSI Open SystemInterconnection (Interconexión de Sistema Abierto)

OWASP: Open Web Application Security Project

S

SISEPEC Sistema de la EPEC

SQL StructuredQueryLanguage (Lenguaje estructurado de Consultas)

SI Sistema de Información

T

TIC Tecnologías de la información y Comunicación

U

UML Unified Modeling Language

V

Variable Es una característica que al ser medida es susceptible de adoptar diferentes valores en el transcurso del tiempo.

X

XSS Cross Site Scripting

INDICE GENERAL

CERTIFICACIÓN	II
DERECHOS INTELECTUALES.....	III
AGRADECIMIENTO	IV
DEDICATORIA.....	V
INDICE DE ABREVIATURAS	VI
INDICE GENERAL.....	VIII
ÍNDICE DE FIGURAS	X
ÍNDICE DE TABLAS	XII
CAPÍTULO 1	1
INTRODUCCIÓN	1
1.1 Justificación	6
1.2 Objetivos.....	8
1.2.1 Objetivo General	8
1.2.2 Objetivos Específicos.....	8
REVISIÓN DE LITERATURA.....	9
2.1 Seguridad Informática.....	9
2.2 Ataque Informático	10
2.3 Vulnerabilidad de un Sistema Informático	11
2.3.1 Ataques más comunes.....	12
2.3.2 Técnicas de Aseguramiento de Código fuente.....	16
2.3.2.1 Técnica contra el Ataque SQL Injection	17
2.3.2.2 Técnica contra el Ataque Cross-Site-Scripting (XSS)	24
2.3.2.3 Técnica contra el Ataque Buffer Overflow (DOS).....	27
2.3.2.4 Técnica contra el Ataque Directorio Transversal.....	30
2.3.2.5 Técnica contra el Ataque de Intercepción Criptográfica	32
2.3.2.6 Técnica contra el Ataque del Día 0	35
2.4 Los Hackers	35
2.4.1 ¿Quiénes son?.....	35
2.4.2 ¿Quiénes no lo son?	36
CAPÍTULO 3	37
MATERIALES Y MÉTODOS.....	37

3.1 Diseño de la investigación	37
3.2 Tipo de estudio	37
3.3 Métodos	38
3.4 Técnicas	38
3.5 Planteamiento de la Hipótesis	39
3.6 Variables.....	39
3.7 Operacionalización de las Variables.....	39
3.7.1 Operacionalización Conceptual	39
3.7.2 Operacionalización Metodológica	40
3.8 Población y Muestra	41
3.9 Instrumentos de Recolección de Datos	42
3.10 Validación de Instrumentos	49
3.11 Ambientes de Prueba	49
CAPÍTULO 4	53
RESULTADOS Y DISCUSIÓN	53
4.1 Procesamiento de la Información	53
4.2 Resumen de los experimentos de la variable independiente.....	59
4.3 Resumen de los experimentos de la variable dependiente	60
4.4 Resumen de Equivalencias.....	65
4.5 Prueba de Hipótesis	66
4.6 Propuesta de Guía	67
I. Paso 1: Establecer un módulo de Auditoría de accesos	68
II. Paso 2: Análisis de vulnerabilidades de Código Fuente	70
III. Paso 3: Políticas de acceso al sistema operativo del servidor.....	73
IV. Paso 4: Políticas de claves de “todos” los servicios	74
CONCLUSIONES	75
RECOMENDACIONES	77
RESUMEN	78
SUMMARY	79
GLOSARIO DE TÉRMINOS	80
BIBLIOGRAFÍA	84
ANEXOS	89

ÍNDICE DE FIGURAS

Figura 1.1: Total de sitios en todos los dominios (Agosto 2005 – Mayo 2011) ..	3
Figura 1.2: Estadísticas de Incidentes reportados a CERT (corte Marzo 2011).	4
Figura 1.3: Logos de hackers.....	6
Figura 2.1:Código Vulnerable a SQL Injection	17
Figura 2.2:Resultado de la consulta.....	17
Figura 2.3:Resultado del ataque	19
Figura 2.4:Aseguramiento con PreparedStatement	20
Figura 2.5:Resultado del ataque con la técnica	20
Figura 2.6: Validador de Nombres y Códigos	21
Figura 2.7: Clase Validador.....	22
Figura 2.8: Código de Validación de Entradas.....	22
Figura 2.9: Código final con ESAPI y PreparedStatement	23
Figura 2.10: Funcionalidad de ESAPI	23
Figura 2.11:Código Vulnerable.....	24
Figura 2.12: Ataque Xss.....	24
Figura 2.13: Resultado Ataque Xss.....	24
Figura 2.14: Filtro de caracteres especiales	25
Figura 2.15: Resultado Proceso con filtrado	25
Figura 2.16: Utilización ESAPI en filtro de HTML.....	26
Figura 2.17: Resultado Proceso con filtrado ESAPI.....	27
Figura 2.18: Resultado Proceso con filtrado ESAPI.....	27
Figura 2.19: Código vulnerable a Buffer Overflow.....	28
Figura 2.20: Ataque Buffer Overflow	28
Figura 2.21: Resultado Ataque Buffer Overflow	29
Figura 2.22: Filtro de tamaño de variables	29
Figura 2.23: Resultado Ataque Buffer Overflow con la Técnica.....	29
Figura 2.24: Código vulnerable a DPT(Directory Path Transversal)	30
Figura 2.25: Ataque mediante URL y DPT.....	30
Figura 2.26: Resultado y Código fuente de la página	31
Figura 2.27: Técnica de DPT	32
Figura 2.28: Resultado con Técnica de DPT	32
Figura 2.29: Código Vulnerable.....	33
Figura 2.30: Resultado Wireshark (clave en texto plano).....	33
Figura 2.31: Técnica para evitar la interceptación.....	34
Figura 2.32: Resultado Wireshark con técnica.....	34
Figura 3.1: Sistema de la EPEC – SISEPEC	41
Figura 3.2:Sistema de la EPEC sin Técnicas de Seguridad– SisepecVulnerable	42
Figura 3.3:Logo Nessus	42
Figura 3.4:Logo Nikto.....	43
Figura 3.5:Interfaz Burpsuite	44
Figura 3.6 :Logo Wapiti	44
Figura 3.7:Logo Skipfish	45

Figura 3.8: Interfaz Grendel	46
Figura 3.9: Logo WebSecurify	46
Figura 3.10: Logo XSSS	47
Figura 3.11: Logo AlienValut - OSSIM	48
Figura 3.12: Escenario Sin técnicas de Aseguramiento	51
Figura 3.13: Escenario Con Técnicas de Aseguramiento	52
Figura 4.1: Representación del comportamiento de la fórmula 2	57
Figura 4.2: Representación del comportamiento de la fórmula 3	58
Figura 4.3: Comparación de Confidencialidad	61
Figura 4.4: Comparación de la Integridad	63
Figura 4.5: Comparación de la Disponibilidad	64
Figura 4.6: Resumen de Indicadores	66
Figura 4.7: Comparación de la Seguridad	66
Figura 4.8: Estructura del manual	68
Figura 4.9: Bloqueos de Lista Negra	70

ÍNDICE DE TABLAS

Tabla 2.I: Categorización de Vulnerabilidades	15
Tabla 3.I: Operacionalización Conceptual.....	39
Tabla 3.II: Operacionalización Metodológica	40
Tabla 3.III: Sistemas scanner por tipo de vulnerabilidad que encuentra	49
Tabla 3.IV: Equipos utilizados en las pruebas.....	50
Tabla 3.V: Software utilizado en las pruebas	50
Tabla 4.I: Tabla para recolección de Datos por Scanner	53
Tabla 4.II: Tabla de categorización de vulnerabilidades por indicador	54
Tabla 4.III: Equivalencias de la Severidad (G)	55
Tabla 4.IV: Tabla de valores para la ecuación $Y = 301 + (X10)$	56
Tabla 4.V: Tabla de valores para la ecuación $Y = 401 + X$	58
Tabla 4.VI: Resumen de las variables utilizadas en las fórmulas planteadas ..	59
Tabla 4.VII: Cuadro de vulnerabilidades de Confidencialidad recolectadas...	60
Tabla 4.VIII: Aplicación fórmula 2 para cada Herramienta	61
Tabla 4.IX: Cuadro de vulnerabilidades de Integridad recolectadas	62
Tabla 4.X: Aplicación fórmula 3 para cada Herramienta	62
Tabla 4.XI: Cuadro de vulnerabilidades de Disponibilidad recolectadas	63
Tabla 4.XII: Aplicación fórmula 3 para cada Herramienta	64
Tabla 4.XIII: Cuadro comparativo resumido	65

CAPÍTULO 1

INTRODUCCIÓN

Existen varios factores que afectan la seguridad informática, entre esos, está la programación del código fuente de los sistemas web, que al no tomar las precauciones y medidas necesarias, generará un producto con vulnerabilidades.

Los ataques informáticos hacia las aplicaciones web, son un tema que viene desde algunos años atrás, desde que los sistemas software empezaron a tener una funcionalidad distribuida en un medio abierto y sin protección de la información como es la Internet, desde entonces se han creado protocolos y modelos para asegurar la información, en todo ámbito de los sistemas, empezando por las redes que son utilizadas para el transporte de los datos, seguridades en los sistemas operativos, en los repositorios de información como son las bases de datos y finalizando con las aplicaciones que procesan y ponen a disposición la información.

La seguridad informática en las aplicaciones web, es una característica o requisito que para el usuario final es imperceptible, lo que más le interesa es que el sitio web esté atractivo y potente, sin embargo, el que el sistema esté a disposición de ser accedido desde cualquier parte del mundo incrementa las

posibilidades de que no solamente los usuarios del sistema intenten acceder, sino también los denominados hackers, los mismos que intentan vulnerar las restricciones y obtener información a la que no están autorizados, dentro de esta generación de hackers existen los denominados “hackers de sombrero blanco”, que el propósito es ayudar a eliminar las vulnerabilidades y construir sistemas seguros, pero también existen los “hackers de sombrero negro” que entre sus motivaciones para atacar están [23]:

Dinero, ventaja económica, ventaja competitiva, espionaje político, espionaje industrial, sabotaje, etc.

- Empleados descontentos, fraudes, extorsiones, “insiders”.
- Espacio de almacenamiento, ancho de banda, servidores de correo (SPAM), poder de cómputo, etc.
- Objetivo de oportunidad
 - obtener fama
 - experimentar

Si bien se mencionó antes, este tema tiene ya una historia considerable, también es cierto que a medida que avanza el tiempo nuevos sistemas y herramientas son incorporados al mercado los mismos que ingresan con vulnerabilidades que con el pasar del tiempo y su uso se van corrigiendo o como los informáticos lo denominan “Parchando”, así entonces podemos encontrar debilidades como [4]:

- Agujeros de seguridad en los sistemas operativos.
- Agujeros de seguridad en las aplicaciones.
- Errores en las configuraciones de los sistemas.

- Los usuarios carecen de información respecto al tema.

Esta lista podría seguir extendiéndose a medida que se evalúen mayor cantidad de elementos de un Sistema Informático. En caso de que la empresa u organización detecte un ataque esta no se puede permitir el lujo de denunciar ataques a sus sistemas, pues el nivel de confianza de los clientes bajaría enormemente.

Según el sitio Netcrat, el registro de hosts para sitios web ha ido incrementando exponencialmente durante los últimos años como podemos observar en la Figura 1.1 que publica este sitio, esto demuestra la inclinación de las aplicaciones a funcionar en la web y descentralizar los procesos mejorando la movilidad, disponibilidad y mantenimiento de las mismas.

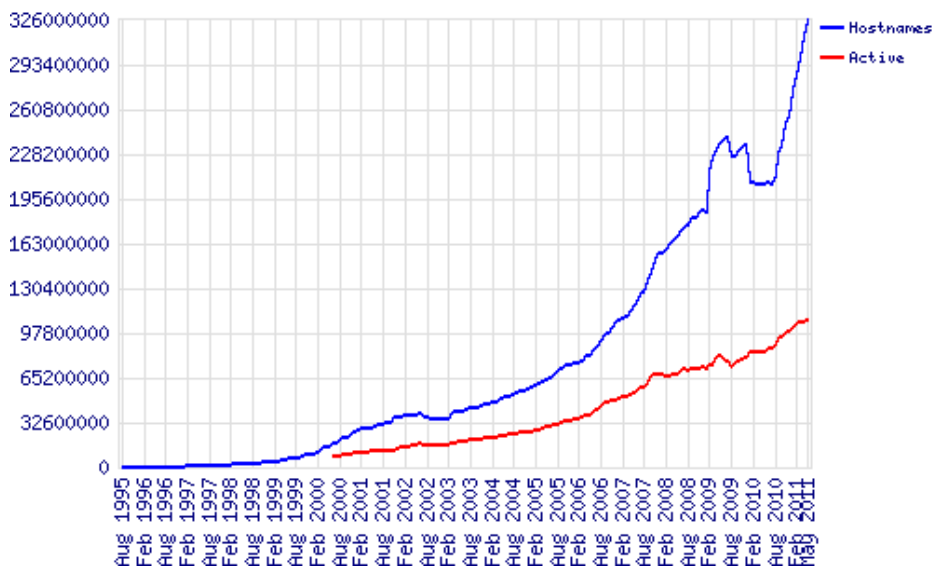


Figura 1.1: Total de sitios en todos los dominios (Agosto 2005 – Mayo 2011)

Mediante estadísticas tomadas del “Centro de Estudios Resposta e Tratamiento de Incidentes de Segurança no Brasil” (Centro de Estudios de Respuesta y Tratamiento de Incidentes de Seguridad en Brasil), los ataques reportados hasta marzo del 2011 se representan en la Figura 1.2 [13]

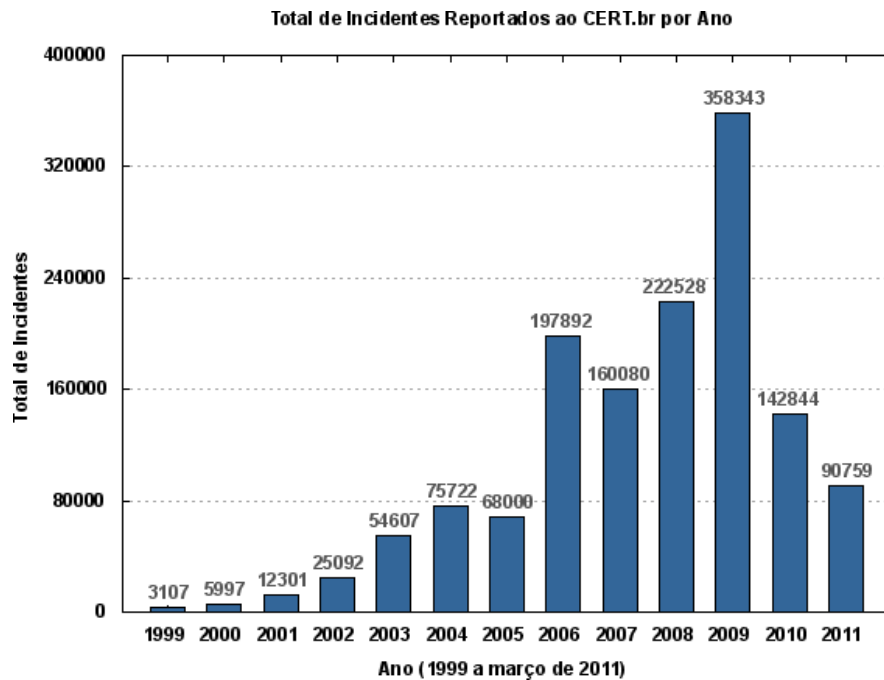


Figura 1.2: Estadísticas de Incidentes reportados a CERT (corte Marzo 2011)

Sólo como ejemplo, el Centro de Investigación y Seguridad Nacional de México (Cisen) [11] informó que del 1 de enero de 2007 al 31 de enero de 2011 sufrió 211 mil 262 ataques cibernéticos, lo que significa que cada año fue atacado en 52 mil 815 ocasiones, un promedio de 145 al día.

Partiendo de esta información, la seguridad informática hoy en día es un tema imprescindible para los administradores de las aplicaciones web, debido a que así como aumentan la calidad de los sistemas, la potencialidad de los ataques y detección de vulnerabilidades aumenta, para todo esto se debe estar

preparado tanto en la prevención, detección como con la reacción ante dichos ataques, todo esto lleva a disminuir el riesgo de ser atacados y como consecuencia reducción del riesgo de pérdidas para la empresa.

Dentro de todo este tema de los ataques informáticos el presente trabajo se enfocará en las vulnerabilidades que un sistema posee y que se pueden explotar por haber utilizado una mala programación del código fuente o a su vez no haber previsto los diferentes ataques que por falencias del lenguaje, sistema operativo, servidor web u otros se necesita filtrar o contrarrestarlos.

Si bien la aplicación web puede realizar el proceso correctamente, individuos no autorizados, tomando caminos diferentes pueden alterar este proceso, obteniendo o modificando información privilegiada, lo que a la postre puede significar pérdidas para la empresa pudiendo ser estas, económicas, de imagen, tiempo, o hasta en extorsión.

