Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

# DEFINICIÓN DE UNA METODOLOGÍA PARA EVALUAR EL GRADO DE SEGURIDAD DE UNA APLICACIÓN WEB

**Manuel Alberto Minera Baldizón**
Asesorado por la Inga. Blanca Cecilia Castillo Marroquín

Guatemala, noviembre de 2011

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA


FACULTAD DE INGENIERÍA

# DEFINICIÓN DE UNA METODOLOGÍA PARA EVALUAR EL GRADO DE SEGURIDAD DE UNA APLICACIÓN WEB

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

**MANUEL ALBERTO MINERA BALDIZÓN**
ASESORADO POR LA INGA. BLANCA CECILIA CASTILLO MARROQUÍN

AL CONFERÍRSELE EL TÍTULO DE

**INGENIERO EN CIENCIAS Y SISTEMAS**

GUATEMALA, NOVIEMBRE DE 2011

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA


## NÓMINA DE JUNTA DIRECTIVA

| | |
|---|---|
| DECANO | Ing. Murphy Olympo Paiz Recinos |
| VOCAL I | Ing. Alfredo Enrique Beber Aceituno |
| VOCAL II | Ing. Pedro Antonio Aguilar Polanco |
| VOCAL III | Ing. Miguel Ángel Dávila Calderón |
| VOCAL IV | Br. Juan Carlos Molina Jiménez |
| VOCAL V | Br. Mario Maldonado Muralles |
| SECRETARIO | Ing. Hugo Humberto Rivera Pérez |

## TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

| | |
|---|---|
| DECANO | Ing. Herbert Rene Miranda Barrios |
| EXAMINADOR | Ing. Calixto Raúl Monzón Pérez |
| EXAMINADOR | Ing. Jorge Leonel Villeda Lemus |
| EXAMINADOR | Ing. Carlos Martin Ruíz Blau |
| SECRETARIA | Inga. Gilda Marina Castellanos de Illescas |

# HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

## DEFINICIÓN DE UNA METODOLOGÍA PARA EVALUAR EL GRADO DE SEGURIDAD DE UNA APLICACIÓN WEB

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha noviembre de 2010.

Manuel Alberto Minera Baldizón

Guatemala, Julio de 2011

Ingeniero
Carlos Alfredo Azurdia Morales
Coordinador de Privados y Revisión de Tesis
Escuela de Ciencias y Sistemas

Estimado Ingeniero:

Por medio de la presente, me permito informarle que he asesorado el trabajo de graduación titulado: **DEFINICIÓN DE UNA METODOLOGÍA PARA EVALUAR EL GRADO DE SEGURIDAD DE UNA APLICACIÓN WEB**, elaborado por el estudiante Manuel Alberto Minera Baldizón, a mi juicio el mismo cumple con los objetivos propuestos para su desarrollo.

Agradeciéndole de antemano la atención que le preste a la presente, me suscribo de usted,

Atentamente,

*Blanca Cecilia Castillo Marroquìn*
*Inga. en Ciencias y Sistemas*
*Colegiado No. 8401*

Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 21 de Septiembre de 2011

Ingeniero
**Marlon Antonio Pérez Turk**
**Director de la Escuela de Ingeniería**
**En Ciencias y Sistemas**

Respetable Ingeniero Pérez:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **MANUEL ALBERTO MINERA BALDIZÓN** carné **90-12801,** titulado: **"DEFINICIÓN DE UNA METODOLOGÍA PARA EVALUAR EL GRADO DE SEGURIDAD DE UNA APLICACIÓN WEB",** y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

**Ing. Carlos Alfredo Azurdia**
Coordinador de Privados
y Revisión de Trabajos de Graduación

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
INGENIERIA EN CIENCIAS Y SISTEMAS
COORDINACION TESIS

ESCUELA DE CIENCIAS Y SISTEMAS

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÌA
ESCUELA DE CIENCIAS Y SISTEMAS
TEL: 24767644

*El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del asesor con el visto bueno del revisor y del Licenciado en Letras, de trabajo de graduación titulado* ***"DEFINICIÓN DE UNA METODOLOGÍA PARA EVALUAR EL GRADO DE SEGURIDAD DE UNA APLICACIÓN WEB"****, presentado por el estudiante MANUEL ALBERTO MINERA BALDIZÓN, aprueba el presente trabajo y solicita la autorización del mismo.*

***"ID Y ENSEÑAD A TODOS"***

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCIÓN DE INGENIERÍA EN CIENCIAS Y SISTEMAS

*Ing. Marlon Antonio Pérez Turk*
*Director, Escuela de Ingeniería Ciencias y Sistemas*

*Guatemala, 21 de noviembre 2011*

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

DTG. 521.2011

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al trabajo de graduación titulado: **DEFINICIÓN DE UNA METODOLOGÍA PARA EVALUAR EL GRADO DE SEGURIDAD DE UNA APLICACIÓN WEB,** presentado por el estudiante universitario **Manuel Alberto Minera Baldizón,** autoriza la impresión del mismo.

IMPRÍMASE:

Ing. Murphy Olympo Paiz Recinos
Decano

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

Guatemala, 21 de noviembre de 2011.

/gdech

## ACTO QUE DEDICO A:

| | |
|---|---|
| **Dios** | Por darme la gran bendición de haber cumplido esta meta. |
| **Mis padres** | Manuel de Jesús Minera Bolaños y María Isabel Baldizón González, por el apoyo y cariño brindado. |
| **Mi esposa** | Patricia Godínez Vargas de Minera, por su amor y apoyo incondicional. |
| **Mis hijos** | José Manuel y Paula María Minera Godínez, por ser mi principal fuente de inspiración y motivación. |
| **Mis hermanos** | Julio César, José Rafael, María Isabel y Carlos Giovani, por su ejemplo. |
| **Mis amigos** | Por tanto apoyo, consejos y motivaciones para lograr esta meta. |

## AGRADECIMIENTOS A:

| | |
|---|---|
| **Dios** | Por cuidarme y guiarme en el camino correcto. |
| **María Santísima** | Por ser mi recurso ordinario y mi guía a Dios. |
| **Mi padre** | Por su apoyo y consejo. |
| **Mi madre** | Por todo su cariño. |
| **Mi esposa** | Por todo el apoyo y amor. |
| **Mis hermanos** | Por sus sabios consejos. |
| **Mis amigos** | Por darme su sincera amistad y sus palabras de aliento en todo momento. |

# ÍNDICE GENERAL

# ÍNDICE DE ILUSTRACIONES

## FIGURAS

## TABLAS

# GLOSARIO

**Amenaza o riesgo informático** — Cualquier tipo de acción que pretenda afectar la seguridad o funcionamiento de un sistema informático. La probabilidad o la frecuencia de que se produzcan daños dentro de un sistema informático.

**Ataque informático** — Conjunto de acciones definidas o planificadas por un ente con el fin de causar daño a un sistema informático.

**Buenas prácticas** — Conjunto coherente de acciones que ha rendido buen o incluso excelente servicio en un determinado contexto, en espera que, en contextos similares, rinda similares resultados.

***Cookie*** — Mensaje enviado por un servidor web a un cliente web, la mayoría de veces almacenado por un archivo de texto en el cliente, y enviado de nuevo al servidor web, cada vez que se consulte de nuevo una página.

**Criptografía** — Rama de la criptología que se ocupa del cifrado de mensajes. Esta se basa en que el emisor emite un mensaje en claro, que es tratado mediante un cifrador con la ayuda de una clave, para crear un texto cifrado.

**Debilidad** Es un tipo de error o *"bug"* en el *software* que, en condiciones adecuadas, podría contribuir a la introducción de las vulnerabilidades a un sistema.

**Fuzz testing** Técnica de probar aplicaciones enviando parámetros o información de entrada incorrectos, para determinar, si un sistema colapsa con este tipo de información maliciosa.

**Hacker** Persona que de manera fraudulenta ingresa a un computador o a una red de computadores y causa algún daño, motivado por ganancias, ideologías, protestas o por pasatiempo.

***Sandboxing*** Técnica muy popular, que define sistemas con sus recursos limitados y aislados, utilizados la mayoría de las veces para probar programas nuevos o de dudosa procedencia.

**Vulnerabilidad** Es una debilidad, o un grupo de debilidades que pueden ser utilizadas en contra de un sistema para que este proporcione información confidencial, genere información errónea o no deseada, o simplemente su servicio sea interrumpido.

# RESUMEN

Debido al incremento en el uso de internet, cada vez más compañías permiten a sus socios y proveedores acceder a sus sistemas de información. Por lo tanto, es fundamental saber qué recursos de la compañía necesitan protección para mantener la integridad de la misma.

Si los sistemas son objeto de algún ataque cibernético, los daños causados a las organizaciones suelen ser bastante considerables. Poco a poco las compañías ponen más énfasis en utilizar aplicaciones robustas para evitar ataques maliciosos.

Para que un sistema sea robusto, se debe fortalecer desde un principio, empezando por la etapa de análisis y diseño, con la definición de un modelo de amenazas. Pasando luego por la etapa de desarrollo, aplicando una serie de buenas prácticas. Por último, poniendo a prueba el sistema, se realizan pruebas de penetración o de robustez.

Dentro del ambiente guatemalteco, muchos de los temas desarrollados en este trabajo son desconocidos, pero al menos ya cuentan con alguna política de seguridad, aunque no se encuentre definida de manera formal.

# OBJETIVOS

## General

Definir una metodología para evaluar el grado de seguridad de una aplicación WEB, basada en una serie de recomendaciones para el desarrollo de controles robustos, capaces de enfrentar cualquier vulnerabilidad.

## Específicos

1. Mejorar el nivel de resistencia a un ataque malicioso y proteger la integridad y confidencialidad de la información.

2. Presentar metodologías de última generación, haciendo un balance de sus ventajas y desventajas y dando a conocer los diferentes ataques que podrían llegar a mitigar, luego de su correcta implementación.

3. Conocer a fondo el término de *Sandboxing* y otros métodos que se utilizan en la actualidad en la etapa de diseño de sistemas.

4. Describir las principales metodologías usadas para la escritura de “código seguro”, dentro de la etapa de programación.

5. Explicar la importancia de la etapa de testeo, y aprender a realizar guiones de pruebas exhaustivas, usando las herramientas apropiadas de testeo de aplicaciones.

6. Determinar el grado de madurez alcanzado por las empresas guatemaltecas desarrolladoras de software, en temas relacionados a la seguridad informática.

# INTRODUCCIÓN

La seguridad informática ha alcanzado un alto grado de madurez debido al uso más constante del internet dentro del diario vivir. Hoy en día es más común que organizaciones basen muchos de sus procesos en sistemas informáticos.

Para determinar qué tan vulnerable puede ser una aplicación contra un ataque cibernético, se debe evaluar una serie de aspectos desde probabilidades para que se desarrolle un ataque, hasta los daños que este puede causar si se llega a completar.

El presente trabajo consta de cuatro partes modulares que se describen a continuación.

En el primer capítulo, se revisa las metodologías y buenas prácticas, durante todo el ciclo de vida del *software*, partiendo desde el análisis y diseño, hasta antes de salir a producción. En el análisis se definen conceptos tales como modelos de amenazas, uso de cajas de arena o "*Sandboxing"*. Para el desarrollo se revisan prácticas en el manejo de caracteres en C y C++, los aspectos que se revisarán al procesar los datos de entrada en un sistema, y cómo realizar consultas a bases de datos sin tener que concatenar cadenas de caracteres a los querys que se utilizan.

En el capítulo dos, se describen dos métodos utilizados para el testeo de aplicaciones antes de salir a producción: el testeo de robustez o *Fuzzing* y las pruebas de penetración dirigidos por *hackers* éticos o buenos.

En el capítulo tres, se define una metodología para determinar el grado de robustez que tiene una aplicación, tomando en consideración una serie de aspectos. Según sea el impacto que tenga cada una de las amenazas, se asignará una calificación de 0 a 10 y luego se sacará una nota final promediando cada una de las amenazas; según sea la nota, así será el grado de robustez.

Por último, en el capítulo cuatro, se hace una breve revisión sobre cómo se encuentra el ambiente guatemalteco en tema de seguridad informática, analizando cuáles han sido los ataques sufridos durante los meses de noviembre 2010 hasta abril 2011, las herramientas utilizadas para detener o solventar dichos ataques, así como las diferentes políticas y medidas de seguridad que maneja cada empresa.

# 1. ASPECTOS A EVALUAR EN EL DESARROLLO DE *SOFTWARE*

A lo largo de todo el ciclo del desarrollo de *software*, se deben hacer consideraciones, para ofrecer al final una aplicación lo más segura posible. A continuación se definen ciertos puntos que se deben considerar en cada una de las fases del desarrollo clásico o de cascada.

## 1.1. Análisis y diseño

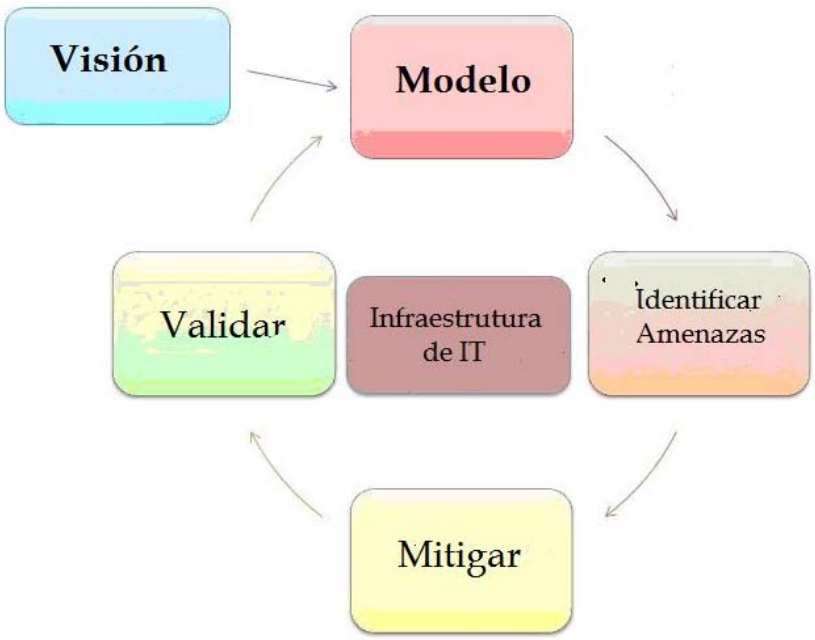
En esta fase se analizan las necesidades de los usuarios del *software* para determinar qué objetivos deben cubrir, se diseña cada módulo que tendrá la aplicación, y la relación que existirá entre cada uno. Por lo regular no se le da la importancia necesaria al análisis y diseño de sistemas, se hacen pequeños diagramas o bosquejos, pero nada formal. O lo que es peor, se empieza a desarrollar sobre la marcha y luego se analizan y definen muchos procesos o tareas relacionadas con la seguridad.

El análisis y diseño es importante para desarrollar aplicaciones seguras; detectar todas las posibles vulnerabilidades en esta fase, puede significar un ahorro bastante significativo en tiempo y recursos. Dentro de las prácticas más comunes que se tienen para detectar vulnerabilidades en la etapa de análisis y diseño están las siguientes:

### 1.1.1. Modelo de amenazas (*Threat Modeling*)

Esta técnica es una de las más comunes utilizadas en el diseño de aplicaciones; dicha técnica consiste en enumerar de manera gráfica los posibles ataques que el sistema podrá sufrir en cualquier parte del mismo. Un modelo de amenazas ayuda a evaluar la probabilidad, daño potencial y prioridad de los posibles ataques.