Todo esto conlleva a que los Administradores tienen cada vez mayor conciencia respecto de la seguridad de sus sistemas y arreglan por sí mismos las deficiencias detectadas.

Cuando un ataque ocurre, los costos que puede causar a la víctima son los siguientes^[23]:

Costos económicos (perder oportunidades de negocio).

- Costos de recuperación.
- Costos de reparación.
- Costos de tiempo.
- Costos legales y judiciales.

- Costos de imagen.
- Costos de confianza de clientes.
- Pérdidas humanas (cuando sea el caso).

Actualmente se encuentra en producción el sistema académico administrativo de la EPEC, con el cual se automatizan los procesos de este departamento, así permite el acceso a la información por parte de todos los actores de los programas de postgrado:

Estudiantes, Docentes, Usuarios Empleados de la Escuela.

Esta aplicación así como la del portal web son destinos de ataques de hackers y en 2 oportunidades (Figura 1.3) han vulnerado la seguridad, lo cual ha contribuido para que este análisis se proponga como caso de estudio prioritario.



Figura 1.3: Logos de hackers

1.1 Justificación

El presente trabajo tiene como fin encontrar técnicas adecuadas para eliminar vulnerabilidades desde el código fuente del Sistema Académico Administrativo de la EPEC que fue desarrollado en JAVA y JSP, así pues se establecen las siguientes posibles técnicas:

- Técnica contra el ataque Cross-Site-Scripting (XSS)
- Técnica contra el ataque SQL injection.
- Técnica contra el ataque Command injection
- Técnica contra el ataque Buffer Overflow (DOS)
- Técnica contra el ataque de Directorio Transversal
- Técnica contra el ataque de Intercepción
- Técnica contra el ataque del Día 0

Sin embargo, luego de toda la investigación es importante citar una frase emitida por Eugene Spaffor: “El único sistema totalmente seguro es aquel que está apagado, desconectado, guardado en una caja fuerte de Titanio, encerrado en un búnker de concreto, rodeado por gas venenoso y cuidado por guardias muy armados y muy bien pagados. Aun así, no apostaría mi vida por él.”

1.2 Objetivos

1.2.1 Objetivo General

- Proponer técnicas de aseguramiento de aplicaciones web desarrolladas en JAVA.

1.2.2 Objetivos Específicos

- Investigar los diferentes ataques y soluciones de defensa, así como las métricas y herramientas para determinar el nivel de seguridad con el que cuenta un sitio web.
- Realizar una simulación de ataques al Sistema de la EPEC sin tener aplicado las técnicas de aseguramiento.
- Aplicar las técnicas de aseguramiento al Sistema de la EPEC.
- Proponer una guía de aseguramiento de aplicaciones web desarrolladas en JAVA.

CAPÍTULO 2

REVISIÓN DE LITERATURA

2.1 Seguridad Informática

Según Jeimy Cano [16] la seguridad informática es un continuo entre técnicas de hacking y análisis de riesgos, que permita a las organizaciones aprender de sus fallas de seguridad y fortalecer sus esquemas de seguridad, no para contar con mayores niveles de seguridad, sino para evidenciar el nivel de dificultad que deben asumir los intrusos para ingresar a los sistemas.

Según la revista electrónica RED [26], la seguridad informática puede entenderse como aquellas reglas técnicas y/o actividades destinadas a prevenir, proteger y resguardar lo que es considerado como susceptible de robo, pérdida o daño, ya sea de manera personal, grupal o empresarial. En este sentido, es la información el elemento principal a proteger, resguardar y recuperar dentro de las redes empresariales.

Según José Domínguez [28] la Seguridad Informática es la preservación de la disponibilidad, confidencialidad e integridad de los datos tratados por la aplicación.

En el presente trabajo la seguridad informática es un área de la informática que se encarga de disminuir el riesgo a atentar el CID, confidencialidad, Integridad, Disponibilidad de un sistema informático.

Por lo tanto un sistema es **Seguro** si tiene Confidencialidad, Integridad y Disponibilidad.

La seguridad informática es un tema que abarca varios roles como: [23]

- Usuarios: Son las principales víctimas, que es el eslabón más débil en la cadena de la seguridad, en gran parte por el desconocimiento, existe una frase “La Ingeniería Social es el ataque más común, debido a que no existe un Parche de Seguridad para los Humanos”
- Gerentes: La mayoría de las empresas incorporan medidas de seguridad hasta que han tenido graves problemas. ¿para que esperarse?
- Los Atacantes: Cualquiera conectado a la red es una víctima potencial, sin importar a que se dedique, debido a que muchos atacantes sólo quieren probar que pueden hacer un hack por diversión.
- Creadores de Sistemas: Aquí es donde se necesita de una correcta programación y configuración, que si bien consume esfuerzo, según estadísticas, con el 20% de esfuerzo se pueden lograr el 80% de resultados.

2.2 Ataque Informático

Según Nuria Matellanes [3] un ataque informático es todo acceso no autorizado a un sistema informático o red de comunicación electrónica de datos, que

además supone una agresión contra el interés del propietario o titular del sistema o de la información sea este la permanencia o entrada en el sistema.

Según Jorge Mieres [5], un ataque informático consiste en aprovechar alguna debilidad o falla (vulnerabilidad) en el software, en el hardware, e incluso, en las personas que forman parte de un ambiente informático; a fin de obtener un beneficio, por lo general de índole económico, causando un efecto negativo en la seguridad del sistema, que luego repercute directamente en los activos de la organización.

Según Carlos Alberto Parra Correa [19] y Jorge Mieres coinciden en que un ataque informático causa efecto negativo sobre un sistema informático, en el presente trabajo un ataque informático es explotar alguna debilidad o falla (vulnerabilidad) de un sistema informático con motivaciones económicas, experimentales, espionaje o alguna ventaja para el atacante.

2.3 Vulnerabilidad de un Sistema Informático

Según la OWASP Foundation [30]: una vulnerabilidad es un agujero o una debilidad en la aplicación, que puede ser un defecto de diseño o un error de aplicación, que permite a un atacante causar daño a las partes interesadas de una aplicación.

Según Toni Puig en su E-Curso “Gestión de riesgos de los sistemas de información” [12] Una vulnerabilidad es la potencialidad o posibilidad de ocurrencia de la materialización de una amenaza sobre un activo.

Los conceptos de OWASP Foundation y Toni Puig coinciden en que una vulnerabilidad es una amenaza con posibilidades de causar daño a la aplicación.

En el presente trabajo se considera que una vulnerabilidad informática es una debilidad de la cual se puede obtener o modificar información o el proceso sin ser autorizados a realizarlo.

2.3.1 Ataques más comunes

Los ataques más comunes existentes en la actualidad son los siguientes [25]:

- **Ingeniería Social**

Es la manipulación de las personas o usuarios que tengan acceso a la red interna para convencerlas de que ejecuten acciones o actos que normalmente no realizan, revelando todo lo necesario para superar las barreras de seguridad. Si el atacante tiene la experiencia suficiente, puede engañar fácilmente a un usuario (que desconoce las mínimas medidas de seguridad) en beneficio propio. Esta técnica es una de las más usadas y efectivas a la hora de averiguar nombres de usuarios y passwords de acceso a la red.

- **Ingeniería Social Inversa**

Consiste en la generación, por parte de los intrusos, de una situación inversa a la originada en Ingeniería Social.

El intruso intenta demostrar que es capaz de dar ayuda a los usuarios, y estos lo llaman ante algún imprevisto. El intruso aprovechara esta

oportunidad para obtener la información necesaria para solucionar el problema del usuario y el suyo propio.

- **Trashing (Cartoneo)**

Generalmente, un usuario anota su login y password en un papel y luego, cuando lo recuerda, lo arroja a la basura. Este procedimiento por más inocente que parezca es el que puede aprovechar un atacante para hacerse de una llave para entrar al sistema, es decir, “nada se destruye, todo se transforma”.

- **Ataques de Monitorización**

Este tipo de ataque se realiza para observar a la víctima y su sistema, con el objetivo de establecer sus vulnerabilidades y posibles formas de acceso futuro.

- **Ataques de Autenticación**

Este tipo de ataque tiene como objetivo engañar al sistema de la víctima para ingresar al mismo. Generalmente este engaño se realiza tomando las sesiones ya establecidas por la víctima u obteniendo su nombre de usuario y contraseña.

- **Denial of Service (DoS)**

Los ataques de Negación de Servicio tienen como objetivo saturar los recursos del sistema de la víctima de forma tal que se inhabilita los servicios brindados por la misma. El ataque más común es el Ping de la muerte, el cual consiste en el envío de más de 65535 bytes a la víctima congestionando el servicio de red.

- **Ataques de Modificación – Daño**

Son los que modifican o eliminan sin autorización datos o software instalados en el sistema víctima. Asociado a este ataque se produce una acción llamada “eliminación de huellas”, para no dejar rastro del atacante.

Dentro de todas estas categorías existen los ataques que explotan vulnerabilidades de código fuente en aplicaciones JAVA, entre ellos están:[³¹]

- Cross-Site-Scripting (XSS) – Cuando un atacante usa una aplicación web para enviar código malicioso en Java Script, mediante la utilización de la URL.
- SQL injection – Usa secuencias de comandos usando lenguaje SQL (StructuredQueryLanguage) y la URL del sitio Web.
- Command injection – Permite a un atacante pasar código malicioso generalmente escrito en Perl y Python a diferentes sistemas.
- Buffer Overflow – Cuando la capacidad de almacenamiento específico del buffer sobrepasa el límite el buffer se inunda. Java y J2EE no son susceptibles al Overflow, sin embargo bucles mal programados pueden causar el desbordamiento de memoria (buffer overflow).
- Directorio Transversal – La habilidad de navegar archivos y directorios con acceso no autorizado.
- Intercepción Criptográfica – Interceptar tráfico cifrado como SSL para intentar descifrar el contenido del mismo. Nada fácil, ya que SSL soporta varios tipos de algoritmos de criptografía.

- Ataque del Día 0 – Tiempo entre que la vulnerabilidad fue descubierta por un investigador o Hacker y el tiempo que le toma a la empresa publicar el parche para corregir la vulnerabilidad.

Luego, dentro de estas subcategorías podemos mencionar que:

Los ataques de Command Injection son utilizados para aplicaciones de escritorio en la que se realizan peticiones a la máquina virtual de JAVA, por lo cual no será considerado para el estudio, para más información se puede referir al sitio de OWASP [9].

Por lo tanto se buscará técnicas para evitar los siguientes ataques:

- Cross-Site-Scripting (XSS)
- SQL injection
- Buffer Overflow
- Directorio Transversal
- Intercepción Criptográfica
- Ataque del Día 0

La tabla 2.I identifica las vulnerabilidades relacionada con el tipo de impacto que produce, tomada del sitio “Instituto Nacional de Tecnologías de la Comunicación”[2]. Cabe mencionar que la severidad de cada vulnerabilidad es definida por la herramienta de escaneo de aplicaciones web.

Tabla 2.I: Categorización de Vulnerabilidades

Categoría	Vulnerabilidad
Confidencialidad	Secuencia de comandos en sitios cruzados - XSS
	Permisos, privilegios y/o control de acceso

	SQL Injection
	Salto de Directorio
	Error de Buffer
	Falsificación de sitios cruzados
	Claves en texto claro
	Inyección de código
Integridad	Permisos, privilegios y/o control de acceso
	SQL Injection
	Salto de Directorio
	Error de Buffer
	Falsificación en Sitios Cruzados
	Inyección de Código
Disponibilidad	Permisos, privilegios y/o control de acceso
	SQL Injection
	Salto de Directorio
	Error de Buffer
	Error en la gestión de recursos
	Falsificación en Sitios Cruzados

Elaborado por: Paúl Paguay

Fuente: Instituto Nacional de Tecnologías de la Comunicación

2.3.2 Técnicas de Aseguramiento de Código fuente

Para la explicación de las técnicas se verificará primero el proceso de ataque y posteriormente se planteará las técnicas para evitarlo.

2.3.2.1 Técnica contra el Ataque SQL Injection Ataque:

Planteado el siguiente código de la Figura 2.1:

```
<%  
String codigo= request.getParameter("id");  
  
String sql = "select codigo, nombres, apellidos from persona where codigo='"+codigo+"'";  
String url="jdbc:postgresql://localhost:5432/prueba";  
String user="postgres";  
String password="sql1";  
String driver="org.postgresql.Driver";  
Class.forName(driver);  
Connection cnn = DriverManager.getConnection(url, user, password) ;  
Statement smt = cnn.createStatement();  
ResultSet rs = smt.executeQuery(sql);  
out.println("<table border=1>");  
out.println("<tr><td>Codigo</td><td>Nombres</td><td>Apellidos</td></tr>");  
while(rs.next()) {  
    out.println("<tr><td>" +rs.getString(1) + "</td>" +  
        "<td>" +rs.getString(2) + "</td>" +  
        "<td>" +rs.getString(3) + "</td>" +  
        "</tr>");  
}  
out.println("<table>");  
%>
```

Figura 2.1: Código Vulnerable a SQL Injection

Y La URL de la aplicación es:

<http://localhost:8080/XssApp/sql/detalle vulnerable.jsp?id=1>

El mismo que da como resultado:

Código	Nombres	Apellidos
1	Paul	Paguay

[Regresar](#)

Figura 2.2: Resultado de la consulta

Con este proceso anteriormente mencionado, cumple la funcionalidad de mostrar la información de un usuario registrado en el sistema, pero analizando el código fuente existen vulnerabilidades como:

- No se filtra o se valida la lectura de parámetros
- Se utilizan Statements, enviando cadenas de texto completas hacia la base de datos

Ahora bien, se manipula la URL con la siguiente manera:

http://localhost:8080/XssApp/sql/detallevulnerable.jsp?id=1%27%20union%20select%20oid::character%20varying,%20relname,relname%20from%20pg_class%20where%20relkind%20=%20%27r%27%20and%20relnamespace=2200%20and%20oid%20not%20in%20%28%20select%20conrelid%20from%20pg_constraint%20where%20%20%20contype=%27f%27%20%29--

o lo que traducido sin codificaciones sería:

```
http://localhost:8080/XssApp/sql/detallevulnerable.jsp?id=1' union  
select oid, relname,relname from pg_class where relkind = 'r' and  
relnamespace=2200 and oid not in ( select conrelid from  
pg_constraint where contype='f' ) --
```

Se puede observar que luego del código utilizado para la consulta se añade otra sentencia mediante el comando UNION.

Uniendo el código fuente del programa con el ataque de sql injection, la sentencia final que ingresa a la base de datos sería la que se muestra a continuación, donde la parte en negrita es la sentencia de ataque:

```
select codigo, nombres, apellidos from persona where codigo='1'  
union select oid, relname,relname from pg_class where  
relkind = 'r' and relnamespace=2200 and oid not in ( select  
conrelid from pg_constraint where contype='f' ) --'
```

Luego de la ejecución se obtiene información de todas las tablas presentes en la base de datos que pertenezcan al esquema public (2200), dando como resultado lo que se muestra en la Figura 2.3:

Código	Nombres	Apellidos
1	Paul	Paguay
24869	tablaprueba	tablaprueba
36458	persona	persona

[Regresar](#)

Figura 2.3:Resultado del ataque

En este ejemplo solo visualizamos las tablas presentes en la base de datos, sin embargo las posibles consultas no tienen restricciones para el atacante.

Técnica:

Para esta técnica se ha tomado como base del Blog “Seguridad en Aplicaciones Web”, de Juan Díez-Yanguas Barber, [18].

- Utilización de PreparedStatements en lugar de Statement
- Validación de entradas

Utilización de PreparedStatements en lugar de Statements

Según Jason Lamen en su publicación “Prepared Statements and SQL injections” [24], señala que la utilización de PreparedStatements permite la validación del ingreso de la información que se obtiene mediante parámetros y no permite la inclusión de caracteres como los apóstrofes (') que son esenciales en el ataque sql injection.

Por otra parte los statements únicamente permite ingresar una sentencia sql a través de una cadena de texto, la misma que no se la puede validar.

El código con la utilización de Prepared Statement quedaría como se muestra en la Figura 2.4

```
<%
String codigo= request.getParameter("id");

String sql = "select codigo, nombres, apellidos from persona where codigo=?";
String url="jdbc:postgresql://localhost:5432/prueba";
String user="postgres";
String password="sql1";
String driver="org.postgresql.Driver";
Class.forName(driver);
Connection cnn = DriverManager.getConnection(url, user, password) ;
//Statement smt = cnn.createStatement();
PreparedStatement psmt= cnn.prepareStatement(sql);
psmt.setString(1, codigo);
ResultSet rs = psmt.executeQuery() ;
out.println("<table border=1>");
out.println("<tr><td>Codigo</td><td>Nombres</td><td>Apellidos</td></tr>");
while(rs.next()) {
    out.println("<tr><td>" +rs.getString(1) + "</td>" +
        "<td>" +rs.getString(2) + "</td>" +
        "<td>" +rs.getString(3) + "</td>" +
        "</tr>");
}
out.println("<table>");
%>
```

Figura 2.4:Aseguramiento con PreparedStatement

Y el resultado del mismo ataque realizado anteriormente sería lo que se muestra en la Figura 2.5, en la que no aparece ningún dato restringido:

Código	Nombres	Apellidos
Regresar		

Figura 2.5:Resultado del ataque con la técnica

Entonces como se ha visto la utilización de ***preparedstatements*** limita los ataques de sql injection, sin embargo, antes de procesar una transacción hacia la base de datos, se puede validar previamente dicha transacción. Dichas validaciones se lo puede lograr mediante la utilización de la API de Seguridad Empresarial ESAPI por sus siglas en inglés (OWASP Enterprise Security API).

Como lo menciona en su sitio OWASP [7], ESAPI es una librería de control de seguridad de aplicaciones web, libre y de código abierto, que hace más fácil a los programadores crear aplicaciones de bajo riesgo. De la misma manera, provee facilidad para la implementación de la seguridad en aplicaciones existentes. Por otro lado ESAPI también da soporte para ser implementado en varios lenguajes de programación, como Java EE, Dot NET, Classic ASP, PHP, etc.

A continuación se detallará como utilizarla:

Primero se agregará una nueva entrada en el archivo de validaciones, como se muestra en la Figura 2.6, en este ejemplo se validarán nombres (Name) y códigos (Codigo).

Para los Nombres se especifica que solo se aceptan caracteres alfabéticos, desde la A hasta la Z y con tildes, mientras que para los códigos Letras desde la A a la Z mayúsculas o minúsculas y números.

```
Validator.Name=^[A-Z][a-zA-Z -áÁéÉíÍóÓúÚüÜñÑ]+$  
Validator.Codigo=^[A-Z][a-zA-Z][0123456789]+$
```

Figura 2.6: Validador de Nombres y Códigos

Posterior se creará una clase para que ocupe las funciones de la API y las provea a la capa de presentación, como se muestra en la Figura 2.7

```
public class Validador {
    public static String validateName(String input)
        throws IntrusionException, ValidationException {
        Validator validador = ESAPI.validator();
        return validador.getValidInput("Nombre", input, "Name", 100, false);}
    public static String validateCodigo(String input)
        throws IntrusionException, ValidationException {
        Validator validador = ESAPI.validator();
        return validador.getValidInput("Código", input, "Codigo", 100, false);}
```

Figura 2.7: Clase Validador

Ahora, esta clase creada se lo utiliza en la capa de presentación para validar las entradas de los parámetros como se muestra en la Figura 2.8.

```
String codigo= request.getParameter("id");
boolean band=false;
try {
    codigo = Validador.validateCodigo(codigo);
    band=true;
} catch (ValidationException ex) {
    out.println("Error: "+ex.getUserMessage());
} catch (IntrusionException ex) {
    out.println("Error: "+ex.getUserMessage());
}
if(band){
```

Figura 2.8: Código de Validación de Entradas

El código finalmente quedaría de la siguiente manera como se muestra en la Figura 2.9

```
String codigo= request.getParameter("id");
boolean band=false;
try {
    codigo = Validador.validateCodigo(codigo);
    band=true;
} catch (ValidationException ex) {
    out.println("Error:"+ex.getUserMessage());
} catch (IntrusionException ex) {
    out.println("Error: "+ex.getUserMessage());
}
}

if(band){
String sql = "select codigo, nombres, apellidos from persona where codigo=?";
String url="jdbc:postgresql://localhost:5432/prueba";
String user="postgres";
String password="sql1";
String driver="org.postgresql.Driver";
Class.forName(driver);
Connection cnn = DriverManager.getConnection(url, user, password) ;
PreparedStatement psmt= cnn.prepareStatement(sql);
psmt.setString(1, codigo);
ResultSet rs = psmt.executeQuery() ;
out.println("<table border=1>");
out.println("<tr><td>Codigo</td><td>Nombres</td><td>Apellidos</td></tr>");
while(rs.next()){
    out.println("<tr><td>" +rs.getString(1) + "</td>"+
        "<td>" +rs.getString(2) + "</td>"+
        "<td>" +rs.getString(3) + "</td>"+
        "</tr>");
}
out.println("<table>");
}
```

Validación de
entradas (ESAPI)

Utilización
PreparedStatement

Figura 2.9: Código final con ESAPI y PreparedStatement

Luego de ejecutar el mismo ataque el resultado es el que se muestra en la Figura 2.10, donde se puede observar que actúa primero la funcionalidad de ESAPI, antes de enviar cualquier petición hacia la base de datos.

Error:Código: Invalid input. Please conform to regex `^[A-Z][a-zA-Z][0123456789]+$`

[Regresar](#)

Figura 2.10: Funcionalidad de ESAPI

2.3.2.2 Técnica contra el Ataque Cross-Site-Scripting (XSS)

Ataque

Tomando como ejemplo el código que se muestra en la Figura 2.11, donde se puede observar que se imprime un parámetro enviado desde una página anterior, sin embargo este es un hueco de seguridad del sistema, debido a que permite ingresar código malicioso.

```
<h1>XSS- Vulnerable</h1>  
Hola <%= request.getParameter("nombre") %>
```

Figura 2.11: Código Vulnerable

Con el código que se muestra en la Figura 2.12, el atacante puede introducir cualquier código malicioso para alterar el correcto funcionamiento de la aplicación.

Nombre:

Figura 2.12: Ataque Xss

La ejecución de código anterior dará como resultado lo que se muestra en la Figura 2.13, que es la ejecución de código malicioso en el sistema:

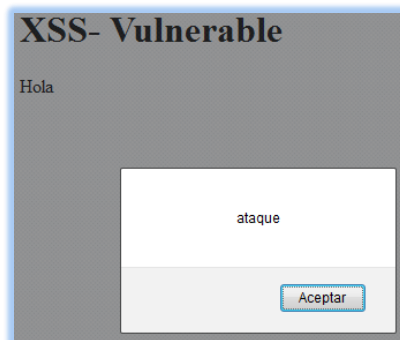


Figura 2.13: Resultado Ataque Xss

Técnica:

- Filtración de caracteres especiales en parámetros de formato establecido
- Filtración de etiquetas HTML.

Para evitar este ataque podemos filtrar los caracteres especiales como “< / % > (),....etc.” (ver Figura 2.14).

```
<h1>XSS- Vulnerable</h1>
Hola <%
//Establecemos los caracteres a filtrar
String[] caractereseliminar = {"<",">","/","%"};
String output= request.getParameter("nombre");
//De la cadena original eliminamos los caracteres maliciosos
for (int i=0; i<caractereseliminar.length; i++) {
    output = output.replace(caractereseliminar[i], "");
}
out.println(output);
%>
```

Figura 2.14: Filtro de caracteres especiales

El resultado de filtrar los caracteres y utilizando el mismo código malicioso de la Figura 2.12, será el de la Figura 2.15

```
Hola scriptalert('hola')script
```

Figura 2.15: Resultado Proceso con filtrado

Si bien esta filtración evita el ataque, se recomienda la utilización de la misma biblioteca ESAPI, para filtrar de la misma manera como se hizo con la técnica para el Ataque de SQL Injection abarcada en el punto 2.3.2.1.

Hasta este punto se puede confiar en que un parámetro que provenga de otra página se lo filtre y no cause problemas, sin embargo en el caso de que un usuario ingrese información de tipo “texto”, tales como enunciados, comentarios (por ejemplo una noticia), en la cual no se deba restringir ningún carácter en el ingreso, existe el conflicto entre que es lo que se permite y que no.

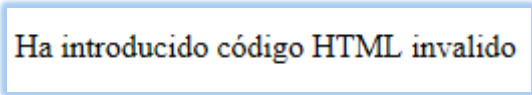
Para este caso no se necesita filtrar letras en particular, sino etiquetas HTML, para lo cual la librería ESAPI también ofrece una solución que se mostrará a continuación.

Como se muestra en la Figura 2.16 se ha utilizado ESAPI, para filtrar el código HTML malicioso ingresado en el parámetro noticia.

```
<%
String input =request.getParameter("noticia");
try { //No se admite una entrada vacía
    if (input.isEmpty() == true) {
        out.println("No se admite el campo vacío");
    } else{ //obtención archivo de configuración a través del classpath
        Policy politica = Policy.getInstance(Validador.class.
            getResource("/antisamy-tinymce-1.4.4.xml"));
        AntiSamy validator = new AntiSamy();
        //Antes de analizar la cadena es convertida en su forma canónica
        ESAPI.encoder().canonicalize(input);
        CleanResults cr = validator.scan(ESAPI.encoder().canonicalize(input), politica);
        //Si ha ocurrido un error se lanza una excepción indicando el error
        if (cr.getNumberOfErrors() != 0) {
            throw new IntrusionException("Ha introducido código HTML invalido",
                cr.getErrorMessages().get(0).toString());
        }else{
            out.println("Detalle Noticia: "+input);
        }
    }
} catch (PolicyException ex) {
    out.println(ex.getMessage());
} catch (ScanException ex) {
    out.println(ex.getMessage());
} catch (Exception ex) {
    out.println(ex.getMessage());
}
%>
```

Figura 2.16: Utilización ESAPI en filtro de HTML

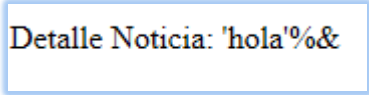
El resultado del ingreso del código malicioso de la Figura 2.12 será el de la Figura 2.17, donde aparecerá el mensaje de error al encontrar código malicioso:



Ha introducido código HTML invalido

Figura 2.17: Resultado Proceso con filtrado ESAPI

Sin embargo permitiendo el ingreso de varios caracteres especiales como por ejemplo: 'hola'%& el resultado será el de la Figura 2.18 donde de puede observar que se ha permitido caracteres especiales restringiendo solamente etiquetas HTML.



Detalle Noticia: 'hola'%&

Figura 2.18: Resultado Proceso con filtrado ESAPI

2.3.2.3 Técnica contra el Ataque Buffer Overflow (DOS)

Este tipo de ataque según la investigación de Tom Olzak [³²], manifiesta que debido al tipo de la arquitectura de los entornos de desarrollo de JAVA y .NET en la que los programadores no administran la asignación de memoria, las aplicaciones escritas en Java y .NET son casi inmunes a los ataques de Buffer Overflow.

Sin embargo no se puede confiar en que totalmente se encuentra libre de este tipo de ataque.

Por lo cual se lo analizará el procedimiento del ataque y cómo evitarlo mediante la técnica.

Ataque:

Tomando como ejemplo el código de la Figura 2.19 en la que se imprime un parámetro obtenido de otra página, se puede evidenciar un código vulnerable a BufferOverflow, debido a que no realiza una validación del valor enviado.

```
<h1>Buffer Overflow</h1>
Hola <%
String output=request.getParameter("nombre");
out.println(output);
%>
```

Figura 2.19: Código vulnerable a Buffer Overflow

Para atacar esta vulnerabilidad, se lo realizará con la siguiente manera mostrada en la Figura 2.20, basado en el video GeoHTTP Remote Buffer OverFlow and DOS [29], donde se envía un parámetro con un tamaño extremadamente grande, lo cual sobrecargará al servidor y generará Deniego de Servicio.

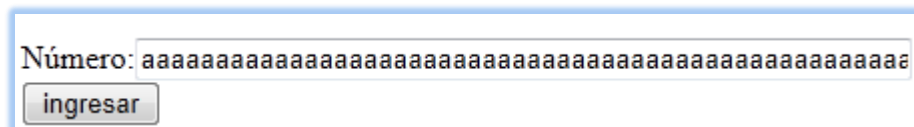
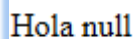
A screenshot of a web form. It features a text input field with the label "Número:" followed by a long string of 'a' characters. Below the input field is a button labeled "ingresar".

Figura 2.20: Ataque Buffer Overflow

El resultado de este ataque se lo visualiza en la Figura 2.21, donde se verifica que el servidor Web se ha bloqueado (Ejemplo realizado con el Navegador Web Internet Explorer 7).

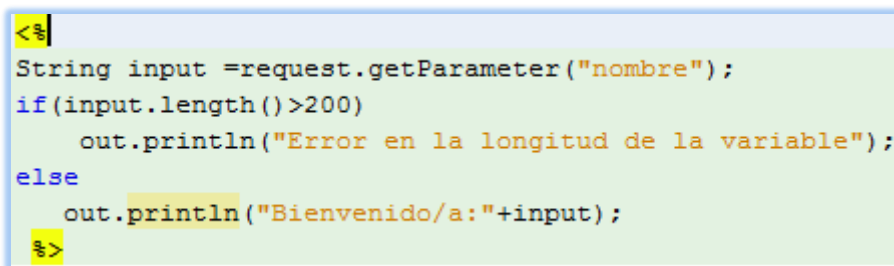


Hola null

Figura 2.21: Resultado Ataque Buffer Overflow

Técnica:

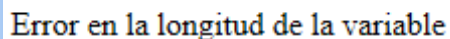
Filtrando la longitud del valor del parámetro como se muestra en la Figura 2.22 se evita la saturación de la memoria del servidor, aunque la misma plataforma se protege de este tipo de ataques es recomendable tomar las medidas necesarias para no sobrecargar al servidor y ser víctimas de ataques de Deniego de Servicio – DOS, donde el servidor dejará de funcionar por la cantidad de información que debe manejar en memoria.



```
<%  
String input =request.getParameter("nombre");  
if(input.length()>200)  
    out.println("Error en la longitud de la variable");  
else  
    out.println("Bienvenido/a:"+input);  
%>
```

Figura 2.22: Filtro de tamaño de variables

Como resultado de la protección del ataque se muestra en la Figura 2.23.



Error en la longitud de la variable

Figura 2.23: Resultado Ataque Buffer Overflow con la Técnica

2.3.2.4 Técnica contra el Ataque Directorio Transversal

Ataque:

Tomando como ejemplo el código vulnerable mostrado en la Figura 2.24 se observa que se recibe un parámetro por ejemplo “detalle.jsp” y de acuerdo a ese parámetro muestra la página correspondiente.

```
<body>
  <h1>Cabecera</h1><hr />
  <%
    String op = request.getParameter("op");
  %>
  <jsp:include page="<&#x27;<%=op%>"/>
</body>
```

Figura 2.24: Código vulnerable a DPT(Directory Path Transversal)

Atacando este código mediante la URL como se muestra en la Figura 2.25 se procederá a acceder a otra ubicación importante como es el archivo de configuración de la aplicación, donde por lo general se guardan datos sensibles tales como cadenas de conexión, perfiles de usuario, librerías, destinos, etc.

```
http://10.1.1.3:8084/XssAppTomcat/dpt/menu.jsp?op=../WEB-INF/web.xml
```

Figura 2.25: Ataque mediante URL y DPT

El resultado del ataque se puede observar en la Figura 2.26 donde aparece información del archivo de configuración del sitio web, y si se revisa el código fuente de la página se observará en su totalidad el archivo atacado, con lo que se ha vulnerado el sistema.



Figura 2.26: Resultado y Código fuente de la página

Técnica:

Para evitar este tipo de ataque, se podría filtrar las variables que ingresan mediante la librería ESAPI, y creando una nueva entrada dentro del archivo de configuración de validaciones (abordado en la técnica para Sql Injection), sin embargo también podemos optar por la utilización de otro método de llamadas a otras páginas como se muestra en la Figura 2.27.

```
<body>
  <h1>Cabecera</h1><hr />
  <%
    String op = request.getParameter("op");
    if(op.equals("detalle.jsp")){
  %>
  <jsp:include page="<%=op%>" />
  <%
    }else{
      out.println("P gina desconocida");
    }
  %>
</body>
```

Figura 2.27: T cnica de DPT

Ahora si se intenta vulnerar mediante DPT se obtendr  el siguiente mensaje como se muestra en la Figura 2.28, donde se muestra la informaci n.

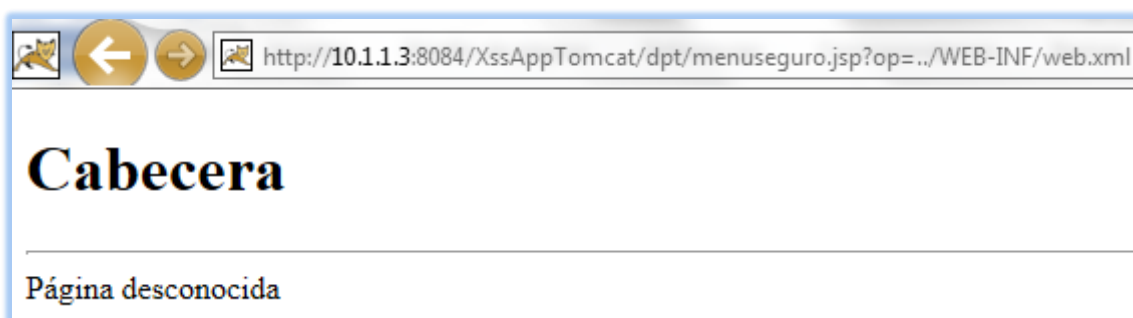


Figura 2.28: Resultado con T cnica de DPT

2.3.2.5 T cnica contra el Ataque de Intercepci n Criptogr fica

Ataque:

Visualizando el c digo fuente de la Figura 2.29 en la que existe un formulario para autenticaci n del usuario, se puede observar que existe vulnerabilidad de no cifrado de informaci n sensible, por lo cual puede ser robada y utilizada para suplantar identidad.

```
<form action="procesavulnerable.jsp" name="form" id="form" method="post" >  
INICIO DE SESI&Oacute;N DEL USUARIO <br />  
C&eacute;dula:<input type="text" maxlength="20" name="txtCedula" id="txtCedula"><br />  
Clave:<input type="password" name="txtClave" id="txtClave"><br />  
<input type="submit" name="btnIngresar" id="btnIngresar" value="Ingresar">  
</form>
```

Figura 2.29: Código Vulnerable

Para este ataque se utilizará la herramienta Wireshark, la misma que permite “sniffear” o “husmear”, la red en busca de información sensible como logins y claves de usuarios del sistema.