Figura 1. **Ciclo de vida de un modelo de amenazas**




Fuente: elaboración propia.

Para definir un modelo de amenazas, se debe realizar los siguientes pasos:

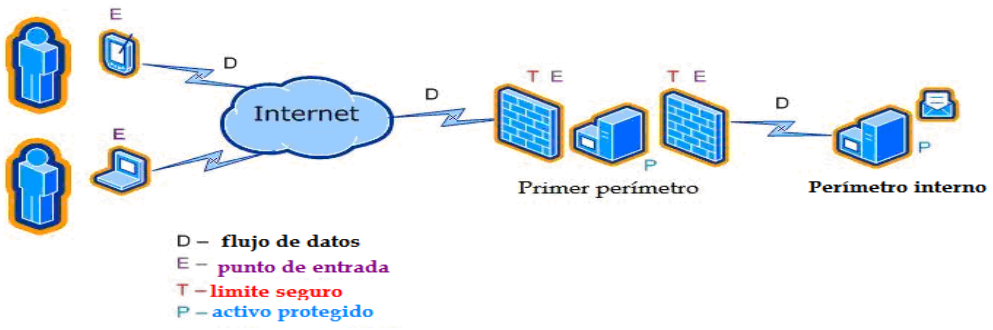
- Definir modelo de la organización:
  - Identificar los objetivos del negocio.

- Identificar cada uno de los roles de usuarios que tendrán relación con el sistema.
- Enumerar todos los datos que el sistema administrará.
- Identificar los casos de uso que el sistema resolverá.
- Describir cada uno de los posibles escenarios de uso para poder definir los flujos de datos, puntos de entrada, ataques de superficie, y recursos que deben de incluirse como parte del modelo.

- Generar un modelo de la arquitectura del sistema

  - Diagrama de componentes de la aplicación.
  - Diagramas de flujos de datos con cada uno de sus respectivos elementos.
  - Diagrama de la interacción de los diferentes componentes de la aplicación.
  - Diagrama de dependencias externas.
  - Diagrama de llamadas de los roles o usuarios a los componentes y eventualmente a los datos almacenados en la aplicación.

Figura 2. **Flujo de datos con sus elementos**




Fuente: IT Infrastructure Threat Modeling Guide Released, http://blogs.technet.com/b/seanearp/archive/2009/06/23/it-infrastructure-threat-modeling-guide-released.aspx [fecha de consulta 01/02/2011].

- Identificar todas las amenazas a la confidencialidad, disponibilidad e integridad de la información almacenada en la aplicación; para esto se puede usar el método *STRIDE* de Microsoft, que consiste en clasificar las vulnerabilidades en varios grupos:

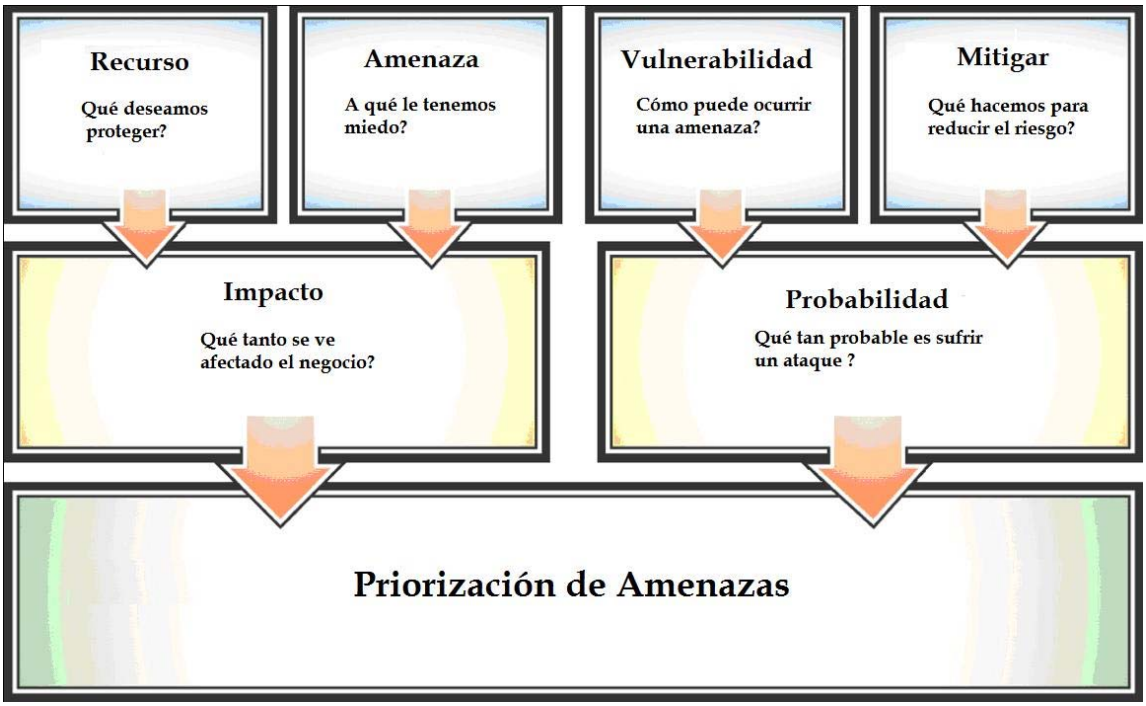
    - ***S****poofing identity* (Secuestro o robo de identidad)
    - ***T****ampering with data* (Manipulación o alteración de datos)
    - ***R****epudiation* (No dejar huellas visibles del ataque)
    - ***I****nformation disclosure* (Divulgación de información clasificada)
    - ***D****enial of service* (Denegar el servicio)
    - ***E****levation of privilege* (Tomar nuevos accesos que no le corresponden)

- Clasificar y valuar los diferentes riesgos, para luego proporcionar sus respectivas soluciones; los riesgos detectados podrán agruparse dentro de las siguientes categorías:

    - Amenazas no resueltas y que deben de ser resueltas.
    - Amenazas no resueltas pero dependientes de otro componente u otra dependencia.
    - Amenazas ya resueltas o mitigadas por el componente.
    - Amenazas ya resueltas por otro componente u otra dependencia.

- Determinar las diferentes formas de contrarrestar los riesgos basados en una prioridad que se evaluará a partir de dos aspectos importantes: el primero la probabilidad que se dé una amenaza y la segunda el impacto o daño que causará dicha amenaza.

Figura 3. **Diagrama para determinar prioridad en la resolución de amenazas**




Fuente: elaboración propia.

- Actualizar de manera continua el modelo original, incluir las nuevas amenazas descubiertas, revisar de manera constante toda la documentación generada y actualizar el avance en la resolución de las diferentes amenazas.

### 1.1.2. Uso limitado de privilegios o accesos

El concepto de ejecutar código con el mínimo de accesos posibles sigue siendo válido en nuestros días; así como era válido hace 30 años, cuando lo mencionó por primera vez Saltzer y Schroeder's en su libro titulado "La protección de la información en sistemas informáticos".

Dependiendo de las diferentes plataformas utilizadas el significado del Uso Limitado de Privilegios puede variar, pero a grandes rasgos el concepto sigue siendo el mismo: "Cada programa y cada usuario de un sistema debe de operar usando el minino conjunto de privilegios necesarios para completar una tarea".

El uso limitado de privilegios es muy importante, ya que esto reduce el posible daño, sí el sistema fuera comprometido. Una aplicación comprometida ejecutándose con todos los privilegios posibles puede causar un mayor daño, que una aplicación comprometida ejecutándose con el mínimo de privilegios.

A continuación se enumeran ciertas técnicas basadas en el uso Limitado de Privilegios:

- Uso del Ejecutar como (*Run As*), en el ambiente Windows, se puede indicar a un programa que se ejecute con un usuario diferente al usuario que esta registrado, por ejemplo, se puede ingresar un usuario administrador, para que la tarea deseada se ejecute de manera normal, esta asignación solo deberá de aplicarse después de confirmar el origen del programa que se desea ejecutar (en la Tabla 1 se muestran algunos ejemplos de su uso).

Tabla I. **Ejemplos de uso del comando *Run as***

| Actividad a realizar | Comando a ejecutarse |
|---|---|
| Ejecutar *Paint* con el usuario administrador de la máquina local | **Runas /user:*localmachinename*\administrator mspaint** |
| Ejecutar el command prompt con el usuario administrador del dominio | **Runas /user:companydomain\domainadmin cmd** |
| Ejecutar *Notepad* con un usuario administrador llamado useradm en el dominio my_domain.com | **Runas /user:useradm@my_domain.com "notepad file.txt"** |

Fuente: elaboración propia.

- Uso de los grupos de súper usuarios (*Power Users Group*), este grupo se administra o almacena en la SAM local de cada máquina, y no dentro del *Active Directory*. Se le asignan permisos intermedios; este grupo de permisos se encuentra entre los usuarios estándares y los permisos de un administrador local. Dentro de las funciones que se pueden realizar con este grupo de usuarios están:

    - Ejecutar aplicaciones obsoletas, además de aplicaciones certificadas para Windows 2000 o Windows XP Profesional.
    - Instalar programas que no modifican archivos del sistema operativo o instalación de servicios del sistema.
    - Configurar recursos del sistema tales como impresoras, fecha y hora, opciones de energía y otros recursos en panel de control.
    - Crear y gestionar cuentas de usuarios y grupos locales.
    - Detener y comenzar los servicios del sistema que no se inician de forma automática.

- Usar permisos de usuario, si se desea restringir de mejor manera los accesos se puede asignar a los usuarios “permisos de usuarios”, este grupo de privilegios se refieren a los accesos o permisos básicos dentro de un sistema.

- Usar aplicaciones específicas, para administración de privilegios o permisos; por último, si se cuenta con usuarios que necesitan realizar tareas muy especiales, lo mejor es buscar *software* especializado en administrar de manera más especifica cada una de las funciones dentro de un sistema.

### 1.1.3. Uso de cajas de arena (Sandboxing)

Se define como una caja de arena al mecanismo de seguridad para separar de manera lógica y física la ejecución de programas. Resulta común esta práctica para ejecutar código que aún no ha sido testeado, o para probar programas desarrollados por terceras personas.

La caja de arena normalmente proporciona un conjunto bien controlado de recursos para los programas “huéspedes”, como un espacio reservado en el disco y la memoria. Acceso a la red, la capacidad de inspeccionar el sistema huésped o leer en dispositivos de entrada son por lo general no permitidos o restringidos en gran medida. En este sentido, las cajas de arena son un ejemplo específico de la virtualización.

La implementación de cajas de arena puede realizarse de 2 maneras distintas: usando *Namespaces* o controles de accesos. Los espacios nombrados o “*namespaces*” trabajan bajo el principio que un “agente” no puede

manipular ningún recurso que no pueda nombrar. Los controles de accesos son una serie de procedimientos o procesos definidos de la manera siguiente "*El agente A puede manipular el recurso R a través del método M".*

Algunos ejemplos de cajas de arena son:

- *Applets*: son programas pequeños que corren en máquinas virtuales o en un contexto de una aplicación más grande. Dentro de los navegadores el uso de *Applets* son muy comunes, ya que se utilizan para ejecutar de manera segura el código embebido en sitios no confiables. Dentro de los Applets más conocidos se tiene: *Adobe Flash, Java Applets* y *Silverlight*.

- Cárceles: es un conjunto de recursos limitados administrados por el *kernel* del sistema operativo. Pueden incluir anchos de banda, accesos a disco, accesos restringidos a la red y *filesystems*. Las cárceles son muy utilizadas en aplicaciones de *hosting*.

- Máquinas virtuales: sirven para emular un computador completo, en el cual un sistema operativo convencional puede levantarse y funcionar, sobre un *hardware* específico. El sistema operativo huésped está limitado a no correr directamente sobre el *hardware* en donde está montado y solo puede accesar a los recursos de esta máquina, por medio del emulador nada más.

- *Linux secure computing mode (seccomp)* es una caja de arena desarrollada dentro del kernel del Linux; si dicho modo está activo entonces solo permite las siguientes llamadas de sistema *write(), read() exit() y sigreturn()*.

## 1.2. Etapa de desarrollo, generación de código seguro

La etapa de desarrollo es donde se genera el código fuente, haciendo uso de prototipos, así como de pruebas y ensayos para corregir errores. Dependiendo del lenguaje de programación y su versión, se crean librerías y componentes reutilizables dentro del mismo proyecto para hacer de manera más rápida la programación.

Acá se debe utilizar buenas prácticas que ya se manejan en la generación de código. Dentro de las prácticas más comunes se encuentran las siguientes:

### 1.2.1. Minimizar el uso de funciones de Carácter o de Buffer inseguras

Para las aplicaciones desarrolladas en C o C++, el rebalse del *Buffer* o la corrupción de la memoria, son de los problemas más comunes que se presentan. Se han hecho varios estudios en donde los causantes de la mayoría de problemas son las funciones de C que manejan caracteres; dentro de los grupos o familias de funciones que más problemas causan están: *strcpy, strncpy, strcat, strncat, scanf, sprint, memcpy, gets*.

Para la mayoría de sistemas las cadenas de caracteres son muy importantes, ya que representan a menudo la fuente principal de ingreso de información al sistema, de igual manera pueden representar la información de salida que genera el mismo. En el Lenguaje C se debe de tener cuidado para manejar cadenas de caracteres debido a que el tipo de dato "*String*" no existe como tal en el lenguaje C; lo que se maneja en dicho lenguaje son punteros a posiciones de memoria.

Para demostrar de mejor manera el daño que puede llegar a causar una función se va a explicar el funcionamiento de la función *strcpy*. La función *strcpy*, está definida de la siguiente manera:

```
char *strcpy(char *dest, const char *src);
```

El código básico de dicha función es el siguiente:

```
char *strcpy(char *dest, const char *src)
 {          while(NULL != *src)
            {
            *dest++ = *src++;
            }           }
```

Lo que hacen estas líneas de código es copiar carácter por carácter de la cadena **dest* a la cadena **scr*; esto se realiza hasta encontrar un carácter *null* en la cadena **scr*. El primer problema que puede ocurrir es cuando la cadena **scr* no tiene ningún carácter *null*, o peor aún que sucede si la cadena **dest* no es lo suficientemente grande para albergar el dato original.

Una primera solución es utilizar la función *strncpy*, definida de la siguiente manera:

```
char *strncpy(char *dest, const char *src, size_t n);
```

Se tiene un nuevo parámetro *"n"* que indica el número de caracteres que se estarán copiando, pero de igual manera siguen los problemas, ya que si el tamaño **scr* es más grande que "*n*", entonces solo se copiarán los primeros "*n*" caracteres y el carácter *null* del final no será incluido. O de igual manera si el **dest* es mas pequeño que los "*n*" caracteres que se necesitan copiar, se incurrirá en rebalses de memoria o corrupción de los mismos.

La solución óptima a este problema es el uso de la función:

```
size_t strlcpy(char *dst, const char *src, size_t size);
```

Los dos beneficios que se obtienen con dicha función es que no se escribe nada fuera de la porción de memoria ya asignada al **dest* y siempre se finalizará la cadena de caracteres **dest* con el campo *null*. Este grupo de funciones ya se encuentran disponibles en las librerías del tipo seguro, desarrolladas para C y C++.

Todo programador debe estar consiente de los problemas que causan dichas rutinas y además conocer las ya revisadas que excluyen dicha vulnerabilidad. A continuación algunas de las librerías que ya son seguras y su equivalencia insegura:

Tabla II. **Funciones inseguras y sus equivalentes seguras**

| Librería insegura | Librería segura |
|---|---|
| strcpy | strcpy_s |
| Strncpy | strncpy_s |
| strcat | strcat_s |
| strncat | strncat_s |
| scanf | scanf_s |
| sprintf | sprintf_s |
| memcpy | memcpy_s |
| gets | Gets_s |

Fuente: elaboración propia.

### 1.2.2. Validar las entradas y salidas del sistema para reducir las vulnerabilidades más frecuentes

Al lograr la validación de toda la información de entrada en una aplicación, y de igual manera rechazar la que no cumple con los formatos esperados, se

podrán reducir de manera considerable las vulnerabilidades dentro de las aplicaciones.

De igual manera, si se valida la información saliente de una aplicación, se pueden mitigar muchas vulnerabilidades en sistemas web. Una aplicación recibe parámetros de entrada, por medio de una secuencia de caracteres que luego se van transformando en variables de ciertos formatos y longitudes; la validación como tal de estos parámetros, se debe de realizar antes de empezar a trabajar con los datos recibidos.

Para hacer una buena validación de la información ingresada se deben considerar los siguientes aspectos:

- En toda variable de entrada deben de validarse 2 aspectos: el primero que exista y el segundo que venga en el formato requerido.
- La información debe de ser lo más simple posible, y encontrarse normalizada o en forma canónica.
- Los datos deben de validarse con los formatos especificados por la aplicación (formatos y longitudes).
- Si existen datos que se rigen a un rango de datos, este se debe de validar, (por ejemplo mayor o igual a cero, año 1950 en adelante, etc.).
- De igual manera se cuenta con datos que están dentro de una lista de valores dados; dichos elementos deben de validarse (p.e. género masculino o femenino, listado de departamentos, listado de países, etc.).

La validación de información, también debe realizarse en aplicaciones que no tienen interface con el usuario final, particularmente aquellas aplicaciones, que exponen de manera pública funciones para ser invocadas vía remota.

Las aplicaciones que usan formatos de texto o XML, pueden utilizar expresiones regulares o comparaciones de texto para validar la información de entrada.

Por último, si se tienen aplicaciones que aceptan códigos o data binaria, se debe utilizar una firma electrónica, para validar el inicio y fin de transmisión de datos; este tipo de transacciones son las más utilizadas por las empresas financieras o bancarias, para el cifrado de la información que reciben y que envían.

### 1.2.3. Minimizar el uso de concatenaciones de cadenas de caracteres para generar sentencias de SQL dinámicas

La mayoría de aplicaciones utiliza una base de datos para el almacenamiento de sus datos e información, y en la gran mayoría de estos casos para consultar o manipular la información, se utilizan sentencias de SQL; estas sentencias se arman utilizando parámetros sin validar. En pocas ocasiones se utilizan procedimientos almacenados, SSL (*Secure Socket Layer*) o encripción propia de la base de datos.

De los aspectos que se deben de tomar en cuenta para un uso adecuado de las bases de datos de manera segura, se tienen los siguientes:

- Nunca tener accesos directos a tablas específicas, pasar primero por vistas que limiten el número de campos consultados, o el número de registros que se busca.
- El usuario de la base de datos que utilice la aplicación, debe de tener accesos limitados. Tiene que crearse con su grupo de permisos para las tareas específicas que ejecutará.

De los daños que puede causar un ataque por inyección de sentencias SQL, se tiene la alteración de datos, lectura de información sensible, o la ejecución de procesos o comandos de sistema que afecten el funcionamiento de la aplicación por ejemplo: *Alter Tables, Truncate Tables, Drop Database*.

Para reducir esta vulnerabilidad, se utilizan herramientas de análisis estático, herramientas de análisis dinámico, o de manera manual.

Dentro de los ataques más comunes, se pueden destacar 2 ejemplos:
Ejemplo 1
Se tiene el siguiente Query dentro del sistema:

*select id, firstname, lastname from authors*
*where firstname = 'Var1' and lastname = 'Var2'*

Y el sistema solicita al usuario el ingreso de Var1 y Var2, el usuario podrá ingresar información como la siguiente: Var1 = basura' ing y para Var2 = Smith, entonces se tendrá generado el siguiente query:

*select id, firstname, lastname from authors*
*where firstname = 'basura' ing' and lastname = 'Smith'*

Acá no se obtiene ningún dato como resultado, únicamente un mensaje de error como el siguiente:

*Incorrect syntax near basura (Error de sintaxis cerca de basura)*

De igual manera se pueden generar varias sentencias de sql, mandando los siguientes valores a las 2 variables definidas anteriormente: Var1 = *Joe' ; delete from customers;* Var2 = espacio en blanco
Para este caso se tendrán los siguientes *querys* ejecutándose uno seguido del otro:

*select id, firstname, lastname from authors*
*where firstname = 'Joe'; delete from customers; and lastname = 'Smith'*

Esto dará como resultado la búsqueda de todas las personas que se llamen Joe y la sentencia que borrará del sistema todos los clientes de la tabla *customers*. Aunque la tercera sentencia ya no se ejecute ya que de igual manera dará el error de sintaxis.

Incorrect syntax near and (Error de sintaxis cerca de and)

Para resolver todos estos inconvenientes se puede definir un código sencillo en Java tal y como se muestra a continuación:

```
String firstname = req.getParameter("firstname");
String lastname = req.getParameter("lastname");
// Aca se ingresan las validaciones locales necesarias
String query = "SELECT id, firstname, lastname FROM authors WHERE forename = ? and
surname = ?";
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, firstname );
pstmt.setString( 2, lastname );
try
{
ResultSet results = pstmt.execute( );
}
```

Ejemplo 2

De igual manera pueden existir vulnerabilidades en código de C#; se analiza el siguiente código:

```
string userName = ctx.getAuthenticatedUserName();
string query = "SELECT * FROM items WHERE owner = '"
+ userName + "' AND itemname = '"
+ ItemName.Text + "'";
sda = new SqlDataAdapter(query, conn);
DataTable dt = new DataTable();
sda.Fill(dt);
…
```

Dicho código funcionará bien, siempre y cuando no se ingrese ninguna comilla ( ‘ ) dentro de las valores obtenidos, pero si se ingresa algún valor con comilla, se puede forzar un query para que siempre devuelva datos, aunque no se ingrese ningún valor válido.

Si se permite que ingrese el siguiente valor se estará consultando de manera completa toda la información de la tabla *items.*

"Joe' OR 'a'='a"

```
SELECT * FROM items
WHERE owner = 'name'
AND itemname = 'Joe' OR 'a'='a';
```

Con el uso del *or* y la restricción que a = a, siempre se estarán obteniendo todos los datos de la tabla *items.*

### 1.2.4. Algoritmos robustos en criptografía

En los últimos años se ha detectado una serie de debilidades en una gran cantidad de algoritmos de encriptamiento y sus respectivas implementaciones.

Debido al uso generalizado de la criptografía para asegurar autenticación, autorización, ingreso o validación de datos, y el aseguramiento de la confidencialidad e integridad de la información, las debilidades en criptografía suelen ser vulnerabilidades de alto impacto dentro de los sistemas.

Al tratarse de comunicación, especialmente de redes de comunicaciones, se debe de dar preferencia a los protocolos estandarizados que hayan sido objeto de revisión o evaluación pública, tales como *Secure Socket Layer* (SSL), *Transport Layer Security* (TLS), *IPSec*, *Kerberos*, *OASIS WS-Security*, *W3C*

*XML Encryption and XML Signature*, antes de usar algoritmos de encriptamiento de bajo nivel, o antes de desarrollar un protocolo propio.

Si va a utilizarse un algoritmo de bajo nivel, solo deben de incluirse algoritmos estandarizados ya reconocidos dentro del ámbito del desarrollo de sistemas.

Dependiendo del nivel de seguridad o por el tipo de sistema que se está desarrollando se puede solicitar que el algoritmo utilizado esté certificado por algún ente a nivel internacional, o avalado bajo alguna norma de tipo internacional como la FIPS 140-2.

Dentro de las tecnologías o algoritmos que se debe desechar por inseguros están:

- MD4
- MD5
- SHA1
- Algoritmos simétricos de criptografía (tales como el DES que solo soporta una llave de longitud de 56 bits)
- RC4 o ARC, ya que es muy difícil implementarlos de manera correcta y segura
- ECB
- Cualquier algoritmo que no haya sido estudiado y evaluado de manera correcta.

Todo el proceso de utilización e implementación de algoritmos de encriptamiento, trae inherente su complejidad, por más pequeña y sencilla que sea una implementación y que necesite de una rutina o algoritmo de

encriptamiento, puede convertirse en una gran vulnerabilidad si no es correctamente utilizada.

Para minimizar los errores comunes de implementación, las funciones de encriptamiento deben de utilizarse como un servicio y obviarse el desarrollo de funciones propias del sistema. Este tipo de reglas deben de definirse como estándares, dentro de las empresas que desarrollan aplicaciones.

Todo desarrollador debe utilizar funciones robustas para generar números Random o Aleatorios, especialmente cuando se están generando llaves para algoritmos de encriptamiento. Nunca usar las funciones *rand()* en C o C++, java.util.Random en Java o System.Random in C# or VB.NET.

Otra buena práctica para la eliminación del encriptamiento débil, es administrar de buena manera el acceso a las llaves dentro de la aplicación, proteger todo el proceso para que dichas llaves nunca queden expuestas a usuarios maliciosos.

### 1.2.5. Uso de bitácoras y contraseñas para usuarios (*Logging y Tracing*)

Al suceder algún evento relacionado con la violación o intromisión a un sistema, una de las primeras reacciones es verificar qué fue lo que pasó. Y para poder verificar lo sucedido se necesita conocer la bitácora o bitácoras del sistema. Dentro de las primeras prácticas sugeridas está el tener habilitadas las diferentes bitácoras del sistema operativo, y de manera complementaria, implementar bitácoras propias de la aplicación.

Los desarrolladores deben de usar los logs de eventos en ambientes *Windows* o el *syslog* para *Linux* y *MacOS*. De igual manera, es bueno utilizar bitácoras independientes al sistema operativo, por ejemplo los archivos W3C dentro de los servidores web.

Al trabajar dentro de un ambiente especial como una caja de arena se deben habilitar bitácoras especiales, ya que muchas de las veces dichas cajas de arena no tendrán acceso a grabar los eventos que ocurran en los *logs* de los sistemas en que residen.

Dentro de las bitácoras propias, no es necesario almacenar mucha información que sea sensible, solamente se requiere la información que realmente ayude a detectar una falla. Una buena bitácora al menos debe de incluir la siguiente información:

- Eventos sesión para usuarios, como autorización y autenticación
- Dirección IP de la máquina del cliente
- Fecha y hora del evento dentro del sistema
- Código específico del evento
- Descripción del evento
- La salida que produjo dicho evento
- Cambios en la configuración inicial del sistema
- Modificación o alteración de los registros de bitácora

Como una buena práctica, está determinar el tipo de log que se usará dentro de la aplicación, para no estar almacenando muchos registros innecesarios, el programador debe de ser capaz de distinguir entre logs para tareas forenses, los *logs* del sistema y los que se utilizan para auditorías. Todo esto con el fin de no almacenar información o registros innecesarios.

# 2. RECOMENDACIONES PARA EL TESTEO DE APLICACIONES

Las actividades de testeo, sirven principalmente para reducir de manera drástica los *Bugs* relacionados con seguridad, descubrir posibles vulnerabilidades y que no sean los usuarios y/o usuarios maliciosos los que las detecten. No se trata de agregar más seguridad al sistema, solo se trata de medir qué tan robusto es el sistema antes de salir a producción.

Existen métodos automáticos de testeo que son destinados a detectar ciertos tipos de *Bugs* relacionados con seguridad, dichos métodos son aplicados sobre el código fuente durante toda la etapa de desarrollo; este proceso se puede realizar de manera periódica durante todo el ciclo de desarrollo, ya que dichos métodos son muy fáciles de aplicar.

Otros tipos de métodos un poco más sofisticados, consisten en definir casos de testeo de seguridad basados principalmente en los resultados del modelo de amenazas.

Este tipo de pruebas difiere de las pruebas realizadas por la gente de control de calidad, ya que no se evalúa funcionalidad del sistema, sino más bien se prueba el sistema de manera maliciosa, para poner al descubierto cualquier vulnerabilidad oculta dentro del sistema.

## 2.1. Definir la superficie de ataques

Un prerrequisito indispensable para un testeo exitoso es identificar una posible área o superficie de ataque, la cual se encontrará definida en el modelo de amenazas, previamente definido.

Al tener bien clara y comprendida la superficie de ataques, el testeo debe de enfocarse en las áreas más peligrosas, o que se definen como las más vulnerables. La mayoría de las veces resulta ser el área con más riesgo, aquella que se encarga de capturar o procesar la información de entrada del sistema.

## 2.2. Diferentes tipos de herramientas para testeo

Existen varios tipos de herramientas de testeo, según sea el objetivo que se persigue, se puede realizar testeo *Fuzzing* o robustez para encontrar errores en el código fuente, generando de manera aleatoria información de entrada al sistema.

Además del testeo *Fuzz* se pueden utilizar escáneres de vulnerabilidades para redes y aplicaciones; estos escáneres pueden evaluar ciertos tipos de vulnerabilidades tales como la inyección de SQL dinámico o de *Cross-site scripting.*

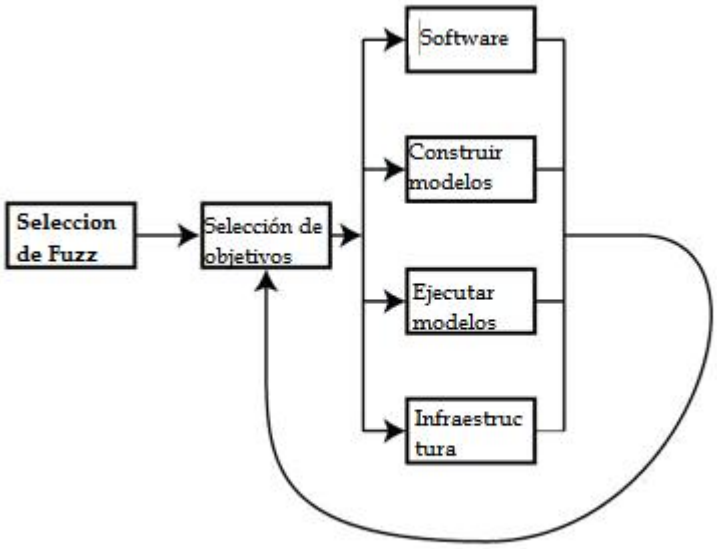
Otro tipo bastante utilizado de testeo es el conocido como Testeo de Penetración. Tiene como principal objetivo tratar de penetrar al sistema utilizando todos los métodos conocidos y utilizados por los usuarios maliciosos o *hackers*.