El atacante lo que hace es permitir a la computadora obtener todos los paquetes de la red y luego filtrar de acuerdo a varios parámetros como podemos observar en la Figura 2.30, donde se ha obtenido información del tráfico de la red, y entre esa información se encuentra el paquete de autenticación donde se envía en texto plano tanto el usuario como la clave, lo cual deja expuesto a robo de información.

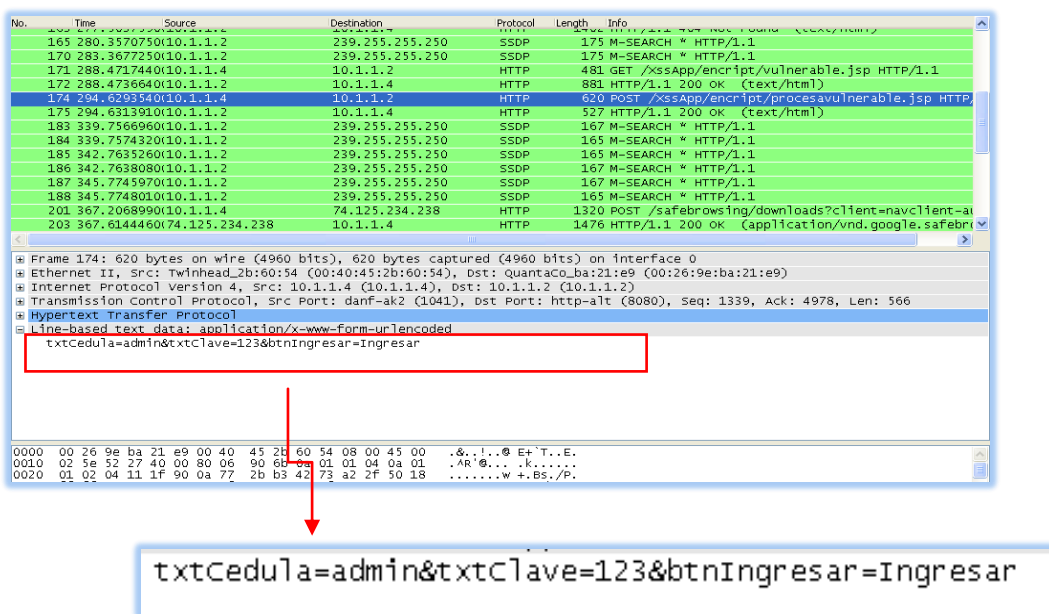


Figura 2.30: Resultado Wireshark (clave en texto plano)

Técnica:

Para evitar este ataque se propone manejar todo dato sensible de una manera encriptada para que en caso de que viaje por la red y existan ataques la información no pueda ser vista fácilmente.

Como se puede observar en la Figura 2.31 se hace uso de javascript para encriptar mediante MD5 la clave luego de presionar en el botón de submit, posteriormente dicha clave viajará por la red hasta el servidor para continuar con el proceso de autenticación, de la misma manera la clave en el servidor deberá ser almacenada en forma encriptada.

```
<form action="procesaseguro.jsp" name="form" id="form" onsubmit="encriptar()" method="post" >
  INICIO DE SESI&Oacute;N DEL USUARIO <br />
  C&eacute;dula:<input type="text" maxlength="20" name="txtCedula" id="txtCedula"><br />
  Clave:<input type="password" name="txtClave" id="txtClave"><br />
  <input type="submit" name="btnIngresar" id="btnIngresar" value="Ingresar">
</form>
```

Figura 2.31: Técnica para evitar la interceptación

Como resultado de la autenticación y visualizando la clave que viaja por la red, la podemos observar con la herramienta Wireshark en la Figura 2.32.

342 370.5160680(74.125.234.228) 10.1.1.4 HTTP 634 HTTP/1.1 200 OK (application/vnd.google.safebr
344 370.5479520(10.1.1.4) 74.125.234.228 HTTP 1270 GET /safebrowsing/rd/chFnb29nLXB0aXNoLXNoYXZhc
352 370.9148840(74.125.234.228) 10.1.1.4 HTTP 289 HTTP/1.1 200 OK (application/vnd.google.safebr
354 370.9159150(10.1.1.4) 74.125.234.228 HTTP 1109 GET /safebrowsing/rd/chFnb29nLXB0aXNoLXNoYXZhc
448 372.2698840(74.125.234.228) 10.1.1.4 HTTP 487 HTTP/1.1 200 OK (application/vnd.google.safebr
478 510.8778400(10.1.1.4) 10.1.1.2 HTTP 477 GET /xssApp/encrypt/seguro.jsp HTTP/1.1
486 510.9432910(10.1.1.2) 10.1.1.4 HTTP 397 HTTP/1.1 200 OK (text/html)
488 516.4911260(10.1.1.4) 10.1.1.2 HTTP 641 POST /xssApp/encrypt/procesaseguro.jsp HTTP/1.1
489 516.4939070(10.1.1.2) 10.1.1.4 HTTP 568 HTTP/1.1 200 OK (text/html)

Frame 488: 641 bytes on wire (5128 bits), 641 bytes captured (5128 bits) on interface 0
Ethernet II, Src: Twinhead_2b:60:54 (00:40:45:2b:60:54), Dst: quantaco_ba:21:e9 (00:26:9e:ba:21:e9)
Internet Protocol Version 4, Src: 10.1.1.4 (10.1.1.4), Dst: 10.1.1.2 (10.1.1.2)
Transmission Control Protocol, Src Port: fptip (1045), Dst Port: http-alt (8080), Seq: 424, Ack: 7644, Len: 587
Hypertext Transfer Protocol
Line-based text data: application/x-www-form-urlencoded
txtCedula=admin&txtClave=202cb962ac59075b964b07152d234b70&btnIngresar=Ingresar

0000 00 26 9e ba 21 e9 00 40 45 2b 60 54 08 00 45 20 .&.!.@ E+ T..E.
0001 02 73 52 f3 40 00 80 05 8f 88 0a 01 01 04 0a 91 58 @

txtCedula=admin&txtClave=202cb962ac59075b964b07152d234b70&btnIngresar=Ingresar

Figura 2.32: Resultado Wireshark con técnica

Para reforzar esta encriptación se puede realizar algunas combinaciones, de 3MD5 con eliminación de caracteres, utilización de llave pública y privada, pero por lo menos que se garantice que los datos sensibles no se transmitirán en texto plano.

2.3.2.6 Técnica contra el Ataque del Día 0

Este tipo de ataque sucede cuando por parte del proveedor del servidor web, base de datos o la plataforma java presenta problemas de código fuente, por lo cual no se podría realizar ninguna acción mientras el proveedor ponga a disposición los parches respectivos para solucionar el inconveniente.

Por tal motivo la técnica o mas bien recomendación sería de esperar un tiempo prudencial hasta que se establezca el nuevo producto y haya salido los parches correspondientes, ejemplo migrar de una plataforma JAVA 1.6 a JAVA 1.7, hasta que hayan solucionado todos los problemas, para de esa forma no ser el “conejiillo de indias” para encontrar fallas en el producto.

2.4 Los Hackers

2.4.1 ¿Quiénes son?

Según Eric Steven Raymond en su publicación “Cómo convertirse en hacker” [10], menciona :“Existe una comunidad, una cultura compartida, de programadores expertos y magos de las redes, cuya historia se remonta décadas atrás a los tiempos de los primeros miniordenadores de tiempo compartido y los tempranos experimentos con ARPAnet. Los miembros de esta cultura crearon el término "hacker". Los hackers construyeron Internet.

Los hackers hicieron de Unix el sistema operativo que es hoy día. Los hackers hacen andar Usenet. Los hackers hacen funcionar la WWW. Si eres parte de esta cultura, si has contribuido a ella y otras personas saben quién eres y te llaman hacker, entonces eres un hacker”.

2.4.2 ¿Quiénes no lo son?

En la misma publicación Eric Steven Raymond menciona: “Existe otro grupo de personas que se llaman a sí mismos hackers, pero que no lo son. Son personas (generalmente varones adolescentes) que se divierten irrumpiendo ilegalmente en ordenadores y haciendo "phreaking" en el sistema telefónico. Los auténticos hackers tienen un nombre para esas personas: "crackers", y no quieren saber nada de ellos. Los auténticos hackers opinan que la mayoría de los crackers son perezosos, irresponsables y no muy brillantes, y fundamentan su crítica en que ser capaz de romper la seguridad no le hace a uno un hacker, de la misma manera que ser capaz de arrancar un coche con un puente en la llave no le convierte en ingeniero de automotores.

Desafortunadamente, muchos periodistas y escritores utilizan erróneamente la palabra "hacker" para describir a los crackers; esto causa enorme irritación a los auténticos hackers.”

CAPÍTULO 3

MATERIALES Y MÉTODOS

3.1 Diseño de la investigación

La investigación será Cuasi Experimental ya que primero se recopiló toda la información referente a los diferentes ataques posibles que puede ser víctima un sitio web JAVA-JSP. Posteriormente se procedió a implementar un sitio web para que automatice los procesos de la Escuela de Postgrado y Educación Continua utilizando la plataforma JAVA-JSP. Sobre esa implementación se agregó selectivamente técnicas de aseguramiento de código fuente según las métricas definidas para este trabajo. Al final, estas diferentes implementaciones de seguridad fueron evaluadas cuantitativamente para sacar nuestras propias conclusiones y recomendaciones, evaluando si la implementación de las técnicas generó los resultados esperados, como es el mejoramiento de la seguridad de la aplicación web JAVA-JSP. Cabe mencionar que los ataques fueron intencionales para demostrar la seguridad en ambas aplicaciones.

3.2 Tipo de estudio

Por la naturaleza de la investigación se considera que el tipo de estudio que se va a realizar es una **investigación descriptiva y aplicada** ya que se realizará un estudio de técnicas de aseguramiento para implementar en una aplicación

web JAVA-JSP luego se realizarán ataques intencionalmente al sitio web sin incorporar las técnicas y posteriormente se realizarán nuevos ataques intencionales al sitio web con las técnicas incorporadas, con los resultados de los 2 escaneos se verificará si la seguridad de la aplicación mejoró, de tal modo de encontrar una guía para asegurar las aplicaciones web desarrolladas en JAVA, específicamente en JSP.

3.3 Métodos

Este proyecto utiliza los siguientes métodos de investigación.

Método Científico: Se utiliza este método para recopilar la información necesaria para determinar las técnicas de aseguramiento de vulnerabilidades de programación que puedan ayudar a mejorar la seguridad en una aplicación web. Estas técnicas serán aplicadas al nuevo sistema con el cual se pretende disminuir las vulnerabilidades.

Método Deductivo: Debido que al estudiar la técnicas de aseguramiento, se trata de encontrar un esquema que ayude a mejorar la Seguridad de un Sitio web desarrollado en JAVA - JSP.

Método Comparativo: ya que se debe comparar el sistema con las técnicas incorporadas y el mismo sistema sin técnicas incorporadas.

3.4 Técnicas

Además se ha utilizado ciertas técnicas, entre ellas están:

- Observación

- Razonamiento
- Recopilación de información.
- Análisis
- Pruebas

3.5 Planteamiento de la Hipótesis

El uso de las técnicas de aseguramiento de vulnerabilidades de programación en JAVA, mejora la seguridad de una aplicación web operativa

3.6 Variables

De acuerdo a la hipótesis se han identificado dos variables:

- **Variable Independiente:**
 - Técnicas de aseguramiento de vulnerabilidades de programación en JAVA
- **Variable Dependiente:**
 - Seguridad de una Aplicación Web

3.7 Operacionalización de las Variables

3.7.1 Operacionalización Conceptual

Tabla 3.I: Operacionalización Conceptual

VARIABLE	TIPO	DEFINICION
Técnicas de aseguramiento de vulnerabilidades de programación en JAVA	INDEPENDIENTE	Conjunto de algoritmos y buenas prácticas que aseguran las vulnerabilidades de programación de una aplicación web operativa.
Seguridad de una aplicación Web	DEPENDIENTE	Cantidad de vulnerabilidades que un sistema web presenta.

3.7.2 Operacionalización Metodológica

Tabla 3.II: Operacionalización Metodológica

HIPÓTESIS	VARIABLES	INDICADORES	INSTRUMENTOS
El uso de las técnicas de aseguramiento de vulnerabilidades de programación en JAVA, mejorará la seguridad de una aplicación web operativa.	Uso de Técnicas de Aseguramiento de vulnerabilidades de programación en JAVA	Técnicas de Aseguramiento aplicados	Hardware Software Código Fuente
	Seguridad de una aplicación web	I1: Confidencialidad	Software de monitoreo (Nessus, Nikto, Burpsuite, Wapiti, Skipfish, Grendel, Websecurity, Xss) Ejecución de algoritmos maliciosos
		I2: Integridad	Software de monitoreo (Nessus, Nikto, Burpsuite, Wapiti, Skipfish, Grendel, Websecurity, Xss) Ejecución de algoritmos maliciosos
		I3: Disponibilidad	Software de monitoreo (Nessus, Nikto, Burpsuite, Wapiti, Skipfish, Grendel, Websecurity, Xss) Ejecución de algoritmos maliciosos

3.8 Población y Muestra

La población es el conjunto total de individuos, objetos o medidas que poseen algunas características comunes observables en un lugar y en un momento determinado, constituye el objeto de investigación, de donde se extrae la información requerida para el estudio; es decir, en este caso para nuestro objetivo la población serán todas las aplicaciones web desarrolladas en JAVA-JSP.

Luego seleccionamos una proporción representativa de la población, que permita generalizar los resultados de una investigación. El objetivo principal de la selección de la muestra es extraer información que resulta imposible estudiar en la población, porque esta incluye la totalidad.

Para el presente trabajo se utilizará una muestra NO ALEATORIA ya que el estudio se enfoca en el Sistema Académico Administrativo de la EPEC - <http://sisepec.esPOCH.edu.ec>, la misma que por contar con el código fuente para incorporar las técnicas de aseguramiento se ha escogido para la comprobación, donde el procesamiento de información será la misma todo dependiendo si se aplicó o no las técnicas.



Figura 3.1: Sistema de la EPEC – SISEPEC



Figura 3.2: Sistema de la EPEC sin Técnicas de Seguridad– Sisepec Vulnerable

3.9 Instrumentos de Recolección de Datos

Para encontrar las vulnerabilidades de ambas aplicaciones se utilizarán sistemas de escaneo de vulnerabilidades como:

- Backtrack
 - **Nessus**



Figura 3.3: Logo Nessus

Nessus es una gran herramienta diseñada para automatizar las pruebas y el descubrimiento de problemas de seguridad conocidos. Normalmente alguien, un grupo de hackers, una empresa de seguridad, o un investigador descubre una manera específica para violar la seguridad de un producto de software. El descubrimiento puede ser accidental o través de la investigación dirigida, la vulnerabilidad, en los diferentes niveles de detalle, se

libera a la seguridad de la comunidad. Nessus está diseñado para ayudar a identificar y resolver estos problemas conocidos, antes que un hacker se aproveche de ellos. Nessus es una gran herramienta con muchas posibilidades.[¹⁷]

- **Nikto**



Figura 3.4:Logo Nikto

Es un escáner de código abierto (GPL) para servidores web que realiza pruebas completas contra los servidores web para varios artículos, incluyendo más de 6500 archivos / CGIs potencialmente peligrosos, los controles de versiones no actualizadas de más de 1250 servidores, y los problemas específicos de la versión de más de 270 servidores. También comprueba los elementos de configuración del servidor, tales como la presencia de múltiples archivos de índice, opciones de servidor HTTP, y tratará de identificar los servidores web y de software instalados. Elementos de exploración y plugins se actualizan con frecuencia y se pueden actualizar de forma automática.[²²]

- **Burpsuite**

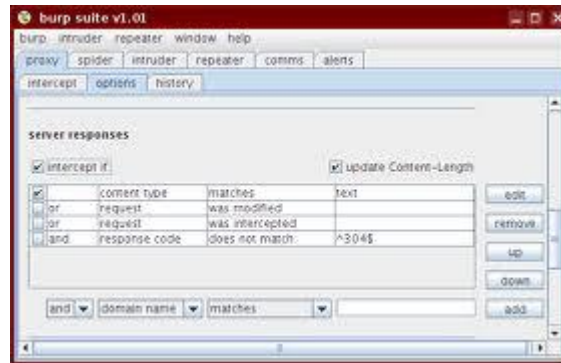


Figura 3.5: Interfaz Burpsuite

Burp Suite es una plataforma integrada para la realización de las pruebas de seguridad de aplicaciones web. Sus diversas herramientas funcionan perfectamente juntos para apoyar el proceso de prueba, a partir de la cartografía y el análisis iniciales de la superficie de ataque de una aplicación, hasta la búsqueda y explotación de vulnerabilidades de seguridad.^[6]

- **Wapiti**



Figura 3.6 : Logo Wapiti

Wapiti le permite auditar la seguridad de sus aplicaciones web. Realiza "recuadro negro", es decir, no estudia el código fuente de la aplicación, pero va a escanear las páginas de la aplicación

web desarrollada en busca de secuencias de comandos y las formas en que se puede inyectar datos.