## 2.3. Test Fuzz o de Robustez

Es una técnica de testeo de *software*, automática o semiautomática, que consiste en ingresar datos inválidos, inesperados y aleatorios a un sistema, para verificar cómo se comporta el sistema frente a esta información alterada. En otras palabras es forzar a un sistema a utilizar y/o consumir data corrupta.

Si una aplicación soporta todas las pruebas de *fuzzing* no quiere decir que realmente esté bien la aplicación; el *fuzz testing* no sustituye por completo las pruebas exhaustivas o pruebas clásicas de funcionalidad. El *fuzzing* como tal solo indica si una aplicación maneja bien las excepciones, cuando se recibe información inválida y verifica que el sistema pueda seguir funcionando.

Figura 4. **Diagrama de un proceso de Fuzzing**




Fuente: elaboración propia.

Una de las principales desventajas del *fuzzing* es que solo sirve para encontrar fallas muy simples u obvias; con la complejidad que manejan ahora los diferentes programas, resulta muy difícil que solo con un test de *fuzzing* sea suficiente para validar una aplicación. Otra desventaja es la popularidad que esta herramienta ha adquirido, dentro de los evaluadores de aplicaciones; pero también muy utilizada ahora por los *hackers*, siendo el objetivo principal de estos hacer fallar las aplicaciones y obtener información sensible.

Por otro lado una de las ventajas del *fuzzing* es encontrar errores bastante severos relacionados a la seguridad, y descubrir vulnerabilidades que suelen ser descubiertas por los usuarios maliciosos.

En el mercado ya se encuentran varias herramientas que realizan dicha labor, pero en muchos de los casos es mejor desarrollar una herramienta propia, para que se acople de mejor manera al sistema que se está evaluando.

Existen dos tipos de test de *fuzzing*, uno se denomina el *fuzzing* válido, en el que se utiliza información válida, no se ingresan datos ilógicos. Por otro lado se tiene el *fuzzing* sencillo, en el que no se valida el tipo de información que se envía, pues puede venir cualquier tipo de información aleatoria. Se puede tener un tercer tipo que sería la combinación de ambos métodos, una parte con información lógica y una segunda etapa donde se incluya cualquier tipo de información.

Se puede realizar otra clasificación de los test de *Fuzzing,* tomando como base la manera en que la información llega al sistema:

- Activación o ingreso por eventos, para sistemas con una interfase gráfica, orientados a programación por eventos.

- Flujo de datos o caracteres, tomando como base archivos de texto o cadenas de caracteres.
- Por medio de bases de datos, datos de manera tabular y ordenada.

## 2.4. Test de penetración

El objetivo principal de un test de penetración es lograr ingresar de manera fraudulenta al sistema, utilizando las mismas técnicas que un *hacker* utiliza normalmente. Puede realizarse tanto de manera manual como por medio de herramientas.

Con esta técnica, se intenta detectar las vulnerabilidades que están relacionadas con la lógica del negocio que se encuentra incluida dentro del sistema.

Este tipo de test puede realizarse de dos maneras diferentes: primero con la participación de elementos ajenos al equipo de desarrollo (y de preferencia fuera de la empresa) y por medio de un equipo interno.

Las dos técnicas tienen sus ventajas y desventajas, habrá casos en los que el grupo de ataque es experto en el área de seguridad informática, pero no conoce a fondo el sistema que están evaluando. De igual manera habrá grupos que desarrollaron todo el sistema, pero nunca han realizado un test de penetración.

Existen varias maneras de realizar los test de penetración, como primer punto se debe determinar si el evaluador será una persona que no conoce el sistema; a este tipo de test se le conocerá como “testeo de caja negra”.

Caso contrario, si el evaluador tiene algún conocimiento del sistema se le llama a la prueba “Testeo de Caja Blanca”. Si se define que se hará un testeo con personas que ya conocen el sistema, se debe de contar con cierta información antes de proceder con la respectiva evaluación:

- Diagrama completo de la red.
- Evaluación completa de funcionalidad por parte del departamento de control de calidad.
- Evaluación previa de todos los puntos de acceso a redes inalámbricas.
- Resultados del test anterior de penetración (si no fuera el primero a realizar).
- Resultados de la evaluación de vulnerabilidades tanto internas como externas.
- Haber realizado recientemente una revisión de las políticas de seguridad de la empresa.

Al tener identificadas todas las amenazas y vulnerabilidades, se procede a definir los puntos que se estarán atacando durante la prueba, lugares de almacenamiento de información, redes internas, interfases, VPN, accesos remotos, redes inalámbricas, etc.

El objetivo principal de estas pruebas es tratar de explotar o utilizar las vulnerabilidades o debilidades con que cuenta el sistema; si alguna de las pruebas es exitosa (por ejemplo, se logra obtener información confidencial), se trabaja en corregir dicha intrusión, y proceder a evaluar de nuevo dicha vulnerabilidad, luego de los cambios realizados.

A continuación se detallan los pasos a seguir para realizar un test de penetración de una manera más formal:

### 2.4.1. Planificación y preparación

Para aplicar un test exitoso, primero se debe realizar una exhaustiva preparación; se puede empezar con una reunión inicial, con el fin de determinar el alcance y objetivos de la prueba. Luego se debe definir el tiempo que durará dicha prueba. Las horas en las que se harán los test, para que no afecte de manera significativa el diario funcionar de la organización.

Si existen sistemas muy delicados que posiblemente no resistan el ataque, deberán de excluirse de las pruebas; de igual manera procurar no saturar la red con mayor tráfico del normal, para no tener ningún efecto dañino dentro de la organización.

### 2.4.2. Recopilación y análisis de información

Luego de una exhaustiva planificación viene el proceso de recopilar la mayor cantidad de información posible acerca del sistema que se evaluará. Para esta tarea pueden utilizarse aplicaciones que se encuentran disponibles en el mercado.

El primer punto donde se debe recabar información es el sitio web de la organización; se determina dentro de la red el punto por el cual se tiene la conexión con internet, y se listan todos los recursos que pueden accederse desde allí.

Luego se pasa a la red interna de la organización donde se enumera el número de máquinas o servidores que pueden afectarse. Qué información suele obtenerse, básicamente nombres de servidores, direcciones IP, nombre del dominio, *proxies, firewalls*, etc.

Paso siguiente es la realización de un escaneo de puertos, para determinar cuáles se encuentran cerrados o abiertos.

### 2.4.3. Detección de vulnerabilidades

Al tener la información del sistema que va a ser evaluado, se procede con la detección de todas las vulnerabilidades; el evaluador deberá contar con todas las posibles amenazas y vulnerabilidades que posee el sistema; luego de tenerlas enumeradas, deberá determinar si es factible explotar dicha vulnerabilidad para penetrar en el sistema.

En la actualidad ya existen herramientas especializadas en detección de vulnerabilidades, básicamente se le proporciona una dirección de la red que se desea evaluar, y esta de manera remota determina todas las vulnerabilidades que pueden ser explotadas.

Por último, al tener la lista definitiva de las vulnerabilidades, se procede a investigar con más detalle, cada una de estas. Y así proceder a la siguiente etapa que es el ataque como tal.

### 2.4.4. Intento de penetración

Al tener identificadas todas las vulnerabilidades, el siguiente paso es verificar los blancos u objetivos más apropiados para un intento de penetración.

Se da inicio al ataque, a cada una de las vulnerabilidades. El atacante no siempre va a lograr su objetivo, ya que dentro del sistema puede existir una vulnerabilidad, pero esto no quiere decir que pueda ser explotada fácilmente.

El cifrado de contraseñas se ha convertido en una de las evaluaciones más recurrentes practicadas en los tests de penetración. Muchos sistemas tienen activos los servicios de *Telnet* o *ftp*, por lo que este lugar resulta ser uno de los primeros en ponerse a prueba con los métodos de cifrado de contraseñas. Los métodos utilizados se pueden dividir en tres tipos:

- Ataque de diccionario: se basa el ataque en una lista de palabras o un archivo de diccionario.
- Cifrado hibrido: al listado de palabras base se le aplica una serie de combinaciones posibles.
- Ataque de fuerza bruta: se aplican todas las posibles combinaciones de caracteres y letras para formar contraseñas.

El ataque de penetración no solo consiste en *software*, también se puede atacar por medio de personas; entre los dos métodos más comunes, están el poder librar la seguridad física para accesar al centro de datos, o a una terminal; y el otro, de llamar directamente a los usuarios y por medio de mentiras, obtener sus diferentes contraseñas.

### 2.4.5. Análisis y elaboración del reporte final

Luego de los intentos de penetración, se procede a elaborar el reporte final; se inicia con un resumen ejecutivo explicando brevemente el proceso realizado, seguido por un análisis de las vulnerabilidades más criticas ordenadas en grado de importancia; esto para ayudar a la organización en la toma de decisiones. Otras secciones de dicho reporte pueden ser:

- Resumen de toda penetración exitosa
- Detalle de toda la información recopilada durante el proceso
- Detalle de todas las vulnerabilidades detectadas
- Sugerencias para resolver las vulnerabilidades detectadas

### 2.4.6. Limpieza luego de la evaluación

Al finalizar el proceso se debe de deshacer toda huella o mancha que se haya dejado dentro del sistema; para poder realizar dicha operación se debe de contar con una lista detallada de todas las actividades que se realizaron dentro del sistema.

La organización debe validar que todo lo que fue modificado sea regresado a su estado anterior, sin afectar de ninguna manera el correcto funcionamiento de los diferentes sistemas. Si por ejemplo fueron creadas nuevas cuentas de usuarios durante el proceso de penetración, estas deben de ser eliminadas.

## 2.5. Herramientas más utilizadas en la actualidad

Existe una diversidad de herramientas en el mercado para realizar evaluaciones de aplicaciones. Estas herramientas generalmente no se basan en un tipo específico de test sino que combinan de buena manera los diferentes métodos de evaluación ya mencionados.

Dependiendo de la necesidad de las empresas, puede optarse por utilizar herramientas gratuitas; para evaluar sitios pequeños o si se maneja una complejidad alta o un sistema muy grande, se debe adquirir herramientas comerciales que suelen ser más robustas o completas.

Una de las herramientas que más se utilizan en la actualidad es *Skipfish*, el cual fue desarrollado por la gente de *Google*.

El uso de una u otra herramienta no hace al *software* más seguro; se debe tener claro que estas herramientas no revisarán de manera específica todo el código de una aplicación dada. Por el contrario solo harán revisiones genéricas de ciertos puntos ya establecidos.

### 2.5.1. *Skipfish*

Esta herramienta multiplataforma desarrollada por Google tiene como fin primordial la detección de vulnerabilidades en aplicaciones o servicios Web.

Desarrollada completamente en lenguaje C, es muy veloz y consume muy poco recurso; dependiendo del sitio que se evalúe, puede generar de 500 a 2,000 *request* por segundo.

Funciona indistintamente en ambientes *Windows o Linux*, y dentro de las principales vulnerabilidades que puede detectar están las relacionadas con la inyección de SQL, manejo de sesiones y *cookies*, ataques de robustez y ataques de fuerza bruta.

Esta herramienta genera un reporte bastante detallado de todos los hallazgos realizados; para utilizarla puede descargarse gratuitamente en:

http://code.google.com/p/skipfish/

Al descargar la aplicación debe compilarse, ya que actualmente sólo se tiene el código fuente; al tener el ejecutable ya se puede empezar a evaluar sitios, la sintaxis para realizar dichas evaluaciones resulta ser muy fácil:

```
./skipfish  http://localhost/application/exam
```

En la siguiente figura se muestra un resumen de lo obtenido por la herramienta.

Figura 5. **Resultados de una evaluación con *Skipfish***


Fuente: elaboración propia, basada en prueba realizada con la herramienta.

### 2.5.2. W3af

Esta herramienta *opensource* es un marco de trabajo completo utilizado para descubrir, auditar y evaluar vulnerabilidades.

Como primera fase intenta detectar todos los posibles puntos de entrada a un sitio. La segunda fase se encarga de tomar todas las entradas detectadas en el primer punto y empieza a atacarla con información para determinar si existe o no una vulnerabilidad. Una de las primeras pruebas que realiza es intentar inyectar sentencias de SQL.

La última parte consiste en atacar cada una de las vulnerabilidades detectadas y determinar si pueden constituirse en algún riesgo. De igual manera, el *software* genera por cada una de las fases documentación en diferentes formatos desde txt, hasta respuestas en http. Dicha aplicación se encuentra disponible en el siguiente link:

http://www.w3af.com/#download

Esta herramienta, resulta ser muy poco amigable con el usuario, ya que todos los comandos son ejecutados desde una línea de comandos.

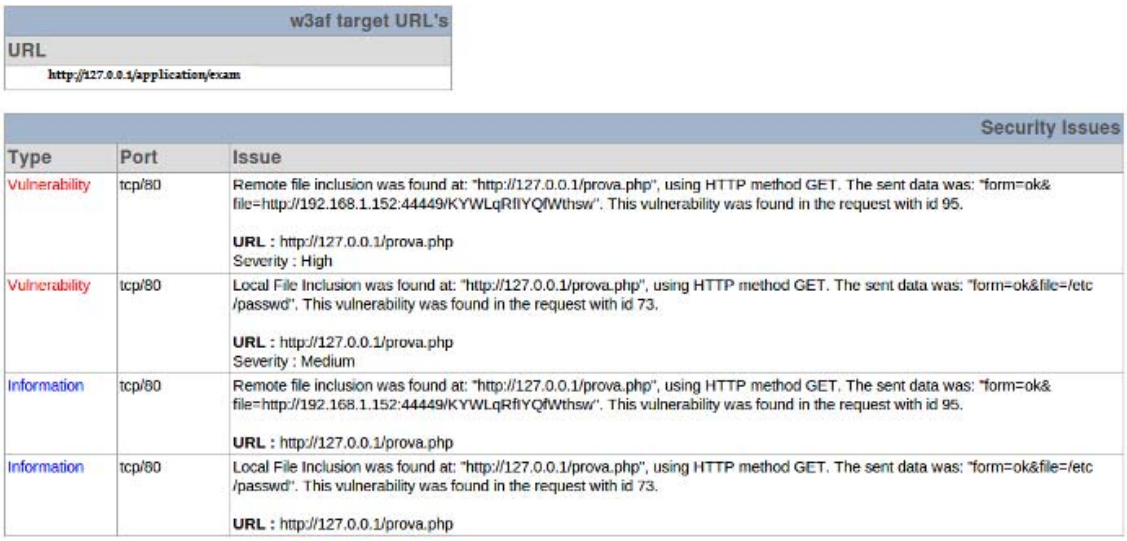
Para iniciar esta aplicación se ejecuta el siguiente comando:

```
sudo w3af_console
```

Dentro de esta consola debe irse ingresando uno a uno los diferentes comandos o instrucciones a ejecutarse. Los comandos más conocidos son: *start* para iniciar la revisión de un sitio, *target* para indicar cual será el sitio que se estará evaluando y *view* para identificar los parámetros por defecto.

El reporte final puede visualizarse con un navegador, como se muestra en la siguiente figura.

Figura 6. **Resultados de una evaluación con *w3af***



w3af target URL's

| URL |
| --- |
| http://127.0.0.1/application/exam |

Security Issues

| Type | Port | Issue |
| --- | --- | --- |
| Vulnerability | tcp/80 | Remote file inclusion was found at: "http://127.0.0.1/prova.php", using HTTP method GET. The sent data was: "form=ok&file=http://192.168.1.152:44449/KYWLqRfIYQfWthsw". This vulnerability was found in the request with id 95.<br><br>**URL :** http://127.0.0.1/prova.php<br>Severity : High |
| Vulnerability | tcp/80 | Local File Inclusion was found at: "http://127.0.0.1/prova.php", using HTTP method GET. The sent data was: "form=ok&file=/etc/passwd". This vulnerability was found in the request with id 73.<br><br>**URL :** http://127.0.0.1/prova.php<br>Severity : Medium |
| Information | tcp/80 | Remote file inclusion was found at: "http://127.0.0.1/prova.php", using HTTP method GET. The sent data was: "form=ok&file=http://192.168.1.152:44449/KYWLqRfIYQfWthsw". This vulnerability was found in the request with id 95.<br><br>**URL :** http://127.0.0.1/prova.php |
| Information | tcp/80 | Local File Inclusion was found at: "http://127.0.0.1/prova.php", using HTTP method GET. The sent data was: "form=ok&file=/etc/passwd". This vulnerability was found in the request with id 73.<br><br>**URL :** http://127.0.0.1/prova.php |

Fuente: elaboración propia, basada en prueba realizada con la herramienta.

### 2.5.3. Nikto

Es una herramienta para evaluar servidores WEB, fue desarrollada con tres fines primordiales: detectar archivos de configuración desactualizados, programas y servidores desactualizados y archivos inseguros. Puede ser utilizado en cualquier plataforma que sea compatible con *Pearl*.

Se puede utilizar dicha herramienta enviándole una dirección IP, el nombre de un sitio o un *url* completo, el cual se encarga de empezar a revisarlo; si en dado caso no se define un puerto por defecto, él asume el puerto 80.

Esta herramienta puede revisar varios puertos a la vez, enviando como parámetro un rango o un listado de los mismos. De igual manera como otras herramientas, revisa las bases de datos relacionadas, para no tener problemas con inyección de código SQL.

A continuación se listan todos los tests, que ejecuta dicha herramienta:

- Carga de archivos
- Archivos principales
- Malas configuraciones del sistema
- Problemas de divulgación de información
- Inyección de XSS/Script/HTML
- Recuperación remota de archivos
- Negación del servicio
- Ejecución remota de línea de comandos
- Inyección de SQL
- Identificación por *software*
- Inclusiones remotas al código fuente

http://cirt.net/nikto2

Al igual que las otras herramientas revisadas anteriormente, esta funciona con comandos o instrucciones. Para ejecutar un simple escaneo basta con ingresar la siguiente línea de código:

```
perl nikto.pl -h localhost/application/exam
```

Como ya se indicó, pueden revisarse varios puertos a la vez usando el parámetro –p y se da un rango o un listado de puertos.

### 2.5.4. Websecurify

Utilizada para detectar vulnerabilidades, basada en testeos Fuzz, y en testeos de penetración; dentro de los tipos de vulnerabilidades que verifica se pueden destacar los siguientes:

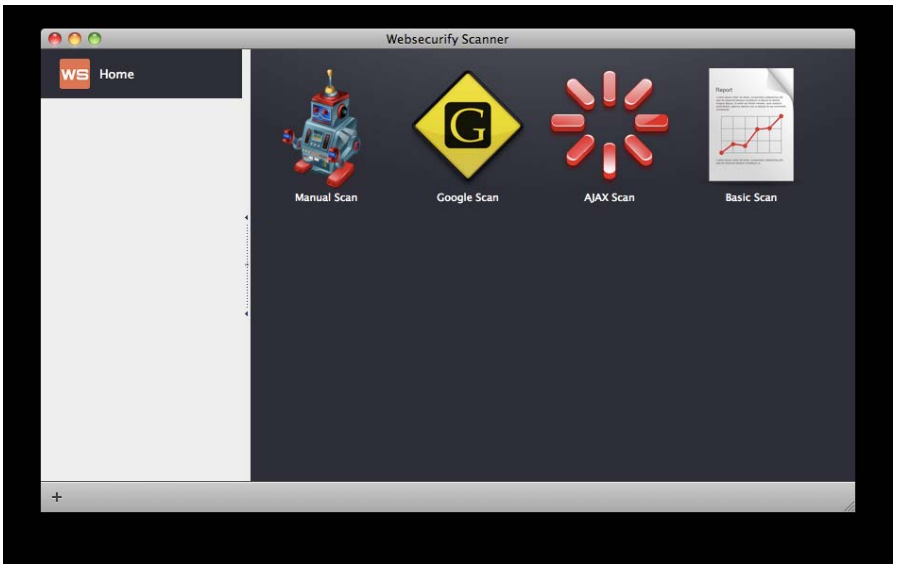
- Inyección de SQL
- Inclusión de archivos locales y remotos
- *Cross-site Scripting*
- Problemas de divulgación de información
- Problemas de seguridad en sesiones
- Otras varias categorías incluyendo las primeras 10 señaladas por la OWASP

Esta herramienta, hace mucho hincapié en el uso de los testeos de penetración. Puede complementarse fácilmente con el uso de ADDS – ONS (si se desea probar especificaciones propias de las aplicaciones).

http://www.websecurify.com

De todas las herramientas revisadas anteriormente, esta resulta ser de las más amigables, ya que la interacción con el usuario es en modo gráfico, desde el inicio hasta el reporte final, tal y como se muestra a continuación.

Figura 7. **Consola principal del Websecurify**




Fuente: elaboración propia, basada en prueba realizada con la herramienta.

# 3. GUÍA DE EVALUACIÓN DE VULNERABILIDADES PARA UNA APLICACIÓN WEB

## 3.1. Definir aspectos o métricos a evaluar

En los modelos de amenazas, se indicó la necesidad de evaluar todas y cada una de las posibles vulnerabilidades, tanto el impacto que causarán como qué tan probable será que una vulnerabilidad se logre concretar.

Como parte central en la definición de una metodología para evaluar vulnerabilidades, se deben definir los aspectos que se tomarán en cuenta para proceder a evaluar una aplicación; a continuación se define cada uno de ellos:

- Daño potencial: si por alguna razón ocurre una amenaza, y esta se llega a concretar, con esta medida se quiere determinar el nivel de impacto que causará; su rango de medición será desde un daño nulo, hasta considerar el daño más severo.

- Forma de activarse: esta medida determina la complejidad o facilidad que tendrá una amenaza en activarse, su rango se tomará desde una secuencia de pasos complejos, hasta un programa que se ejecuta de manera automática tan solo con ingresar al sistema (*Plug and Play*).

- Daño a usuarios: esta medida indica el total de usuarios finales que se ven afectados debido a la ocurrencia del problema; acá no se evalúa un número exacto de usuarios damnificados si no más bien qué porcentaje se

ve afectado, su rango va desde un 0% hasta el 100% de usuarios afectados.

- Nivel de detección: con esta medida se determina si una falla o amenaza es fácil de detectar, o si una amenaza puede estar oculta y solo puede ser detectada por medio de procedimientos exhaustivos.

- Confidencialidad: este es uno de los principales métricos, ya que está de manera directa relacionado con el fin u objetivo de la empresa, y tomando como premisa que el activo más importante de una empresa es su información. Acá se determina si la integridad de la información está o no comprometida.

- Disponibilidad de la aplicación: una aplicación luego de recibir un ataque puede verse afectada en el funcionamiento; esta medida indica si la aplicación puede mantenerse en funcionamiento o si dicho funcionamiento se ve afectado, acá también se evalúa el tiempo que una aplicación puede pasar fuera de servicio.

- Corrección del problema: luego de haberse dado la amenaza, viene el proceso de recuperación; con este métrico se determina qué tan difícil es resolver el daño ocasionado.

- Daño colateral: cada aplicación se utiliza en diferentes ámbitos, dichas aplicaciones en la mayoría de casos, son parte importante de las organizaciones. Con el daño colateral se evalúa qué tanto daño causa la vulnerabilidad en el entorno en que se encuentra el sistema y la organización. Cabe mencionar que acá se pueden enmarcar daños gravísimos como fallecimientos de personas, si se refiere a aplicaciones

dentro de hospitales, o pérdida de activos o dinero, si se trata de empresas financieras.

### 3.2. Ponderación y priorización de los diferentes métricos evaluados

Cada uno de los métricos utilizados será valuado en un rango de 0 (cero) a 10 (diez). Luego se sumarán todos los valores obtenidos y se dividirá dentro de ocho (8). El total final será redondeado para obtener un número entero siempre dentro del rango entre 0 y 10.

*Punteo = Redondeo ([M1 + M2 + M3 +… + M9] / 8)*

Para cada uno de los métricos se tienen 3 posibles valores, desde valor 0 (cero o nulo, ninguna afectación), 5 (valor intermedio) y 10 (valor máximo o nota máxima que se puede asignar). Aunque según el grado de afinamiento que se le quiera dar a este sistema, igual se puede utilizar cada uno de los valores desde 0 hasta 10, para dar una medida mucho más precisa.

A continuación se muestran ciertos aspectos para considerar el punteo que se le dará a cada uno de los métricos evaluados.

Tabla III. **Valores a asignar a los diferentes métricos**

| **Daño potencial** | |
|---|---|
| **Valor asignado** | **Descripción** |
| 0 | No causará ningún daño |
| 5 | Cuando la información de un usuario específico es comprometida o afectada |
| 10 | Sistema completo es comprometido o destruido |

Continuación tabla III…

| Forma de reproducirse | |
|---|---|
| **Valor asignado** | **Descripción** |
| 0 | Muy difícil de reproducirse, hasta para un usuario experto o administrador |
| 5 | Nivel intermedio de permisos, y una secuencia de 2 o tres pasos para iniciarlo |
| 10 | Solo ingresar una dirección en un navegador, además puede ejecutarlo cualquier usuario, sin ninguna validación o verificación |
| **Daño a usuarios** | |
| **Valor asignado** | **Descripción** |
| 0 | Ningún usuario verá interrumpido su diario operar |
| 5 | Varios usuarios son afectados, pero no todos |
| 10 | Todos los usuarios de la aplicación saldrán afectados |
| **Nivel de detección** | |
| **Valor asignado** | **Descripción** |
| 0 | Casi imposible de detectar, se necesita permisos de administrador para detectarlos y acceso al código fuente |
| 5 | Puede ser detectado, con el simple hecho de hacer algún monitoreo dentro de la red |
| 10 | Aparece en la barra de navegación o en alguna de las formas o pantallas del sistema |
| **Confidencialidad** | |
| **Valor asignado** | **Descripción** |
| 0 | No existe ningún impacto en la confidencialidad del sistema |
| 5 | Divulgación considerable de información, puede darse el acceso a archivos propios del sistema, solo se cuenta con acceso a ciertas tablas de la base de datos |
| 10 | Divulgación completa de la información, el intruso tiene el control sobre el sistema, la información puede ser manipulada por completo |
| **Disponibilidad de la aplicación** | |
| **Valor asignado** | **Descripción** |
| 0 | No existe ningún impacto en la disponibilidad del sistema |
| 5 | Reducción en el rendimiento del sistema, interrupciones en el uso de los recursos del sistema |
| 10 | Sistema o recurso completamente fuera de servicio |

Continuación tabla III…

| Corrección del problema | |
|---|---|
| **Valor asignado** | **Descripción** |
| 0 | Existe algún parche o programa que corrija de manera oficial el problema |
| 5 | Solo se podrá aplicar una corrección temporal, no existe por el momento una solución definitiva |
| 10 | Aún no existe una solución completa al problema |
| **Daño collateral** | |
| **Valor asignado** | **Descripción** |
| 0 | No existe posibilidad de pérdida de vidas humanas, activos de la empresa, producción o ingresos. |
| 3 | Ligera repercusión en la operatoria de la organización |
| 5 | Puede repercutir en un daño moderado, o en la pérdida de ganancias de manera limitada a la organización |
| 8 | Daño significativo físico o de destrucción en la organización. Pérdida significativa de ingresos |
| 10 | El nivel máximo de destrucción que puede afectar a una organización (según el ámbito en que se desenvuelva) |

Fuente: elaboración propia.

La asignación de punteos a cada uno de los métricos suele ser bastante subjetiva por lo que debe de realizarse en consenso con cada uno de los participantes o conocedores del sistema, de igual manera si se requiere un mayor grado de exactitud se pueden incluir más valores a cada métrico o algunos de ellos, según se considere conveniente.

### 3.3. Determinar el grado de seguridad de una aplicación

Al trabajar con un modelo de amenazas se procedió a definir una lista de todas las posibles amenazas o vulnerabilidades que pueden afectar a un sistema. A esta lista de amenazas debe de aplicársele el sistema de evaluación definido anteriormente.

Para observar de mejor manera el sistema de evaluación se presenta un caso de una tienda por departamentos, que desea implementar una tienda virtual.

Tienda por departamentos “Almacenes Superior”

Una tienda por departamentos llamada Almacenes Superior, desea implantar un sitio Web para habilitar su “Tienda Virtual”. El sitio Web debe proporcionar información de existencias, precios, descuentos y promociones especiales, según la época. Toda la información sobre clientes, productos y existencias se encuentran en un servidor NT con *SQL Server 2003* como base de datos principal. El sistema se debe de desarrollar en *Visual Basic .NET*.

Dentro de las funciones principales que debe de incluir el sistema están:

- Creación de clientes y usuarios de la aplicación
- Compras por internet utilizando tarjetas de crédito u otros mecanismos como *PayPal*
- Consultas de existencias de productos en los diferentes departamentos
- Activación y envío de claves por correo electrónico
- Fidelización o programas de lealtad
- Ayuda en línea para compradores especiales
- Inventario 100% en línea

Las siguientes amenazas/ataques pueden afectar a la aplicación:

- Ataques de fuerza bruta o de diccionario contra la información encriptada en la base de datos.
- Espionaje dentro de la red para capturar credenciales de los clientes.
- Un *hacker* puede capturar una "*cookie*" de autenticación para suplantar la identidad del usuario registrado.
- Inyección de código SQL, para ejecutar comandos en la base de datos con el fin de accesar y/o modificar información.
- Duplicación o captura de "*cookies*" permitiendo al *hacker* robar la identidad de un usuario y registrarse como un usuario válido.
- Disponibilidad de información sensible para cualquier cliente sin ninguna excepción, bastará con solicitarla o buscarla.
- Un *hacker* puede llegar a tomar el control del servidor Web, obtener accesos no autorizados a la base de datos y ejecutar comandos contra la base de datos.
- El *hacker* obtiene las llaves de encripción, que se utilizan para encriptar información sensible (incluyendo números de tarjeta de crédito).
- Un *hacker* o cliente normal puede tener accesos no autorizados al servidor Web y sus recursos.

Cada una de estas amenazas fueron evaluadas por los métricos definidos al inicio de este capítulo y en las siguientes tablas se muestran los resultados.

Tabla IV. **Ponderación ataques de fuerza bruta**

| **Ataques de fuerza bruta o diccionario a la información encriptada** | | |
|---|---|---|
| DAÑO | 5 | |
| ACTIVACIÓN | 5 | |
| USUARIOS | 10 | |
| DETECCIÓN | 5 | |
| CONFIDENCIALIDAD | 10 | |
| DISPONIBILIDAD | 5 | |
| CORRECCIÓN | 5 | |
| COLATERAL | 10 | |
| NOTA FINAL | | **7** |

Fuente: elaboración propia.