Una vez que se obtiene esta lista, Wapiti actúa como un fuzzer, la inyección de cargas útiles para determinar si un script es vulnerable.^[33]

- **Skipfish**



Figura 3.7:Logo Skipfish

De código abierto, totalmente automático, la aplicación activa de seguridad web herramienta de reconocimiento.

Alta velocidad: escrito en C puro, con un manejo muy optimizado HTTP y un tamaño mínimo de la CPU, la herramienta fácilmente alcanza 2.000 solicitudes por segundo con los objetivos de respuesta.

Facilidad de uso: la herramienta cuenta con la heurística para apoyar una variedad de frameworks y páginas web de tecnología mixta, con esta función automática de aprendizaje on-the-fly creación lista de palabras, y formulario de autocompletado.

De seguridad lógica de vanguardia: alta calidad, bajo en falsos positivos de seguridad, comprobación de seguridad diferencial

capaz de detectar una serie de defectos sutiles, incluyendo los vectores de inyección a ciegas.[²¹]

- **Grendel**



Figura 3.8: Interfaz Grendel

Utiliza motores de búsqueda populares para las direcciones URL y las incorpora en el análisis para encontrar la superficie adicional del sitio. También es una de las únicas herramientas que hacen algo de detección de páginas 404 que es un requisito indispensable si se quiere reducir los falsos positivos.[²⁷]

- **Websecurify**



Figura 3.9:Logo WebSecurify

Es una solución avanzada de pruebas construida para identificar los problemas de seguridad de las aplicaciones web.

Entre las características están:

- Disponible para todos los principales sistemas operativos (Windows, MAC, OS, Linux) incluyendo los dispositivos móviles (iPhone, Android)
- Interfaz gráfica fácil de usar
- Construido en apoyo a la internacionalización
- Extensible con la ayuda de plugins

Forma parte de la suite de Backtrack y también es ampliamente utilizado para la seguridad de las aplicaciones como lo mencionan y utilizan en los libros “BackTrack 4: Assuring Security by Penetration Testing”^[3] y The Basics of Hacking and Penetration Testing: Ethical Hacking and Penetration Testing Mode Easy.^[1]

○ **Xsss**



Figura 3.10:Logo XSSS

Es un escáner de cross site scripting (Xss) por fuerza bruta.

Básicamente esta herramienta ataca por las URL de las aplicaciones web escaneando las vulnerabilidades con indexación.

- **OSSIM**



Figura 3.11: Logo AlienValut - OSSIM

Open Source Security Information Management (OSSIM), es una consola de seguridad de código abierto, bajo la plataforma unix y que ha sido probada por mas de 22.000 usuarios a nivel mundial, compuesta por 22 herramientas líderes en el campo de la seguridad de sistemas.

Compuesto por:

- Servidor OSSIM (Consola de Gestión)
- Framework (Interacción entre Módulos)
- Base de datos de OSSIM (Eventos)
- Agentes (Sondas Colectoras)^[14]

- Pruebas de scripts maliciosos

Con las herramientas antes detalladas también se cuenta con scripts maliciosos que una vez encontrados las vulnerabilidades se pueden explotar las mismas, sin embargo la finalidad del presente documento no es la “penetración” de sistemas sino el encontrar el porcentaje de vulnerabilidades de seguridad que se han cerrado con las técnicas de aseguramiento.

En la tabla III se ha clasificado a los sistemas de escaneo antes mencionados en categorías de obtención de datos de acuerdo al tipo de vulnerabilidades que identifica o escanea.

Tabla3.III: Sistemas scanner por tipo de vulnerabilidad que encuentra

Tipo de Vulnerabilidad	Herramienta Software
Confidencialidad	Burpsuite, Grendel, Wapiti, Skipfish, Código
Integridad	XSS, WebSecurify, Grendel, Nikto, Wapiti
Disponibilidad	Wapiti

Elaborado por: Paúl Paguay

3.10 Validación de Instrumentos

Los instrumentos anteriormente mencionados son de software libre que registran miles de descargas y son utilizados cotidianamente por los hackers para hallar y explotar vulnerabilidades en sistemas web, como se mencionan en las publicaciones de “Cloud Vulnerability Assessment” donde señala que la suite de Backtrack ofrece una gran ventaja por las herramientas que posee^[8]. En el libro BackTrack 4 “Assuring Security by Penetration Testing” donde se manifiesta que hasta el 19 de Julio del 2010, se habían descargado 1.5 millones de usuarios ^[3].

3.11 Ambientes de Prueba

De acuerdo a los procedimientos generales establecidos se ha determinado la utilización de un ordenador portátil con software Backtrack 5 y otro con el sistema OSSIM conectado a una red que tiene acceso al servidor de la aplicación Sisepec, el mismo que servirá para realizar el escaneo de la aplicación sin las técnicas aplicadas y con las técnicas.

Ambos escenarios contarán con el mismo hardware y software, de las siguientes características:

Hardware

Tabla 3.IV: Equipos utilizados en las pruebas

Cantidad	Equipo	Marca	Modelo	Especificaciones
1	Ordenador	HP	Pavilion dv6	Procesador Core i3 4G Memoria RAM 500G Disco Duro
1	Ordenador	Averatec	2200	Procesador 1.8 (Dual COre) 1G Memoria RAM 500G Disco Duro
1	Ordenador	VMWare	8	Procesador COre i3 1G Memoria RAM 40 G Disco Duro
1	Modem	DLINK	DSL-524B	4 puertos Conexión a internet

Elaborado por: Paúl Paguay

Software

Tabla 3.V: Software utilizado en las pruebas

Nombre	Descripción
Suite Backtrack	Sistema operativo con herramientas de escaneo pre instaladas
OSSIM	Sistema Operativo de aseguramiento de Sistemas de Información.
Centos 5	Sistema Operativo para servidores de

	aplicaciones
GlassFish 3.1	Servidor de Aplicaciones JAVA
Postgresql	Servidor de Base de Datos

Elaborado por: Paúl Paguay

En la Figura 3.12 y 3.13 se muestran los 2 escenarios para las pruebas con la diferencia en el Servidor SISEPEC, con el sitio aplicado las técnicas y sin aplicarlas.

Escenario 1

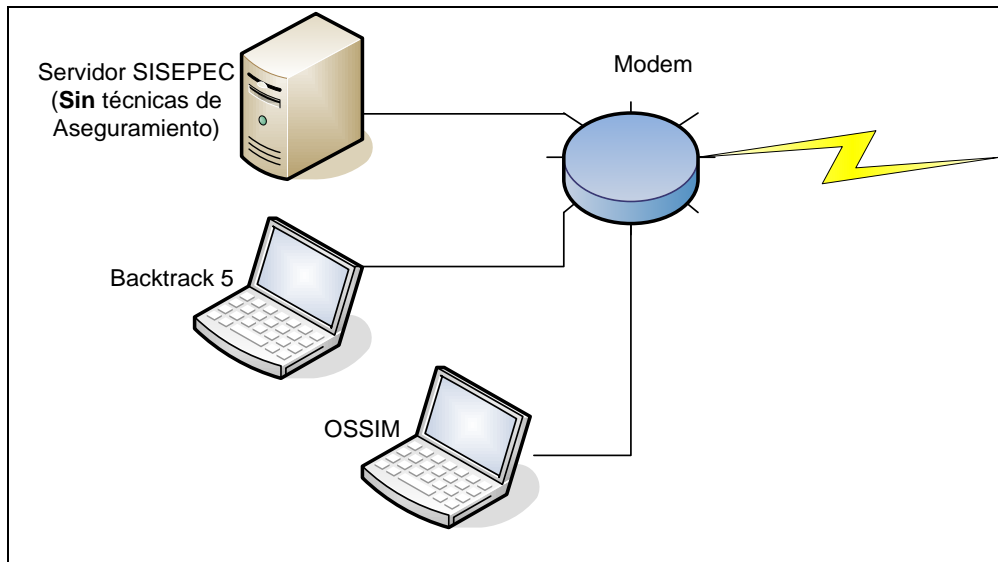


Figura 3.12: Escenario Sin técnicas de Aseguramiento

Escenario 2:

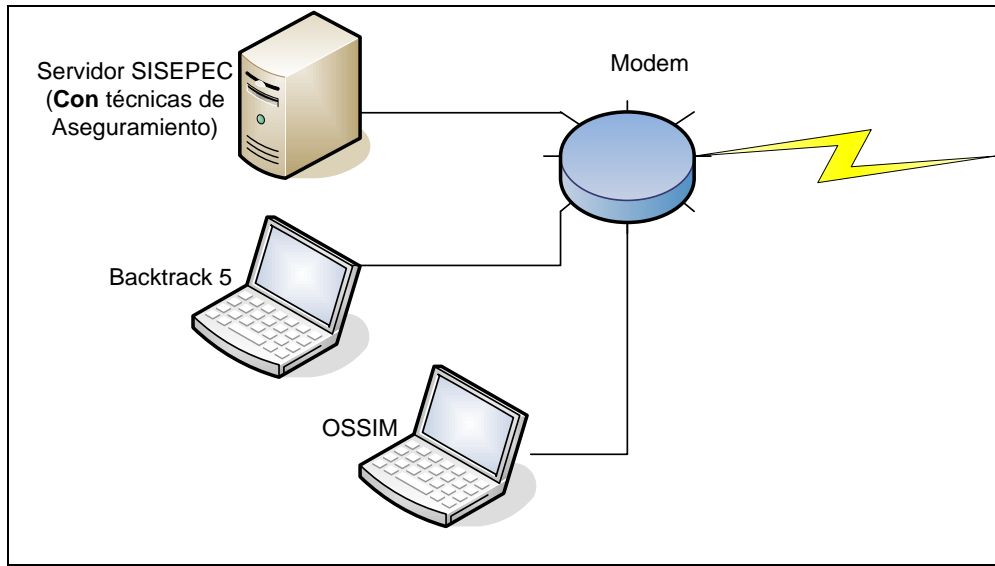


Figura 3.13:Escenario Con Técnicas de Aseguramiento

CAPÍTULO 4

RESULTADOS Y DISCUSIÓN

4.1 Procesamiento de la Información

Para las tablas y una mejor visualización se utilizará la siguiente abreviatura:

- S: Sistema Sisepec con las técnicas implementadas
- SV: Sistema Sisepec sin las técnicas implementadas.

Para la recolección de datos se realizará con la Tabla 4.I, la misma que se detalla en el anexo 1.

Tabla 4.I: Tabla para recolección de Datos por Scanner

Herramienta	Nombre vulnerabilidad	Sistema EPEC			
		Sist.		Sist.V.	
		#	G	#	G
Xxxx	Xxxx	X	X	X	X

Elaborado por: Paúl Paguy

Donde:

Categoría Vulnerabilidad: (Confidencialidad, Integridad y Disponibilidad)

Herramienta: Será el software o el ataque utilizado

Indicador: Valor del escaneo o ataque

Sist.: Sistema con técnicas de aseguramiento

Sist.V.: Sistema sin técnicas de aseguramiento

G: Severidad del ataque con la siguiente escala mostrada en la tabla 4:

Se creó la Tabla 4.II para la recolección de las vulnerabilidades encontradas de acuerdo a los indicadores planteados en la operacionalización de las variables (Pag. 22)

Tabla 4.II: Tabla de categorización de vulnerabilidades por indicador

Indicador: (Confidencialidad, Integridad, Disponibilidad)					
Herramienta	Nombre vulnerabilidad	Sist(Con Técnicas)		Sist.V. (Sin Técnicas)	
		[Vc,Vi,Vd]	G	[Vc,Vi,Vd]	G
-----	-----	--	--	--	--

Elaborado por: Paúl Paguay

Donde:

Herramienta: Será la herramienta de monitoreo o el ataque utilizado

Nombre vulnerabilidad: Nombre de la vulnerabilidad

S: Sistema con técnicas de aseguramiento

SV: Sistema sin técnicas de aseguramiento

G: Severidad del ataque con la siguiente escala mostrada en la tabla V

[Vc,Vi,Vd]: Número de vulnerabilidades encontradas Vc= Vulnerabilidades de confidencialidad, Vi=Vulnerabilidades de Integridad, Vd= Vulnerabilidades de Disponibilidad.

En la tabla 4.III se ha generado la categorización de la Severidad de las vulnerabilidades, dado que las herramientas de scaneo generan 4 tipos de vulnerabilidades, así como se menciona en el sitio de INTECO^[1]:

Muy Bajo o Informativo: Se pueden visualizar información como, servidores que pueden estar levantados en el servidor, protocolos, que no generan riesgo o que son muy generales.

Bajo: La aplicación permite ver datos como puerto de escucha del servidor como 80 (http), 5432 (postgres), 3306 (mysql), etc.. Datos que deberían estar ocultos.

Medio: Puertos abiertos y aceptando conexiones, información del servidor como versiones, plataformas, etc. Datos que con un análisis más profundo se pueden explotar vulnerabilidades.

Alto: Cuando ya se tiene una vulnerabilidad, y se puede explotar la vulnerabilidad con scripts o herramientas de penetración.

Con la categorización anterior se ha procedido a ponderar cada uno de los tipos de severidades.

Tabla 4.III: Equivalencias de la Severidad (G)

Valor	Equivalencia	Descripción
0	Muy Bajo	Visible Información sin riesgo
1	Bajo	Visible Datos comunes
2	Medio	Visible Datos Importantes
3	Alto	Vulnerabilidad expuesta

Elaborado por: Paúl Paguay

Fuente: Instituto Nacional de Tecnologías de la Comunicación

Para la valoración de la Seguridad se utilizarán las siguientes fórmulas propuestas por el autor, definiendo una igualdad entre la Seguridad la Confidencialidad, Integridad, Disponibilidad, Vulnerabilidades y Severidad.

En primer lugar, partiendo de la definición de La seguridad de la información según ISO 27001, es “la preservación de la confidencialidad, integridad y disponibilidad de la información”. Entonces la seguridad será igual a la suma de la confidencialidad (C), Integridad (I) y la Disponibilidad (D). (Fórmula 1)

$$S = C + I + D \quad (1)$$

En el presente trabajo, a la integridad por la relevancia de esta característica representará el 40% de la seguridad, la confidencialidad un 30% y la Disponibilidad un 30%, dando un total de la sumatoria de 100%.

La Fórmula 2 representa la **confidencialidad (C)** que será el 30 % de la seguridad dividido para la sumatoria de todas las vulnerabilidades de

confidencialidad (V_c) multiplicada por su severidad. Donde si $V_c = 0$ el máximo de la confidencialidad será 30%.

$$C = \frac{30}{1 + (V_c * G/10)} \quad (2)$$

Representando mediante valores la fórmula 2, tendría la siguiente estructura:

$$Y = \frac{30}{1 + (X/10)}$$

Donde **Y** sería la variable dependiente y **X** la independiente.

Tabla 4.IV: Tabla de valores para la ecuación $Y = \frac{30}{1 + (\frac{X}{10})}$

X	Y
0	30
1	27,27
2	25
3	23,07
4	21,42
...	...

La gráfica para el conjunto de datos sería la que se muestra en la Figura 4.1, donde se puede evidenciar que mientras una aplicación siga aumentando el número de vulnerabilidades, así como su severidad, la seguridad irá cayendo, cabe mencionar que los valores para X serán del conjunto de los números naturales.