Tabla V. **Captura de credenciales a usuarios**

| **Captura de credenciales a usuarios por medio del Internet** | | |
|---|---|---|
| DAÑO | 10 | |
| ACTIVACIÓN | 5 | |
| USUARIOS | 10 | |
| DETECCIÓN | 10 | |
| CONFIDENCIALIDAD | 10 | |
| DISPONIBILIDAD | 5 | |
| CORRECCIÓN | 10 | |
| COLATERAL | 10 | |
| NOTA FINAL | | **9** |

Fuente: elaboración propia.

Tabla VI. **Captura de *cookies***

| **Capturar *cookies* de identificación** | | |
|---|---|---|
| DAÑO | 10 | |
| ACTIVACIÓN | 5 | |
| USUARIOS | 10 | |
| DETECCIÓN | 0 | |
| CONFIDENCIALIDAD | 10 | |
| DISPONIBILIDAD | 5 | |
| CORRECCIÓN | 10 | |
| COLATERAL | 8 | |
| NOTA FINAL | | **7** |

Fuente: elaboración propia.

Tabla VII. **Inyección código SQL**

| **Inyección de código SQL** | | |
|---|---|---|
| DAÑO | 10 | |
| ACTIVACIÓN | 0 | |
| USUARIOS | 5 | |
| DETECCIÓN | 0 | |
| CONFIDENCIALIDAD | 5 | |
| DISPONIBILIDAD | 10 | |
| CORRECCIÓN | 10 | |
| COLATERAL | 7 | |
| NOTA FINAL | | **6** |

Fuente: elaboración propia.

Tabla VIII. **Duplicidad de *cookies***

| **Duplicidad de *cookies* para suplantar identidad** | | |
|---|---|---|
| DAÑO | 10 | |
| ACTIVACIÓN | 5 | |
| USUARIOS | 10 | |
| DETECCIÓN | 10 | |
| CONFIDENCIALIDAD | 10 | |
| DISPONIBILIDAD | 5 | |
| CORRECCIÓN | 10 | |
| COLATERAL | 10 | |
| NOTA FINAL | | **9** |

Fuente: elaboración propia.

Tabla IX. **Disponibilizar información sensible**

| **Disponibilidad de información sensible para cualquier cliente sin restricciones** | | |
|---|---|---|
| DAÑO | 10 | |
| ACTIVACIÓN | 0 | |
| USUARIOS | 10 | |
| DETECCIÓN | 5 | |
| CONFIDENCIALIDAD | 10 | |
| DISPONIBILIDAD | 10 | |
| CORRECCIÓN | 10 | |
| COLATERAL | 10 | |
| NOTA FINAL | | **8** |

Fuente: elaboración propia.

Tabla X. **Tomar control del servidor web**

| **Tomar control del servidor WEB para enviar operaciones al servidor de base de datos** | | |
|---|---|---|
| DAÑO | 10 | |
| ACTIVACIÓN | 0 | |
| USUARIOS | 5 | |
| DETECCIÓN | 5 | |
| CONFIDENCIALIDAD | 5 | |
| DISPONIBILIDAD | 10 | |
| CORRECCIÓN | 10 | |
| COLATERAL | 8 | |
| NOTA FINAL | | **7** |

Fuente: elaboración propia.

Tabla XI. ***Hacker* se apodera de llaves para encripción**

| ***Hacker* obtiene las llaves para encripción de información** | | |
|---|---|---|
| DAÑO | 10 | |
| ACTIVACIÓN | 0 | |
| USUARIOS | 5 | |
| DETECCIÓN | 10 | |
| CONFIDENCIALIDAD | 10 | |
| DISPONIBILIDAD | 10 | |
| CORRECCIÓN | 10 | |
| COLATERAL | 10 | |
| NOTA FINAL | | **8** |

Fuente: elaboración propia.

Tabla XII. **Control de los recursos del servidor web y ponderación final de la aplicación**

| **Control de los recursos del servidor WEB** | | |
|---|---|---|
| DAÑO | 10 | |
| ACTIVACIÓN | 0 | |
| USUARIOS | 5 | |
| DETECCIÓN | 5 | |
| CONFIDENCIALIDAD | 5 | |
| DISPONIBILIDAD | 5 | |
| CORRECCIÓN | 10 | |
| COLATERAL | 7 | |
| NOTA FINAL | | **6** |

| **NOTA DE LA APLICACIÓN** | **7** |
|---|---|

Fuente: elaboración propia.

Al tener el punteo de cada amenaza, se procede a sacar un promedio simple y la nota final será comparada contra cada una de las siguientes notas:

- La nota más baja irá de 0 (cero) a 3 (tres); el sistema es muy robusto y tiene muy poca probabilidad de sufrir un ataque.

- El siguiente rango va de 4 (cuatro) a 5 (cinco), el sistema es moderadamente vulnerable; puede sufrir ataques muy especializados. Se debe definir qué vulnerabilidades son de mayor probabilidad de ocurrencia y cuáles pueden resolverse en un corto o mediano plazo.

- De 6 (seis) a 7 (siete), el sistema es vulnerable; se debe definir inmediatamente una estrategia para contrarrestar las amenazas más inminentes.

- Por último, si el sistema está arriba de 8 (ocho) o llega a 10 (diez) que es la nota máxima, se dice que es un sistema altamente vulnerable, se debe evitar que el sistema salga a producción (o inhabilitarlo si ya está en funcionamiento), y definir una estrategia para corregir de manera integral las amenazas más latentes. Al finalizar esta primera etapa de correcciones se procede a aplicarle la misma evaluación y ver el grado de avance en la resolución de problemas; aquí se podrá determinar si ya puede proceder a salir a producción o no.

Siguiendo con el ejemplo de los Almacenes Superior, la nota final fue de 7 (siete), lo que procede acá es verificar qué amenazas pueden resolverse en el corto y mediano plazo y cual será la metodología para mitigarlas.

A medida en que se vaya utilizando esta metodología y conforme se tome experiencia en el uso de la misma, todo el cuadro anterior podrá resumirse en una hoja de Excel más compacta como la mostrada en la tabla 13.

Tabla XIII. **Ponderación resumida de las amenazas del sistema de Almacenes Superior**

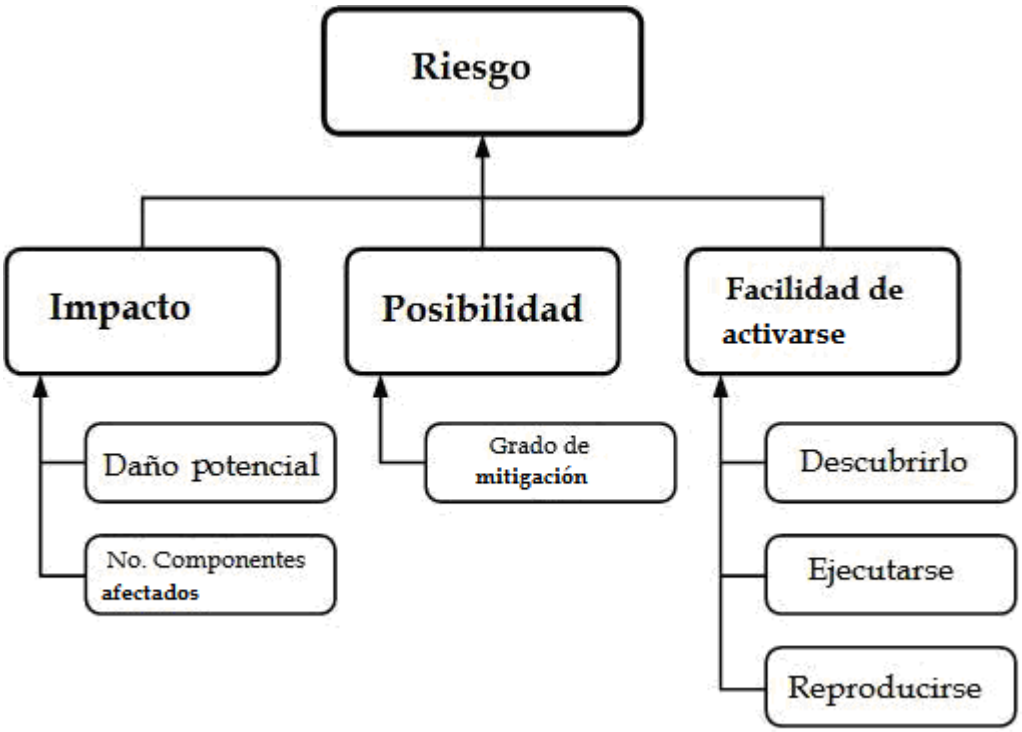
| VULNERABILIDAD | DAÑO | ACTIVACIÓN | USUARIOS | DETECCIÓN | CONFIDENCIALIDAD | DISPONIBILIDAD | CORRECCIÓN | COLATERAL | NOTA FINAL |
|---|---|---|---|---|---|---|---|---|---|
| Ataques de Fuerza Bruta o Diccionario a la información encriptada | 5 | 5 | 10 | 5 | 10 | 5 | 5 | 10 | 7 |
| Captura de credenciales a usuarios por medio del Internet | 10 | 5 | 10 | 10 | 10 | 5 | 10 | 10 | 9 |
| Capturar *cookies* de identificación | 10 | 5 | 10 | 0 | 10 | 5 | 10 | 8 | 7 |
| Inyección de código SQL | 10 | 0 | 5 | 0 | 5 | 10 | 10 | 7 | 6 |
| Duplicidad de Cookies para suplantar identidad | 10 | 5 | 10 | 10 | 10 | 5 | 10 | 10 | 9 |
| Disponibilidad de información sensible de clientes, sin ninguna restricción | 10 | 0 | 10 | 5 | 10 | 10 | 10 | 10 | 8 |
| Tomar control del servidor WEB para enviar operaciones al servidor de base de datos | 10 | 0 | 5 | 5 | 5 | 10 | 10 | 8 | 7 |
| Hacker obtiene las llaves para encripción de información | 10 | 0 | 5 | 10 | 10 | 10 | 10 | 10 | 8 |
| Control de los recursos del servidor WEB | 10 | 0 | 5 | 5 | 5 | 5 | 10 | 7 | 6 |
| | | | | | | | | | 0 |
| | | | | | | | | | 0 |
| | | | | | | | | | 0 |
| **NOTA FINAL** | | | | | | | | | **7** |

Fuente: elaboración propia.

## 3.4. Determinar orden para atacar las vulnerabilidades detectadas

Mitigar todas las amenazas y vulnerabilidades (luego de haberlas identificado a todas o la gran mayoría), puede resultar muy costoso tanto en recursos monetarios como recursos humanos. En algunos casos es prácticamente imposible el mitigar una vulnerabilidad sin tener que rediseñar por completo el sistema. Es en este preciso momento en que se tiene que utilizar ciertos criterios para determinar qué vulnerabilidades se van a enfrentar primero y cuáles se dejarán para una segunda fase o si existe alguna debilidad con la que se tendrá que convivir, ya que no se podrá solventar.

Figura 8. **Determinar el riesgo de una vulnerabilidad según el modelo definido**




Fuente: elaboración propia.

Cada una de las amenazas detectadas genera un riesgo al sistema y a la organización en sí. Para determinar el riesgo como tal, se debe de considerar los siguientes aspectos:

- Impacto que causará la amenaza.
- La probabilidad que dicha amenaza se presente dentro del sistema.
- El costo final de mitigar la amenaza comparado contra el máximo precio que puede llegar a costar si se produce dicho daño.
- Por último, determinar qué tan alta fue la nota que sacó la amenaza evaluada dentro del sistema de métricos definidos anteriormente.

Como objetivo primordial de la administración del riesgo, se tiene la reducción del impacto que pueda llegar a causar una amenaza. Para cumplir dicho objetivo se debe de enfrentar la amenaza con una estrategia de reducción del riesgo. En general existen 5 opciones para mitigar las amenazas:

- No hacer nada, esperar que la amenaza nunca se llegue a completar.
- Informar acerca del riesgo, realizar documentación, advertir a los usuarios de los posibles peligros.
- Mitigar el riesgo, aplicando medidas correctivas donde corresponda.
- Aceptar el riesgo, luego de haber evaluado el impacto en el negocio.
- Transferir el riesgo, trasladar el problema a entes externos, por ejemplo contratar un seguro contra ataques informáticos.

Al tener listo el orden en que las amenazas serán resueltas, se debe tener claro cómo se resolverá cada una de ellas. A continuación se presenta una tabla donde se define un tipo de amenaza y las posibles medidas a tomar para mitigarlas, definitivamente.

Tabla XIV. **Tipos de amenazas y medidas para suprimirlas**

| Amenaza | Contramedida |
|---|---|
| Autenticación | Credenciales o contraseñas de autenticación deben protegerse mediante encripción tanto en almacenamiento como en tránsito |
| | Usar protocolos resistentes a ataques de fuerza bruta, diccionario o reemplazo |
| | Implementar políticas para manejar contraseñas robustas |
| | Utilizar autenticación de confianza dentro del servidor y no la autenticación de SQL |
| | Al reiniciar contraseñas no deben de revelarse los nombres de usuarios válidos |
| Autorización | Listas de permisos para fortalecer los accesos autorizados a los recursos |
| | Accesos basados en roles para restringir el acceso a operaciones específicas |
| | Usar el principio de menos privilegios para cuentas de usuarios y servicios |

Continuación tabla XIV…

| | |
|---|---|
| Administración de la configuración | Procesos de menos privilegios son utilizados y las cuentas de servicios no tienen ningún permiso administrativo |
| | Habilitar todas las bitácoras posibles en las tareas de administración |
| | Accesos a los archivos de configuración, exclusivamente para los administradores del sistema |
| Protección de datos en almacenamiento y envíos | Uso de algoritmos estándares de encripción y manejo correcto del tamaño de llaves |
| | HMACs (*Hashed message authentication codes*) son utilizados para proteger la integridad de la información |
| | Toda data confidencial está encriptada tanto en transporte como en el almacenamiento |
| | Almacenamiento especializado para guardar las diferentes llaves |
| | No transferir al descubierto ni contraseñas ni data sensible |
| Validación de parámetros y datos | Reforzar verificaciones de tipos de datos, formatos, longitudes y rangos |
| | Validar toda la información enviada al cliente |
| | No tomar decisiones de seguridad basados en parámetros enviados |
| | Validar parámetros de entrada por medio de listas de datos |
| | Codificar los datos de salida |
| Manejo de Errores y excepciones | Manejar todas las excepciones de una manera estructurada |
| | Privilegios son restaurados a nivel apropiado en caso de producirse un error o una excepción |
| | Mensajes de error son eliminados por lo que el atacante no podrá revisarlos luego de sufrir un ataque |
| Administración de usuarios y sesiones | No almacenar en la *cookie* información sensible |
| | Los contenidos de las *cookies* de autenticación están encriptadas |
| | Las *cookies* deben estar configuradas para expirar al terminar una sesión |
| | Sesiones son resistentes a ataques de replicación |
| | Canales de comunicación seguros son utilizados para proteger las *Cookies* de autenticación |
| | Forzar al usuario a volverse a conectar cuando ejecute funciones críticas |
| | Las sesiones expiran al desconectarse del sistema |
| Bitácoras y auditorías | La información confidencial (por ejemplo, contraseñas, información de identificación personal, etc.) no se almacena |
| | Los controles de acceso (ACL, por ejemplo) se aplican en los archivos de bitácora, para evitar accesos no autorizados |

Fuente: elaboración propia.

## 3.5. Flexibilidad para adaptar la metodología de una guía de evaluación de vulnerabilidades

Una de las primeras dudas que resultan al utilizar esta metodología, es qué tanto la metodología se acoplará a las necesidades del negocio. Puede ser que no se necesite evaluar todos los métricos que acá se definen, o por el contrario, que se necesite incluir algún factor o métrico dentro del modelo.

Para estar conformes con dicha metodología, las empresas o instituciones pueden realizar 3 tipos de cambios propuestos:

### 3.5.1. Incluir más métricos

Se puede incluir más factores que representen una alta prioridad para la organización, que vayan acordes con el rol de la misma; por ejemplo las entidades gubernamentales, pueden incluir factores relacionados con la divulgación de secretos de Estado, o con incumplimientos en leyes o normativas establecidas. De igual manera se puede variar el método de cálculo, incluyendo las probabilidades de que suceda o no un ataque.

### 3.5.2. Ajustar opciones de evaluación

En las secciones anteriores se definieron 3 niveles de punteos: bajo, mediano y alto, representados por los valores 0, 5 y 10; para tener una nota más ajustada a la realidad se pueden incluir más factores que se relacionen directamente con la organización, y no solo sean tomados los que se propusieron como ejemplos.

### 3.5.3. Incluirle pesos a cada uno de los métricos

Este modelo actualmente le otorga el mismo peso o valor a cada uno de los diferentes métricos, pero según las necesidades de la organización se puede agregar un peso o valor extra a uno o varios factores; básicamente, para enfatizar los factores que son más importantes para la organización.

Todas estas variaciones van haciendo más complejo el modelo, pero a medida en que se vaya estandarizando, se podrá obtener una nota más real, y las decisiones tomadas con base en dicha nota, serán respaldadas de mejor manera.

# 4. MEDIDAS DE SEGURIDAD EN EL AMBIENTE GUATEMALTECO

Para determinar cuáles medidas de seguridad se aplican en el medio guatemalteco, se realizó una encuesta dirigida a profesionales de las áreas de tecnología, sistemas, auditoría y seguridad informática de empresas e instituciones.

## 4.1. Descripción de la encuesta

La encuesta fue publicada en un sitio especializado de encuestas bajo el nombre de *"Normas de seguridad aplicadas por las empresas guatemaltecas en el desarrollo de aplicaciones Web"*.

Fue distribuida dentro del mercado nacional vía correo electrónico y puede consultarse visitando el siguiente sitio:

http://www.encuestafacil.com/RespWeb/Qn.aspx?EID=960782

Esta encuesta se dividió en tres categorías:

- Demografía
- Fallas de seguridad
- Herramientas y prácticas de seguridad informática

## 4.2. Análisis de los resultados obtenidos

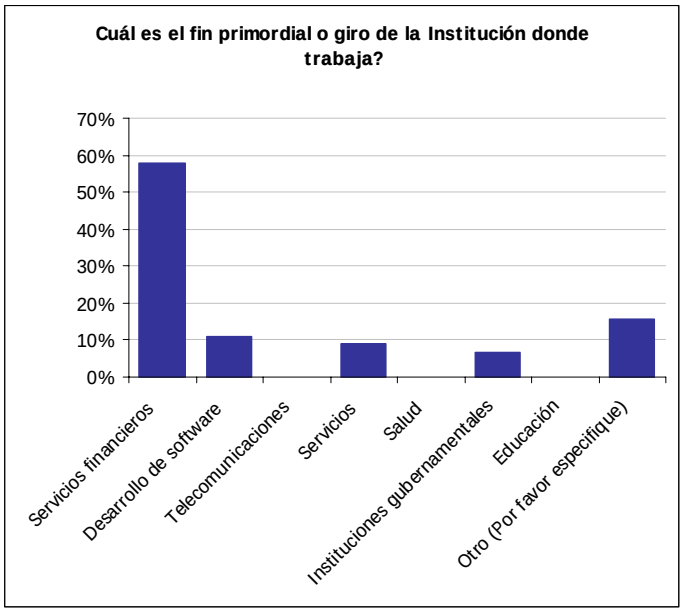
En la encuesta participó un total de 45 personas; los resultados obtenidos y su análisis por cada una de las categorías, se presentan a continuación. Las conclusiones y recomendaciones se harán en el apartado correspondiente.

### 4.2.1. Demografía

- Propósito**:** identificar los sectores económicos a los que pertenecen las empresas evaluadas, tamaño de las empresas involucradas, específicamente el tamaño del departamento encargado del soporte tecnológico y el perfil de la persona que participó en la encuesta.

- Análisis de resultados: en cuanto al sector de la economía se observa claramente que la mayoría de las personas que participaron de la encuesta trabajan en compañías de los sectores financiero y desarrollo de *software*, las empresas del sector gobierno y de servicios, están en una segunda categoría; no se contó con la participación del sector educativo, salud y telecomunicaciones.

Figura 9. **Naturaleza de las empresas evaluadas**

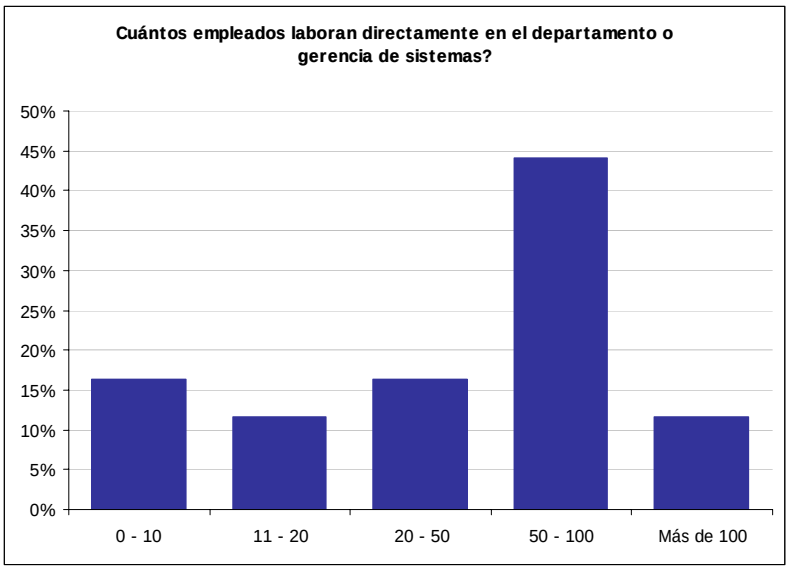



Fuente: elaboración propia.

En la muestra se puede determinar que la mayoría de empresas cuenta con un departamento de sistemas relativamente grande; casi un 50% de los encuestados trabaja en un departamento con un promedio de 50 a 100 colaboradores. Por el volumen tan grande de personas, hace suponer que en dichos departamentos existe una estructura bien organizada y que se cuenta con un área responsable de la seguridad informática.

Se observa que tanto las empresas con arriba de 100 colaboradores, como los demás rangos de empleados, manejan porcentajes bastante parecidos que oscilan entre el 10% y el 15%.

Figura 10. **Número de colaboradores en las empresas evaluadas**



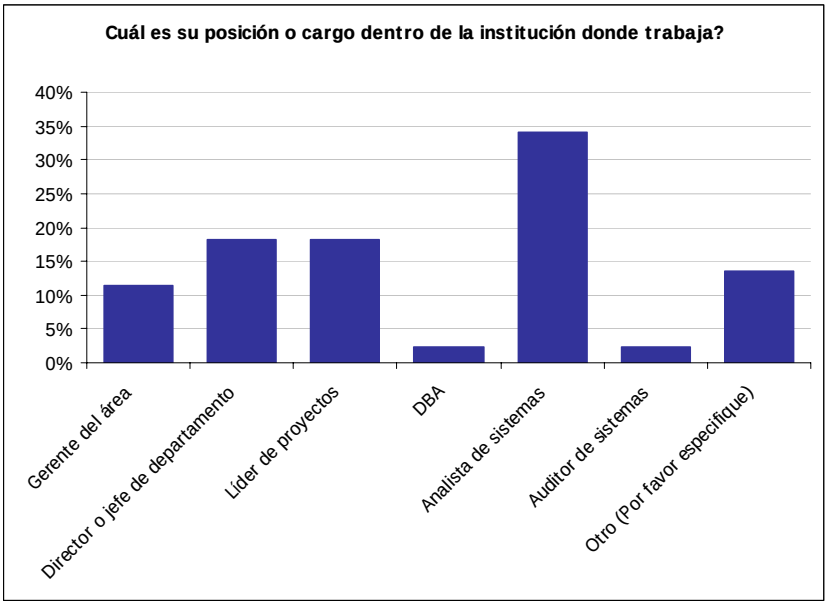

Fuente: elaboración propia.

Para esta encuesta se buscó a personas que tuvieran cierto grado de conocimiento y/o experiencia en el desarrollo de sistemas. Se seleccionaron puestos intermedios y altos dentro de la jerarquía de las empresas.

Muchas de las personas participantes pertenecen al área de desarrollo de sistemas; el mayor porcentaje de la muestra estuvo representado por el rol de analistas de sistemas con un 35%; el resto estuvo distribuido entre los líderes de proyectos y los gerentes o encargados de departamento.

Cabe mencionar que los puestos de auditores de sistemas y personal de control de calidad, tuvieron el menor porcentaje de participación.

Figura 11. **Puestos que ocupan los colaboradores en las empresas evaluadas**




Fuente: elaboración propia.

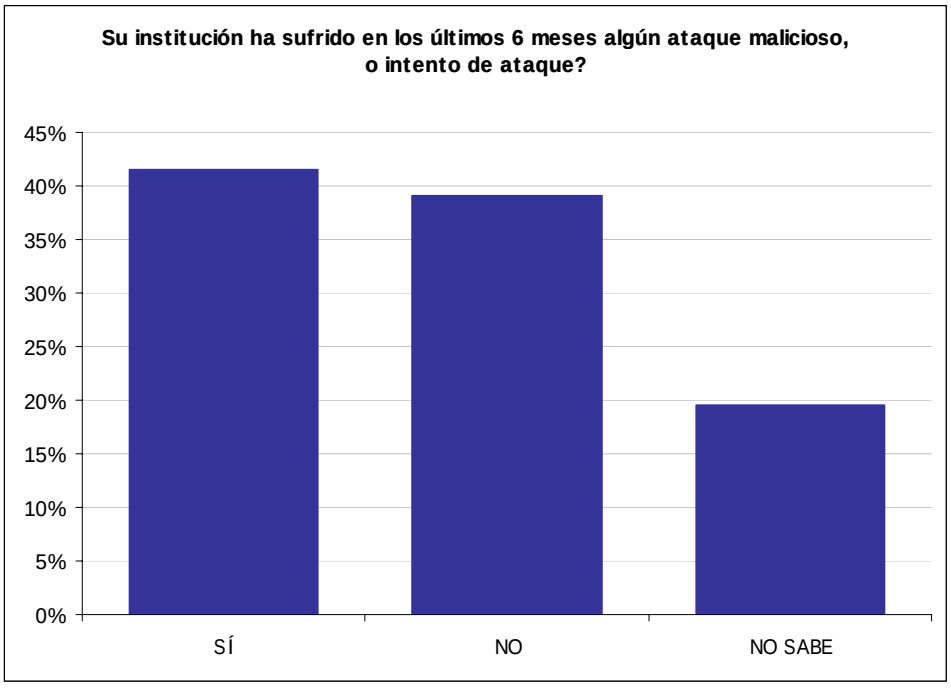
### 4.2.2. Fallas de seguridad

- Propósito: revisar los tipos de ataques o problemas de seguridad más frecuentes sobre las empresas participantes en los últimos 6 meses y determinar el tiempo promedio utilizado para corregir los problemas encontrados.
- Análisis de resultados**:** como resultado de la encuesta se detectó que en los últimos seis meses casi un 50% de las empresas de la muestra ha sufrido por lo menos un ataque, y un 20% no está enterada si sufrió o no algún ataque.

Para ayudar al 20% que no se encuentra informado, se debe trabajar mucho la parte de comunicación de los problemas que se han tenido, de igual manera informar la solución de los mismos.

Al informar a los colaboradores de la empresa de cualquier ataque o daño sufrido se está reduciendo de manera muy significativa, la probabilidad que este mismo problema se vuelva a dar; se hace concientización en el uso adecuado de los recursos informáticos y máxime cuando el ataque se debió a algún descuido humano.

Figura 12. **Ataques durante los últimos 6 meses**



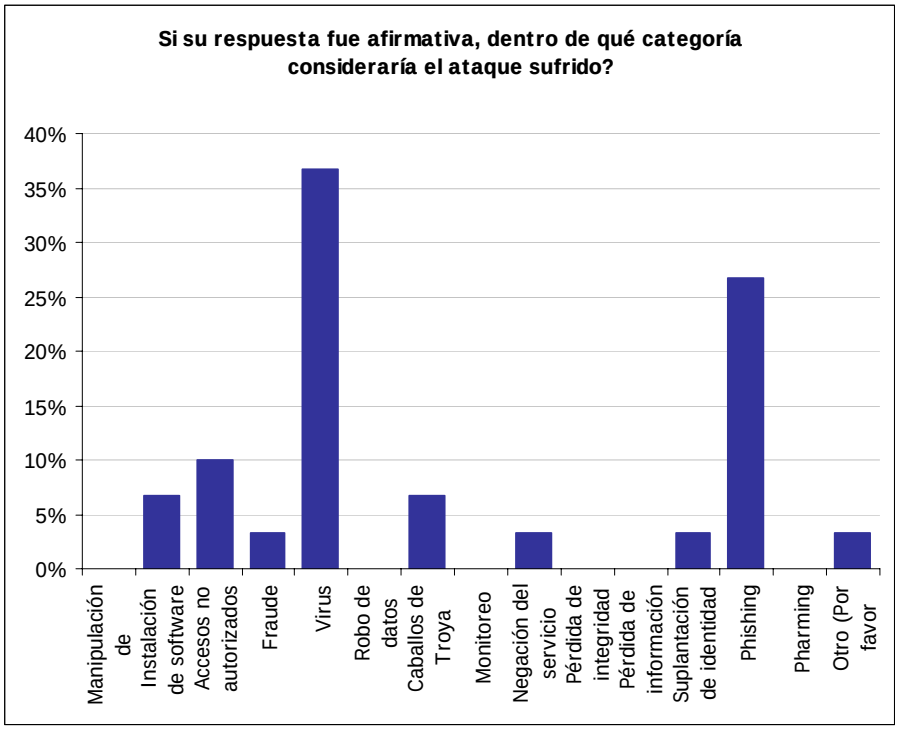

Fuente: elaboración propia.

Para determinar de mejor manera cuáles fueron los ataques que han sufrido las empresas en los últimos 6 meses, se enumeró cada una de las posibles amenazas.