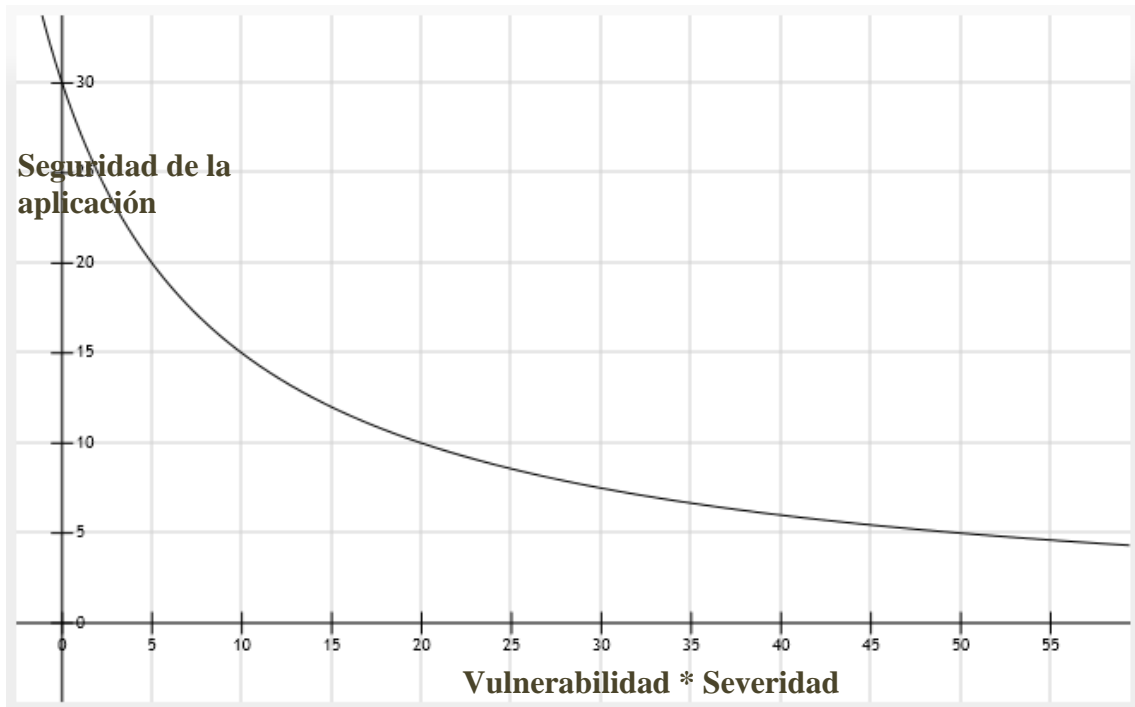


Figura 4.1: Representación del comportamiento de la fórmula 2

La Fórmula 3 representa la **Integridad (I)** que será el 40 % de la seguridad dividido para la sumatoria de todas las vulnerabilidades de integridad (V_i) multiplicada por su severidad. Donde si $V_i = 0$ el máximo de la integridad será 40%.

$$I = \frac{40}{1 + (V_i * G/10)} \quad (3)$$

Representando mediante valores la fórmula 2, tendría la siguiente estructura:

$$Y = \frac{40}{1 + X}$$

Donde **Y** sería la variable dependiente y **X** la independiente.

Tabla 4.V: Tabla de valores para la ecuación $Y = \frac{40}{1+X}$

X	Y
0	40
1	36,36
2	33,33
3	30,76
4	28,57
...	...

La gráfica para el conjunto de datos sería el que se muestra en la Figura 4.2, donde se puede evidenciar que mientras una aplicación siga aumentando el número de vulnerabilidades, así como su severidad (Variable X), la seguridad irá cayendo, cabe mencionar que los valores para X serán del conjunto de los naturales.

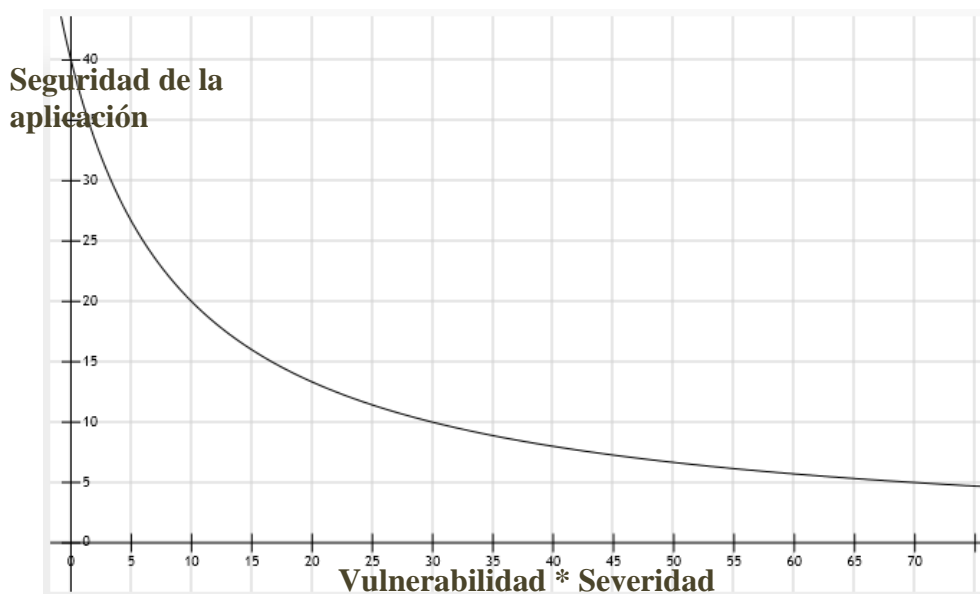


Figura 4.2: Representación del comportamiento de la fórmula 3

La Fórmula 4 representa la **Disponibilidad (D)** que será el 30 % de la seguridad dividido para la sumatoria de todas las vulnerabilidades de

disponibilidad (Vd) multiplicada por su severidad. Donde si Vd =0 el máximo de la disponibilidad será 30%.

$$D = \frac{30}{1 + (Vd * G/10)} \quad (4)$$

La representación gráfica y comportamiento de la fórmula de la disponibilidad sería la misma que la que se usó para la confidencialidad, visualizar Figura 4.1.

Tabla 4.VI: Resumen de las variables utilizadas en las fórmulas planteadas

Resumen de Variables
C: Confidencialidad
I: Integridad
D: Disponibilidad
Vc: Número de Vulnerabilidades de Confidencialidad
Vi: Número de Vulnerabilidades de Integridad
Vd: Número de Vulnerabilidades de Disponibilidad
G: Severidad de la vulnerabilidad
S: Seguridad

Elaborado por: Paúl Paguay

4.2 Resumen de los experimentos de la variable independiente

Se han monitoreado las aplicaciones con 8 herramientas software de escaneo de aplicaciones web para encontrar vulnerabilidades, sean estos de Confidencialidad, Integridad y Disponibilidad.

Por otro lado se han implementado las técnicas de aseguramiento al sistema SISEPEC, mientras que el sistema SISEPECVULNERABLE, solo ha mantenido su funcionamiento sin uso de las técnicas de blindado del sistema.

4.3 Resumen de los experimentos de la variable dependiente

Luego del escaneo con las herramientas de detección de vulnerabilidades (Ver anexo 1) en los sistemas con las técnicas de aseguramiento y sin las mismas se ha obtenido los siguientes comportamientos de la variable dependiente.

CONFIDENCIALIDAD

Tabla 4.VII: Cuadro de vulnerabilidades de Confidencialidad recolectadas

Indicador 1: Confidencialidad					
Herramienta	Nombre vulnerabilidad	Sistema con Técnicas		Sistema Sin Técnicas	
		Vc	G	Vc	G
Burpsuite	Claves en texto claro en la red	0	3	1	3
XSS	Vulnerabilidades de XSS	0	2	10	2
WebSecurify	Vulnerabilidad de XSS	3	2	18	2
Grendel	Vulnerabilidad de XSS	0	2	1	2
	Permisos, privilegios y/o control de acceso	0	2	1	2
Nikto	Permisos, privilegios y/o control de acceso	2	2	37	2
Wapiti	Vulnerabilidades de XSS	5	2	255	2
	Inyección de Código	0	3	2	3
Skipfish	Vulnerabilidades de Xss	1	3	2	3
	Vulnerabilidades de Xss	0	2	3	2
Scripts	Inyección de Código	0	3	1	3

Elaborado por: Paúl Paguay

Luego de la obtención de vulnerabilidades de confidencialidad con las herramientas de escaneo se procedió a aplicar la fórmula 2 al resultado obtenido con cada herramienta, posterior se ha calculado el promedio de la confidencialidad del total de las herramientas como se observa en la Tabla 4.VIII.

Tabla 4.VIII: Aplicación fórmula 2 para cada Herramienta

Indicador 1: Confidencialidad						
Herramienta	Sistema con Técnicas			Sistema Sin Técnicas		
	Vc	G	30	Vc	G	30
			$1 + (Vc * G/10)$			$1 + (Vc * G/10)$
Burpsuite	0	3	30	1	3	23.08
XSS	0	2	30	10	2	10
WebSecurify	3	2	18.75	18	2	6.52
Grendel	0	2	30	1	2	21.43
	0	2		1	2	
Nikto	2	2	21.43	37	2	3.57
Wapiti	5	2	15	255	2	0.57
	0	3		2	3	
Skipfish	1	3	23.08	2	3	13.64
	0	2		3	2	
Scripts	0	3	30	1	3	23.07
Promedio Confidencialidad			24.78%	Promedio Confidencialidad		12.74%

La confidencialidad del Sistema CON Técnicas es de un 24.78%.

La confidencialidad del Sistema SIN Técnicas es de un 12.74%.

La confidencialidad del sistema ha incrementado 12.04% con respecto del que no se ha incorporado las técnicas como se representa en la figura 4.3.

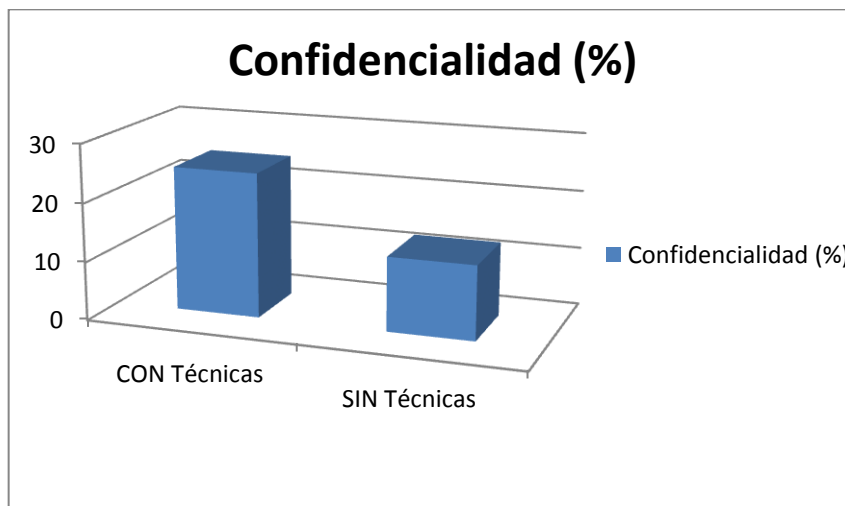


Figura 4.3: Comparación de Confidencialidad

INTEGRIDAD

Tabla 4.IX: Cuadro de vulnerabilidades de Integridad recolectadas

Indicador 2: Integridad					
Herramienta	Nombre vulnerabilidad	Sistema con Técnicas		Sistema Sin Técnicas	
		Vi	G	Vi	G
Grendel	Permisos, privilegios y/o control de acceso	0	2	1	2
Nikto	Permisos, privilegios y/o control de acceso	2	2	37	2
Wapiti	Vulnerabilidades de XSS	5	2	255	2
	Inyección de Código	0	3	2	3
Skipfish	Vulnerabilidades de Xss	1	3	2	3
	Vulnerabilidades de Xss	0	2	3	2
Scripts	Inyección de Código	0	3	1	3

Elaborado por: Paúl Paguay

Luego de la obtención de vulnerabilidades de integridad con las herramientas de escaneo se procedió a aplicar la fórmula 3 al resultado obtenido con cada herramienta, posterior se ha calculado el promedio de la integridad del total de las herramientas como se observa en la Tabla 4.X.

Tabla 4.X: Aplicación fórmula 3 para cada Herramienta

Indicador 2: Integridad						
Herramienta	Sistema con Técnicas			Sistema Sin Técnicas		
	Vi	G	$\frac{40}{1 + (Vi * G/10)}$	Vi	G	$\frac{40}{1 + (Vi * G/10)}$
Grendel	0	2	40	1	2	33.33
Nikto	2	2	28.57	37	2	4.76
Wapiti	5	2	20	255	2	0.77
	0	3		2	3	
Skipfish	1	3	30.76	2	3	18.18
	0	2		3	2	
Scripts	0	3	40	1	3	30.78
Promedio Integridad		31.87%		Promedio Integridad		17.56%

La integridad del Sistema CON Técnicas es de un 31.87%.

La integridad del Sistema SIN Técnicas es de un 17.56%.

La integridad del sistema ha incrementado 14.31% con respecto del que no se ha incorporado las técnicas como se representa en la figura 4.4.

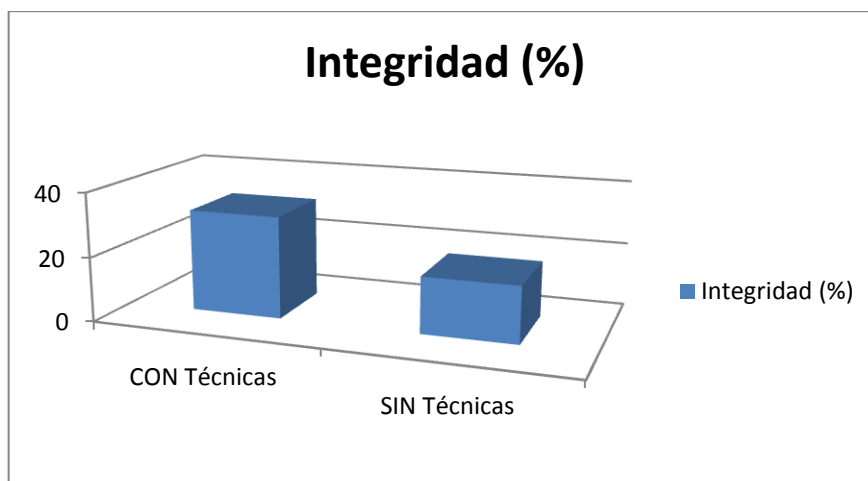


Figura 4.4: Comparación de la Integridad

DISPONIBILIDAD

Tabla 4.XI: Cuadro de vulnerabilidades de Disponibilidad recolectadas

Indicador 3: Disponibilidad					
Herramienta	Nombre vulnerabilidad	Sistema con Técnicas		Sistema Sin Técnicas	
		Vd	G	Vd	G
Grendel	Permisos, privilegios y/o control de acceso	0	2	1	2
Nikto	Permisos, privilegios y/o control de acceso	2	2	37	2
Wapiti	Error en gestión de recursos	2	2	1127	2

Elaborado por: Paúl Paguay

Luego de la obtención de vulnerabilidades de disponibilidad con las herramientas de escaneo se procedió a aplicar la fórmula 3 al resultado

obtenido con cada herramienta, posterior se ha calculado el promedio de la disponibilidad del total de las herramientas como se observa en la Tabla 4.XII.

Tabla 4.XII: Aplicación fórmula 3 para cada Herramienta

Indicador 3: Disponibilidad						
Herramienta	Sistema con Técnicas			Sistema Sin Técnicas		
	Vd	G	$\frac{30}{1 + (Vd * G/10)}$	Vd	G	$\frac{30}{1 + (Vd * G/10)}$
Grendel	0	2	30	1	2	25
Nikto	2	2	21.43	37	2	3.57
Wapiti	2	2	21.43	1127	2	0.13
Promedio Disponibilidad			24.28%	Promedio Disponibilidad		9.57%

La disponibilidad del Sistema CON Técnicas es de un 24.28%.

La disponibilidad del Sistema SIN Técnicas es de un 9.57%.

La disponibilidad del sistema ha incrementado 14.71% con respecto del que no se ha incorporado las técnicas como se representa en la figura 4.5.

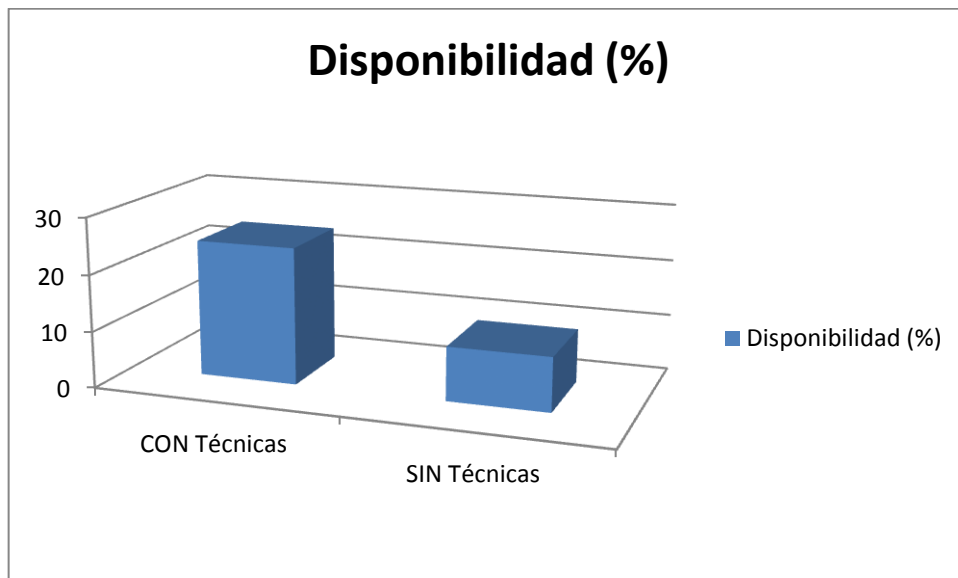


Figura 4.5: Comparación de la Disponibilidad

4.4 Resumen de Equivalencias

Remplazando los valores obtenidos en la sección anterior en la fórmula 1, la seguridad resulta:

SEGURIDAD Sistema **CON** Técnicas (Remplazando en Fórmula 1)

$$Ssisepec = Cs + Is + Ds$$

$$Ssisepec = 24.78 + 31.87 + 24.28$$

$$Ssisepec = 80.93\%$$

SEGURIDAD SISEPEC **SIN** Técnicas (Remplazando en Fórmula 1)

$$Ssisepecvulnerable = Csv + Isv + Dsv$$

$$Ssisepecvulnerable = 12.74 + 17.56 + 9.57$$

$$Ssisepecvulnerable = 39.87 \%$$

Tabla 4.XIII: Cuadro comparativo resumido

	CON Técnicas (%)	SIN Técnicas (%)	Mejora
Confidencialidad (30%)	24.78	12.74	12.04
Integridad (40%)	31.87	17.56	14.31
Disponibilidad (30%)	24.28	9.57	14.71
Total (100%)	80.93	39.87	41.06

Elaborado por: Paúl Paguay

Los Indicadores tanto en el Sistema **SIN** Técnicas así como en el sistema **CON** Técnicas cambian como se puede apreciar en la figura 4.6.

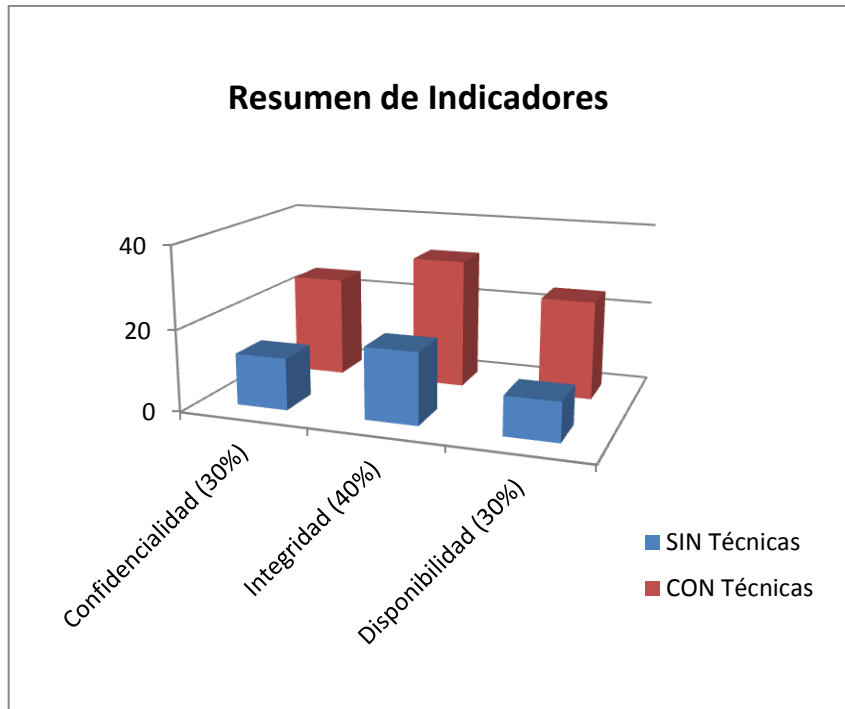


Figura 4.6: Resumen de Indicadores

4.5 Prueba de Hipótesis

Luego de la tabulación de los datos se encontraron los siguientes resultados:

- Seguridad SISEPEC (S_s)= 80.93%
- Seguridad SISEPEC VULNERABLE (S_{sv})= 39.87%

Los mismos que son representados en la Figura 4.7.

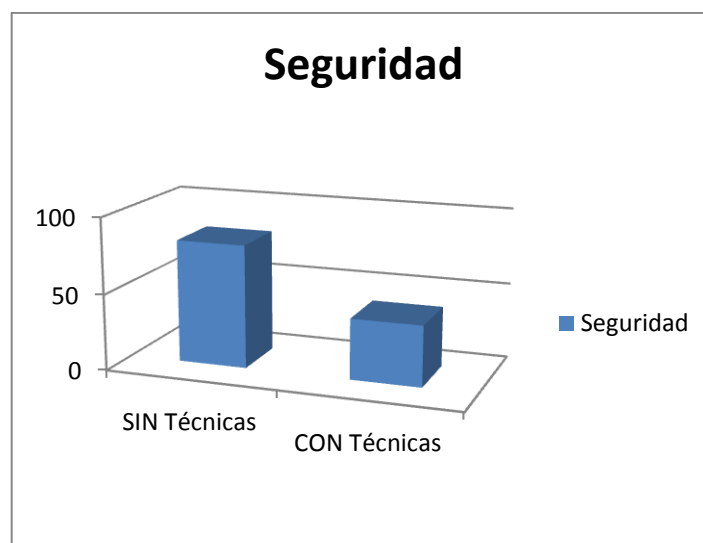


Figura 4.7: Comparación de la Seguridad

Mejora de Seguridad=80.93% - 39.87%

Mejora de Seguridad= **41.06%**

Mediante el análisis de las seguridades en ambos sistemas, se ha verificado que la implementación de técnicas de aseguramiento, mejoran hasta en un 41.06% la seguridad, respecto de una aplicación que no aplique ninguna de estas estrategias, con lo cual se comprueba la hipótesis planteada.

4.6 Propuesta de Guía

Luego de la implementación de varias técnicas de aseguramiento se ha generado la siguiente guía para mejorar la seguridad de una aplicación web desarrollada en Java – JSP.

GUÍA DE ASEGURAMIENTO DE APLICACIONES WEB CREADAS EN JAVA JSP

El presente documento es una guía para el aseguramiento de aplicaciones Web creadas en Java-JSP y que se encuentran en producción, mediante la cual se seguirán una cantidad de pasos hasta minimizar la inseguridad de la aplicación.

Estructura del Manual

En la Figura 4.8 se muestra la estructura y pasos a seguir del manual propuesto.

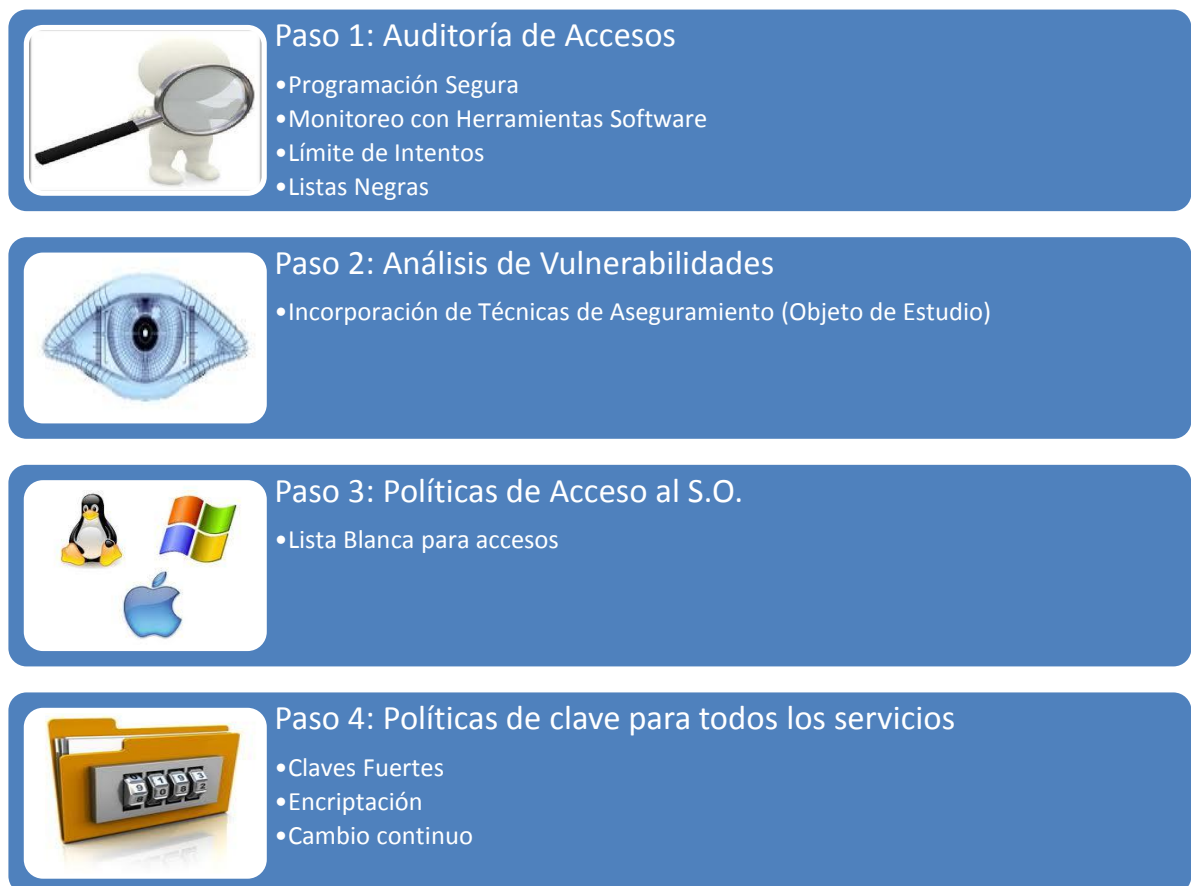


Figura 4.8: Estructura del manual

A continuación se detalla cada uno de los módulos del manual para el aseguramiento de aplicaciones:

I. Paso 1: Establecer un módulo de Auditoría de accesos

1. Programación Segura

- Indudablemente, para contar con una aplicación segura, se debe contar con una adecuada consideración en todas las etapas del ciclo de vida del proyecto del desarrollo del sistema, empezando desde el análisis hasta la implementación.

2. Monitoreo con herramientas software

- Registro de todos los accesos con el detalle que se ha introducido en los campos de información sensible como claves, fechas, tipos (Exitoso, Fallido), usuarios, etc.

3. Límite de intentos

- Por IP

Bloquear IPs que generen demasiadas peticiones o a su vez, tenga registrado en la parte de monitoreo de peticiones muchos accesos fallidos.

Este límite debe ser especificado ejemplo hasta 40 intentos fallidos para una determinada ip.

Luego de que varios accesos fallidos sean generados y aparezca un acceso exitoso, este acceso deberá encerrar nuevamente el contador de accesos fallidos.

- Por usuario

Si se especifica en algún campo el valor de un usuario, el mismo que generalmente es público, ejemplo, apellidos, cédulas, códigos, y los accesos fallidos han sido repetitivamente generados por este mismo usuario es posible que se desee hackear o hurtar la clave de este usuario, por lo cual se recomienda que este límite debe ser considerablemente más pequeño que el anterior límite, ejemplo 5 o 3 dependiendo de la criticidad del sistema.

- Cuidados a tomar con proxys

Al establecer estos límites de intentos, es importante tomar en cuenta a los usuarios internos que se conectan mediante próxys ya que la ip es la misma, por lo cual es necesario hacer un estudio de la tasa de peticiones y multiplicarla por un coeficiente, recomendación del autor.

$$100 * 0,1 = 10$$

Donde 0,1 es la probabilidad que el 10% de usuarios se equivoquen la contraseña y 100 es la tasa de solicitudes de autenticación.

4. Lista negra

- Si luego de especificar los límites de accesos fallidos, existen usuarios o ips que superen estos límites, deben ser ingresados en una lista negra para el bloqueo en toda la aplicación, así como notificaciones al administrador, para el análisis con el área de redes y bloquear a nivel de hardware o ubicación del atacante mediante monitoreo interno en caso de ser el caso.

Como se puede observar en la Figura 4.9 se ha registrado un total de 700 accesos fallidos, de los cuales a 7 se han bloqueado.

```
sisepec=# select count(*) from epec.blacklist;
count
-----
      700
(1 row)

sisepec=# select count(*) from epec.blacklist where activo=true;
count
-----
       7
(1 row)

sisepec=#
```

Figura 4.9: Bloqueos de Lista Negra

II. Paso 2: Análisis de vulnerabilidades de Código Fuente

1. Técnica contra el ataque Cross-Site-Scripting (XSS)

- Filtración de caracteres especiales y “códigos” para evitar el enmascaramiento de código malicioso.

A continuación una recomendación y ejemplo de caracteres a filtrar:

```
String[] caracteres = {"!", "#", "$", "%", "&", "/", "(", ")", "|", "\\", "{", "}", " ", "<", ">", "_", "^", ":", "?", " ", "~", "*", "[", "]"};
```

- Utilización de librería ESAPI para la filtración de código malicioso

2. Técnica contra el ataque SQL injection.

La típica vulnerabilidad cuando se conoce cual motor de base de datos se utiliza en el sistema, es la de incrustar código malicioso en el campo de las contraseñas de los usuarios, como se muestra en el siguiente ejemplo:

```
1' or 1=1 and "strCedula"='060272477-5
```

En la que con (1') se cierra la validación del password y se aumenta otra cláusula, que siempre va a cumplir (1=1) y por último despliego la información del usuario que deseo hurtar la clave ("strCedula"='060272477-5); al final tendremos ejecutándose la siguiente instrucción:

```
SELECT "strCedula","strNombre","strCodTipo","strCodEstado",  
       "dtFechaUltIng","dtFechaUltCC","strPass","strMail"  
FROM epec."Usuario" where "strCedula"='0602724775' and "strPass"='1'or  
1=1 and "strCedula"='0602724775';
```

Lo cual permitirá saltar la validación de la contraseña.

Utilización de Prepared Statement en lugar de Statements al momento de acceder a la base de datos de la aplicación.

3. Técnica contra el ataque Buffer Overflow

Este ataque puede ser bloqueado con el módulo de auditoría de accesos, para controlar el nivel de peticiones de la página “sensible” o de “datos sensibles”, para el bloqueo de ips o usuarios, la cual creará una lista negra para el control.

Filtrar el tamaño de la variable, de tal manera que no se sature la memoria principal que ocupa la aplicación, es importante almacenar el origen del problema, para ingresar en lista negra.

4. Técnica contra el ataque Directorio Transversal

Nuevamente filtro de caracteres, para evitar la posibilidad de cambiar de directorio mediante la URL y permitir accesos a directorios superiores a la aplicación o privados, eliminar la navegación en directorios.

Ejemplo en el servidor apache agregando la siguiente línea:

```
-Indexes
```

No utilizar variables directamente para el direccionamiento de páginas.

5. Técnica contra el ataque de Intercepción Criptográfica

La mejor manera de resguardar datos sensibles de un sitio web, cualquiera que este sea, es mediante certificados digitales, sin embargo muchas instituciones no cuentan con el presupuesto suficiente para costear las tarifas anuales por el servicio, por lo cual tomo un rol muy importante el módulo de auditoría de accesos, lista negra y encriptación de claves en almacenamiento y transporte.

Se puede tomar como medida en la página que contiene información sensible, la encriptación previa mediante javascript para el transporte de información encriptada.

6. Técnica contra el ataque de Día 0

Un problema frecuente es la utilización de nuevas tecnologías, debido a las innovaciones o incorporaciones adicionales a versiones anteriores, sin embargo se cae en el riesgo de que también aparezcan con vulnerabilidades tanto en el Framework como en su utilización, por lo cual es importante esperar a las versiones estables antes de optar por una migración de plataforma o versión de algún software.

III. Paso 3: Políticas de acceso al sistema operativo del servidor

1. Lista blanca para accesos

Si bien es cierto, las aplicaciones web o cualquier sistema software lo podemos blindar con todas las seguridades necesarias, sin embargo no podemos olvidar que este sistema correrá o funcionará sobre otro que es el sistema operativo, es muy común, por la movilidad de los usuarios, permitir el acceso SSH, Escritorio Remoto o Telnet(poco común por seguridad).

Pero la apertura de estos servicios y mediante estos de los puertos permite abrir vulnerabilidades que afecten a nuestra aplicación.

Por lo cual se debe crear una lista blanca en el acceso, por ejemplo en el sistema Linux, tenemos la posibilidad en el archivo hosts.allow

Así también el acceso no debe ser dada directamente a la red externa sino solo a una máquina que actúe a manera de puerta de acceso a todos los servidores con aplicaciones sensibles, con esto duplicaremos la dificultad, puesto que se deberá vulnerar primero el acceso a esta máquina denominada “puerta de acceso” y posteriormente vulnerar el servidor con los servicios informáticos.

IV. Paso 4: Políticas de claves de “todos” los servicios

1. Claves fuertes

- Obligar al usuario a ingresar claves de mínimo 8 caracteres, que incluyan Mayúsculas, minúsculas, números y signos.

2. Encriptación

- Las claves no solo deben ser almacenadas de manera encriptada en la base de datos, sino además transportarlas de manera codificada, recomendable aplicación de varios algoritmos y varias veces, así como la combinación con una clave pública. Utilización de javascripts para el ocultamiento de llaves o información sensible, en el transporte de información.

3. Cambio continuo

- Obligatoriedad de cambio de clave, cuando el administrador del sistema lo crea conveniente (depende del nivel de criticidad)

CONCLUSIONES

- El sistema que se implementó con las técnicas de aseguramiento mejoró su seguridad en cuanto a vulnerabilidades de código fuente hasta en un 41.06% respecto de una que no implementó ninguna técnica.
- Luego del estudio se puede visualizar que la seguridad se incrementó en los 3 indicadores CID, la Confidencialidad (12.04%), Integridad (14.31%) y Disponibilidad (14.71%).
- La aplicación Web CON técnicas y SIN técnicas, realizan correctamente los procesos de la Escuela de Postgrado y Educación Continua de la ESPOCH, sin embargo requerimientos “no funcionales” como la Seguridad, constituyen un requerimiento primordial.
- Es importante saber que los sistemas informáticos automatizan los procesos y la eficiencia de los mismos, sin embargo abre muchas posibilidades de un ataque, el mismo que puede generar problemas a la Imagen de la Institución o situaciones más críticas como pérdida, cambio o robo de información.
- Existen “falsos positivos” que es una detección de una vulnerabilidad de un sistema sin que éste sea realmente una vulnerabilidad. Para confirmar o descartar su validez es necesario probar si efectivamente genera alguna consecuencia dicha vulnerabilidad, por ejemplo en las herramientas de escaneo proporcionan el enlace de ataque, con lo cual facilita la prueba por parte del desarrollador de la aplicación.

- Las herramientas de escaneo generan resultados de vulnerabilidades, los mismos que deben ser analizados uno por uno para de esa manera asegurar lo mejor posible la aplicación.
- Todos los sistemas informáticos poseen componentes para su funcionamiento como base de datos, protocolos de comunicación, lógica de negocio, interfaces, los cuales se puede asegurar, sin embargo otro componente del sistema es el Usuario final el cuál también puede ser vulnerado y por supuesto a este último eslabón no se lo puede automatizar por lo cual la capacitación constituye una etapa importante en el fin de asegurar en lo posible los sistemas.

RECOMENDACIONES

- ✓ La seguridad informática de las aplicaciones hoy son destinados a ser considerado por los administradores de contratos o Responsables de las áreas de las Tecnologías de la Información de cualquier Institución.
- ✓ Las técnicas de aseguramiento deben ser consideradas e implementadas íntegramente y la actualización constante de nuevas vulnerabilidades, es un trabajo constante.
- ✓ Realizar también implementación de sistemas Biométricos, si no se cuenta con presupuesto para hardware, pues se lo puede realizar con varia información registrada en una base de datos de usuarios.
- ✓ Nunca olvidar la encriptación y cifrado de información sensible.
- ✓ Citando una frase del libro del Arte de la Guerra, Sun Tu, donde dice que: “No debemos depender de la posibilidad de que el enemigo no nos ataque, sino el hecho de que logramos que nuestra posición sea inatacable”^[12].

RESUMEN

El presente trabajo tiene como objetivo establecer Propuestas de técnicas de aseguramiento de aplicaciones Web desarrolladas en JAVA, específicamente con la tecnología JSP, aplicado al sitio Web de la Escuela de Postgrado y Educación Continua – ESPOCH.

Las técnicas establecidas se aplicaron a un sistema en producción como es el Sistema Académico Administrativo de la EPEC – SISEPEC, con lo cual resultaron 2 escenarios Web. Uno con técnicas implementadas y el otro sin ninguna técnica. Se analizó con igual servicio, como es el de la automatización de los procesos del departamento. En cada escenario, se evaluaron las vulnerabilidades por categoría, como es Confidencialidad, Integridad y Disponibilidad (CID), que son los indicadores de la Seguridad de un Sistema Web.

Para el análisis se utilizó herramientas de escaneo de vulnerabilidades de software libre que son usados por comunidades de Hackers a nivel mundial como son las suites Backtrack y OSSIM.

Luego de la recolección y tabulación de datos se pudo comprobar que la aplicación con las técnicas implementadas disminuye la cantidad de vulnerabilidades de todas las categorías del CID, con lo cual se mejoró la seguridad del sitio Web en un 41.06%, respecto a no implementar las técnicas.

Se concluye que existió una mejora de la seguridad del sistema Web. Luego se creó una guía para el aseguramiento de vulnerabilidades de código fuente con el fin de utilizar en futuras implementaciones Web JAVA-JSP.

Se recomienda que las técnicas de aseguramiento deban ser consideradas e implementadas íntegramente así como la actualización constante de nuevas vulnerabilidades, son necesarias para este tipo de sistemas Web.

SUMMARY

This work aims to establish technical proposals to assure Web applications developed in JAVA mainly with JSP technology applied to the Web site of the Graduate and Continuing Education School - ESPOCH.

The established techniques were applied to a production system such as the System Administrative Academic EPEC - SISEPEC, finding 2 stages Web to work on. One was implemented with such techniques and other without any technique. It was analyzed with the same service, the automation of the processes of the department. In each scenario, it was evaluated the vulnerabilities by category, such as Confidentiality, Integrity and Availability (CID), which are indicators of the safety of a Web system.

To analyze the vulnerability, scanning tools based on free software were used. These software are the same used by hacker communities worldwide such as the suites Backtrack and OSSIM.

After collection and tabulation of data it was found that the techniques implemented application decreases the number of vulnerabilities in all categories of the CID, which it improved the Web site security by 41.06% compared to not implementing techniques.

As a conclusion, the system security Web was improved. Afterwards a security guide for securing source code of future implementations JAVA-JSP on the Web was done.

It is recommended that assurance techniques must be considered and implemented in full. Also the constant updating of new vulnerabilities is necessary for this type of Web systems.

GLOSARIO DE TÉRMINOS

D

DBMS Sistema de Gestión de Base de Datos

DER Diagrama Entidad Relación

DPT Directory Path Transversal, Tipo de ataque a sitios Web

E

ESPOCH Escuela Superior Politécnica de Chimborazo

EPEC Escuela de Postgrado y Educación Continua, Departamento cuya misión es Formar talento humano a través del postgrado y la educación continua, que permita el crecimiento institucional, de la colectividad y del país, a través de una respuesta oportuna y de calidad a las demandas actuales y futuras de la sociedad, recurriendo para ello al desenvolvimiento de la ciencia, la tecnología, el arte y la cultura.

ESAPI Enterprise Security API, Conjunto de librerías para el aseguramiento de aplicaciones empresariales publicada por OWASP.

F

FALSO POSITIVO Error de una herramienta de escaneo, en la que indica que existe una vulnerabilidad cuando en realidad no la hay.

H

HTML HyperText Markup Language («lenguaje de marcado de hipertexto»)

J

JAVA Java es un lenguaje de programación y la primera plataforma informática creada por Sun Microsystems en 1995. Es la tecnología subyacente que

permite el uso de programas punteros, como herramientas, juegos y aplicaciones de negocios. Java se ejecuta en más de 850 millones de ordenadores personales de todo el mundo y en miles de millones de dispositivos, como dispositivos móviles y aparatos de televisión.

G

GlassFish GlassFish es un servidor de aplicaciones de software libre desarrollado por Sun Microsystems, compañía adquirida por Oracle Corporation, que implementa las tecnologías definidas en la plataforma Java EE y permite ejecutar aplicaciones que siguen esta especificación. La versión comercial es denominada Oracle GlassFish Enterprise Server (antes SunGlassFish Enterprise Server) . Es gratuito y de código libre, se distribuye bajo un licenciamiento dual a través de la licencia CDDL y la GNU GPL.

M

Metadata Datos acerca de los datos

N

Negocio Término genérico que se utiliza en el ámbito tecnológico para referirse a una Institución, Empresa u Organización

O

Objetivo Estado deseado a alcanzar. Es un enunciado breve que define los resultados esperados de la institución y establece las bases para la medición de los logros obtenidos.

OWASP: Open Web Application Security Project, proyecto de código abierto dedicado a determinar y combatir las vulnerabilidades que hacen que el software sea inseguro.

P

Password Clave

Programa Un Programa es todo conjunto de proyectos que guardan un lineamiento base común a todos ellos, y cuyos objetivos están alineados con un objetivo estratégico institucional.

Proyecto Es un esfuerzo temporal emprendido para crear un producto o servicio único. Consiste en un conjunto de actividades que se encuentran interrelacionadas y coordinadas; la razón de un proyecto es alcanzar objetivos específicos dentro de los límites que imponen un presupuesto, un alcance y un lapso de tiempo previamente definido.

S

SI Sistema de Información

STRUCTURE QUERY LANGUAGE

Es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas.

T

TIC Tecnologías de la información y Comunicación

U

UML Unified Modeling Language

V

Variable Es una característica que al ser medida es susceptible de adoptar diferentes valores en el transcurso del tiempo.

W

Wireshark Sistema de monitoreo de red, que permite capturar paquetes de todo tipo que viajan por la red.

BIBLIOGRAFÍA

1. **EGEBRETSON, P.**, The Basics of Hacking and Penetration Testing., Washington-Estados Unidos., 2011 Pp., 111.

EBOOKS

1. **AYENSON, M**, “Cloud Vulnerability Assessment”, Abril 2012, MQP CDR#GXS1002.,
https://www.wpi.edu/Pubs/E-project/Available/E-project-042412-130544/unrestricted/Vulnerability_Assessment_Within_The_Cloud_MA_AG.pdf., 2012-10-20
2. **KIRAN M**, Security Code Review- Identifying Web Vulnerabilities.
http://www.infosecwriters.com/text_resources/pdf/Code_Review_KMaraju.pdf.,
2012-10-20.
3. **LINCOLN, R.** “BackTrack 4 “Assuring Security by Penetration Testing”, 1a. Ed., UK, 2011.,
<http://books.google.com.ec/books?id=SodvK4NMBgwC&printsec=frontcover&hl=es#v=onepage&q&f=false>.,
2012-10-20.

BIBLIOGRAFÍA DE INTERNET

1. **ACTUALIDAD VULNERABILIDADES.**,
http://cert.inteco.es/vulnSearch/Actualidad&Actualidad_Vunerabilidades/?postAction=getVulnsHom.,
2012-10-20.
2. **ACTUALIDAD VULNERABILIDADES.**,
http://www.inteco.es/Formacion/Amenazas/Vulnerabilidades/Tipos_Vulnerabilidades/.,
2012-10-20.

3. **ALGUNAS RAZONES PARA LA REPRESIÓN PENAL AUTÓNOMA DEL INTRUSISMO INFORMÁTICO.,**
<http://foros.uexternado.edu.co/ecoinstitucional/index.php/derpen/article/view/File/1026/970.>,
2012-10-20.
4. **AMENAZAS LÓGICAS - TIPOS DE ATAQUES.,**
<http://www.segu-info.com.ar/ataques/ataques.htm>.,
2011-06-06.
5. **ATAQUES INFORMÁTICOS.,**
https://www.evilmfingers.com/publications/white_AR/01_Atques_informaticos.pdf., 2011-06-07.
6. **BURP.,**
<http://www.portswigger.net/burp/>.,
2013-01-23.
7. **CATEGORY:OWASP ENTERPRISE SECURITY API.,**
https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API.,
2013-01-21.
8. **CLOUD VULNERABILITY ASSESSMENT.,**
https://www.wpi.edu/Pubs/E-project/Available/E-project-042412-130544/unrestricted/Vulnerability_Assessment_Within_The_Cloud_MA_AG.pdf.,
2012-10-20.
9. **COMMAND INJECTION IN JAVA.,**
https://www.owasp.org/index.php/Command_injection_in_Java.,
2013-01-20.
10. **CÓMO CONVERTIRSE EN HACKER. TRADUCIDO POR MIQUEL VIDAL.,**
<http://hsrhr.byethost14.com/documentos/Como%20convertirse%20en%20hacker.pdf> .,
2013-01-23.
11. **CONTRA EL CISEN, UN ATAQUE DE HACKERS CADA 10 MINUTOS.,**
<http://impreso.milenio.com/node/8933614>.,
2011-06-07.
12. **EL ARTE DE LA GUERRA.,**
<http://www.slideshare.net/moplin/gestin-de-vulnerabilidades>.,

2013-01-20.

13. **ESTADÍSTICAS DE INCIDENTES REPORTADOS.**,
[http://www.cert.br/stats/incidentes/.](http://www.cert.br/stats/incidentes/),
2011-06-07.
14. **GESTIÓN DE LA SEGURIDAD CON OSSIM.**,
[www.itdeusto.com.](http://www.itdeusto.com/),
2012-10-20.
15. **GESTIÓN DE RIESGOS DE LOS SISTEMAS DE INFORMACIÓN.**,
[http://www.mailxmail.com/curso-gestion-riesgos-sistemas-informacion/identificacion-vulnerabilidades-impactos.](http://www.mailxmail.com/curso-gestion-riesgos-sistemas-informacion/identificacion-vulnerabilidades-impactos/),
2011-06-07.
16. **INSEGURIDAD INFORMÁTICA: UN CONCEPTO DUAL EN SEGURIDAD INFORMÁTICA.**,
[http://ingenieriasimple.com/papers/inseg-inf.pdf.](http://ingenieriasimple.com/papers/inseg-inf.pdf),
2012-10-20.
17. **INTRODUCTION TO NESSUS.**,
[http://www.bandwidthco.com/sf_whitepapers/penetration/Introduction%20to%20Nessus.pdf.](http://www.bandwidthco.com/sf_whitepapers/penetration/Introduction%20to%20Nessus.pdf),
2013-01-23.
18. **JAVA EE: SEGURIDAD EN APLICACIONES WEB (II). EVITANDO INYECCIÓN SQL Y XSS CON ESAPI.**,
[http://blog.jdiezfoto.es/informatica/java-ee-seguridad-en-aplicaciones-web-ii-evitando-inyeccion-sql-y-xss-con-esapi/.](http://blog.jdiezfoto.es/informatica/java-ee-seguridad-en-aplicaciones-web-ii-evitando-inyeccion-sql-y-xss-con-esapi/),
2013-01-21.
19. **LAS AMENAZAS INFORMÁTICAS: PELIGRO LATENTE PARA LAS ORGANIZACIONES ACTUALES.**,
[http://revistas.uis.edu.co/index.php/revistagti/article/view/1259/1656.](http://revistas.uis.edu.co/index.php/revistagti/article/view/1259/1656),
2011-06-06.
20. **MANUAL DE SEGURIDAD EN REDES. ARGENTINA.**,
[http://www.arcert.gov.ar/webs/manual/manual_de_seguridad.pdf.](http://www.arcert.gov.ar/webs/manual/manual_de_seguridad.pdf),
2013-01-23
21. **MEET SKIPFISH. OUR AUTOMATED WEB SECURITY SCANNER.**,
[http://googleonlinesecurity.blogspot.com/2010/03/meet-skipfish-our-automated-web.html.](http://googleonlinesecurity.blogspot.com/2010/03/meet-skipfish-our-automated-web.html),
2012-10-20.

22. **NIKTO2.**,
<http://cirt.net/nikto2.>,
2013-01-23.
23. **ORIGEN DE LOS PROBLEMAS DE SEGURIDAD INFORMÁTICA.**,
[http%3A%2F%2Fredimido.glo.org.mx%2Fpresentaciones_y_textos%2Fpresentacion_helios_mier%2FOrigen de los problemas de Seguridad Informatica_nuevo.ppt.](http%3A%2F%2Fredimido.glo.org.mx%2Fpresentaciones_y_textos%2Fpresentacion_helios_mier%2FOrigen_de_los_problemas_de_Seguridad_Informatica_nuevo.ppt.),
2011-06-06.
24. **PREPARED STATEMENTS AND SQL INJECTIONS.**,
<https://isc.sans.edu/diary/Prepared+Statements+and+SQL+injections/2301.>,
2013-01-21.
25. **SEGURIDAD DE LAS TIC BAJO PROTOCOLOS TCP/IP - ANÁLISIS PARTICULAR EN ECUADOR MEDIANTE ESCANEAMIENTO DE PUERTOS.**,
<http://biblioteca.cenace.org.ec/jspui/handle/123456789/86.>,
2011-06-07.
26. **SEGURIDAD INFORMÁTICA, ¿QUÉ, PORQUÉ Y PARA QUÉ?.**,
<http://www.inegi.gob.mx/inegi/contenidos/espanol/ciberhabitat/museo/cerquita/redes/seguridad/intro.htm.>,
2011-06-07.
27. **TOOLS: GREDEL SCANNER A NEW WEB APPLICATION SECURITY SCANNER.**,
<http://www.cgisecurity.com/2008/08/tools-grendel-s.html.>,
2012-10-20.
28. **TRATAMIENTO AUTOMATIZADO Y PROTECCIÓN DE DATOS EN LA ADMINISTRACIÓN PÚBLICA.**,
<http://www.madrid.org/cs/Satellite.>,
2011-06-06.
29. **TUTORIAL: GEOHTTP REMOTE BUFFER OVERFLOW AND DOS.**,
<http://www.youtube.com/watch?v=vyKnk197bUM.>,
2013-01-23.
30. **VULNERABILIDAD.**,
<https://www.owasp.org/index.php/Category:Vulnerability.>,
2011-06-07.
31. **VULNERABILIDADES EN APLICACIONES WEB.**,
<http://www.wisedatasecurity.com/vulnerabilidades-aplicaciones-web.html.>,
2011-06-07.

32. WEB APPLICATION SECURITY - BUFFER OVERFLOWS:ARE YOU REALLY AT RISK?.,

http://www.infosecwriters.com/text_resources/pdf/Buffer_TOlzak.pdf,
2013-01-23.

33. WEB APPLICATION VULNERABILITY SCANNER / SECURITY AUDITOR.,

<http://www.ict-romulus.eu/web/wapiti>.,
20-October-2012.

ANEXOS