Dentro de los ataques más frecuentes se encuentran los ataques de virus; esto era de esperarse ya que cada día aparecen nuevos tipos de ellos. De una manera un poco sorpresiva, el segundo más ocurrido es el *Phishing.*

El robo de identidad y el uso de permisos no autorizados aparecen en un tercer plano. De igual manera, existen 6 tipos de ataques que no fueron señalados, posiblemente por desconocimiento de los mismos o realmente porque no se suscitaron.

Figura 13. **Tipos de ataques sufridos durante los últimos 6 meses**



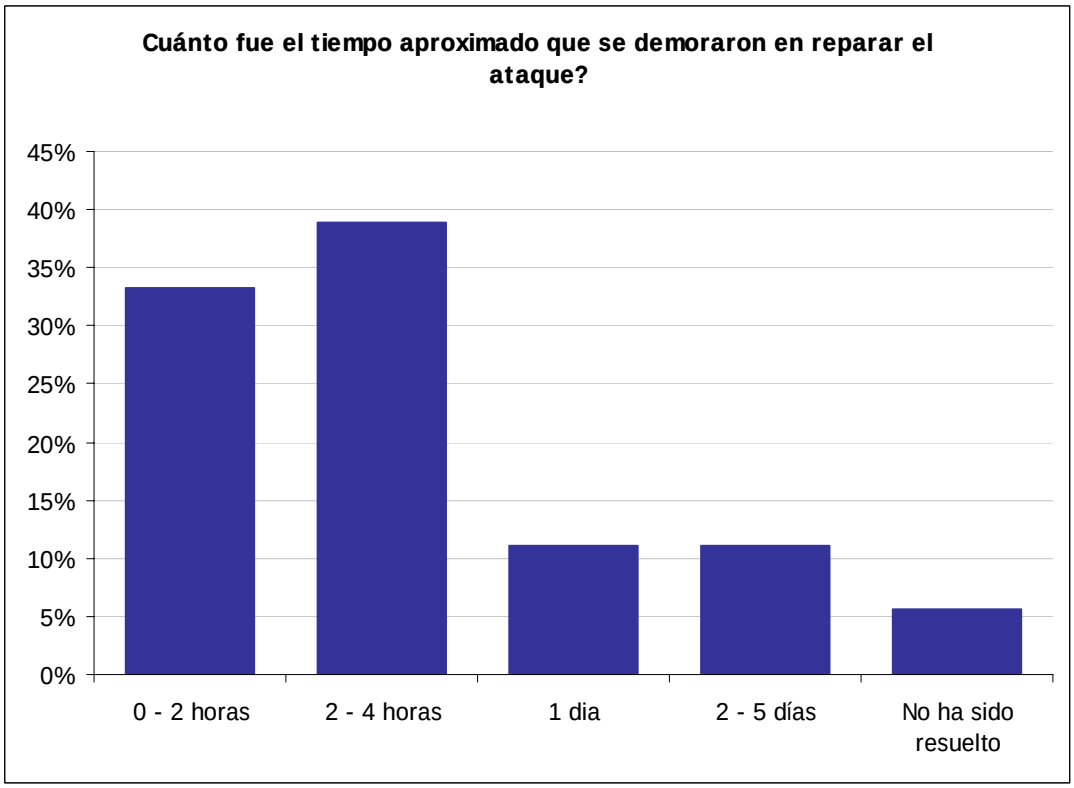

Fuente: elaboración propia.

Después de un ataque  viene el proceso de corrección o recuperación, el cual debe hacerse lo antes posible, para seguir con el funcionamiento normal del sistema.

Dentro de la encuesta se trató de determinar qué tanto tiempo se tarda un departamento de sistemas en resolver el problema.  Casi un 40% indicó que su tiempo promedio era de dos a cuatro horas; un 35%, de cero a dos horas.

Resulta bastante alentador que solo 4 personas respondieron que tardan más de un día en restablecer el sistema y sólo 1 persona aún no ha podido resolver su situación.

Figura 14. **Tiempo de recuperación luego de sufrir un ataque**



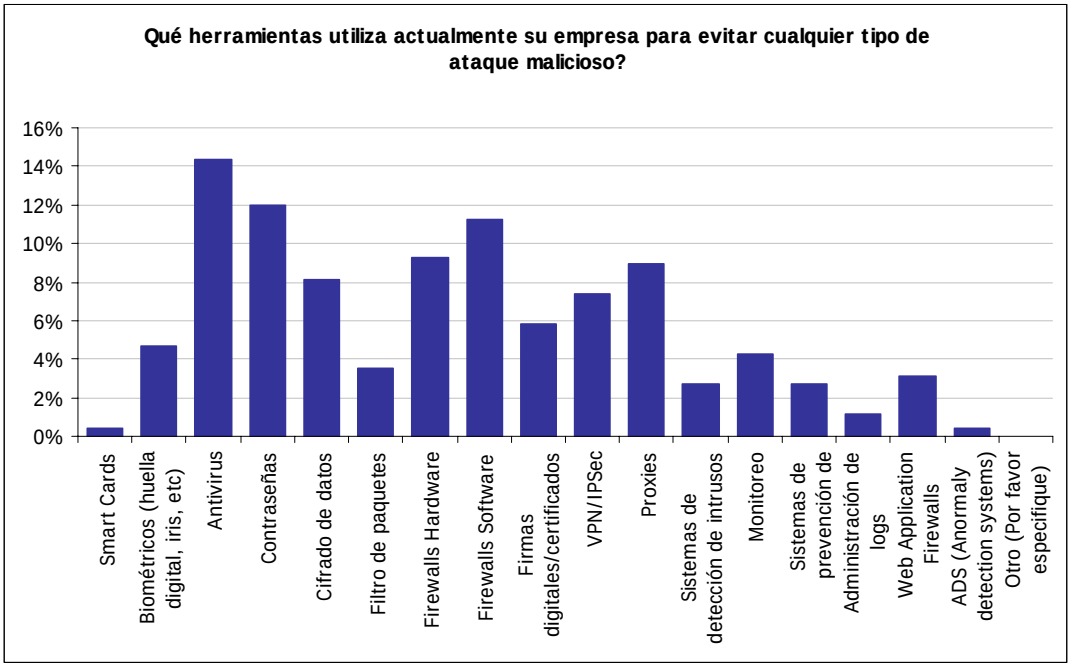

Fuente: elaboración propia.

### 4.2.3. Herramientas y buenas prácticas de seguridad informática

- Propósito: identificar las diferentes herramientas y buenas prácticas en el manejo de la seguridad informática, así como las diferentes metodologías utilizadas durante el desarrollo de *software*.

- Análisis de resultados: luego de determinar el tipo de ataque y cuánto tiempo tardan en recuperarse del mismo, viene la revisión de los métodos o herramientas que se utilizan para combatir estas amenazas. Anteriormente, se determinó que los virus son las amenazas más constantes, por lo que la herramienta que más se usa para combatirlos son los Anti-virus. Otras de las herramientas que más se utilizan son los *Firewalls* y los *Proxies*. Como buenas prácticas también se mencionan el uso de contraseñas u dispositivos biométricos, aunque en menor porcentaje.

Figura 15. **Herramientas para prevenir un ataque**

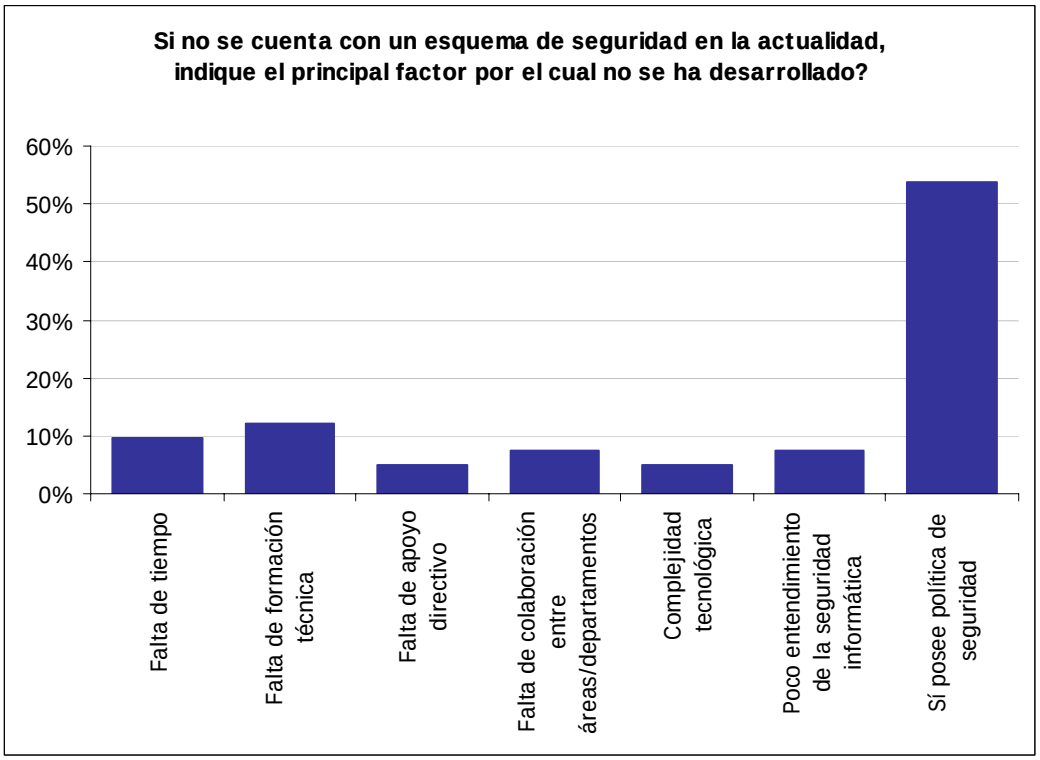



Fuente: elaboración propia.

Dentro de la encuesta se trató de determinar si las empresas poseen una política de seguridad o si existe alguna limitante en el desarrollo de la misma.

Se determinó que el mayor porcentaje de la muestra sí posee una política de seguridad implementada en su empresa. Las empresas que no cuentan con ella definieron como principales obstáculos la falta de tiempo y de capacidades técnicas.

Figura 16. **Factores para no contar con un esquema de seguridad**

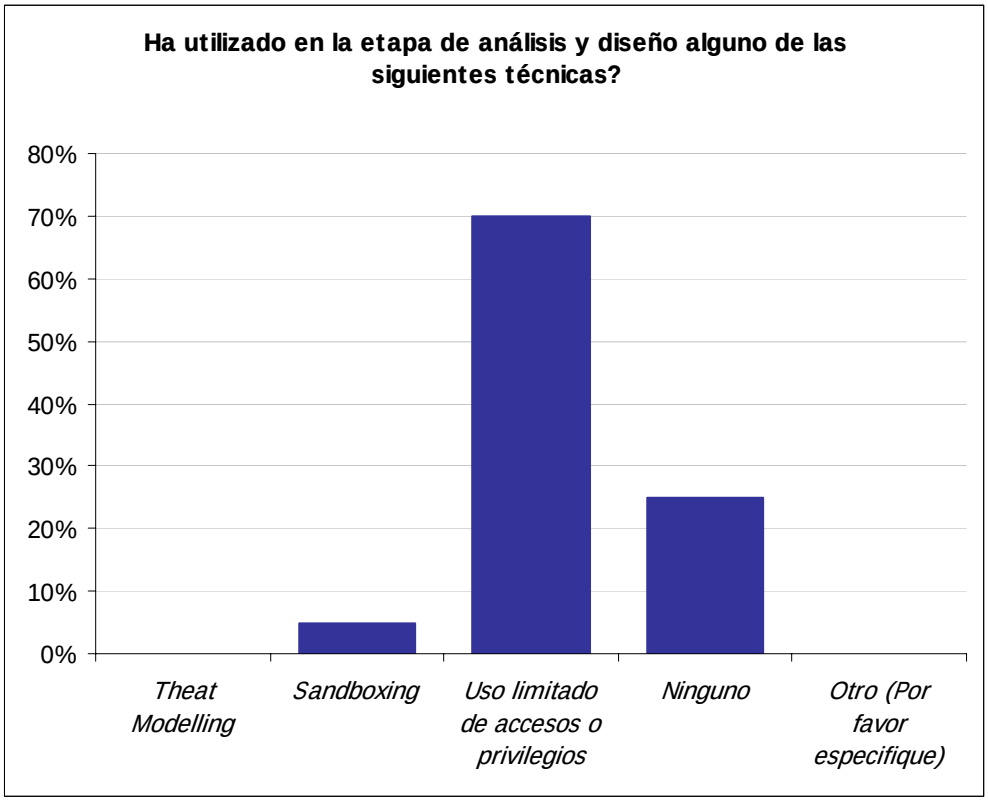



Fuente: elaboración propia.

En lo que respecta al análisis y diseño, el mayor porcentaje de empresas de la muestra utiliza la técnica de uso limitado de accesos.

En la actualidad una de las técnicas más utilizadas es el diagrama o modelo de amenazas, pero ninguna de las empresas de la muestra lo utiliza actualmente. Una muestra muy pequeña ha utilizado las cajas de arena o *Sandboxing,* y el 25% restante no utiliza ninguna.

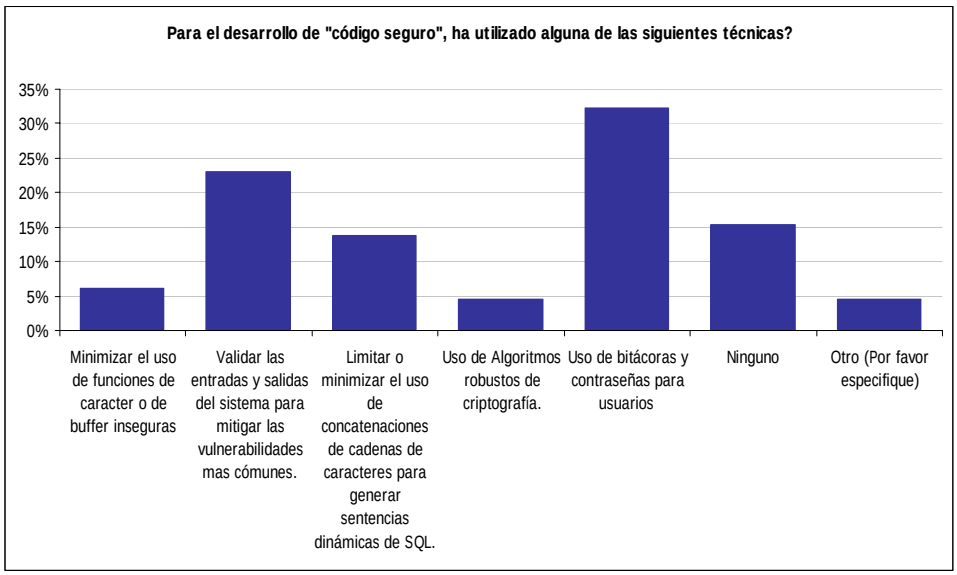
Figura 17. **Técnicas utilizadas durante el análisis y diseño**




Fuente: elaboración propia.

En el desarrollo no es ninguna sorpresa que la categoría más alta es utilizar o implementar una serie de bitácoras dentro de los diferentes sistemas. En segunda categoría se encuentra la validación de las entradas y salidas del sistema. Por último, como tercer categoría en importancia, se señala la eliminación de las concatenaciones de cadenas de caracteres para armar Querys dinámicas.

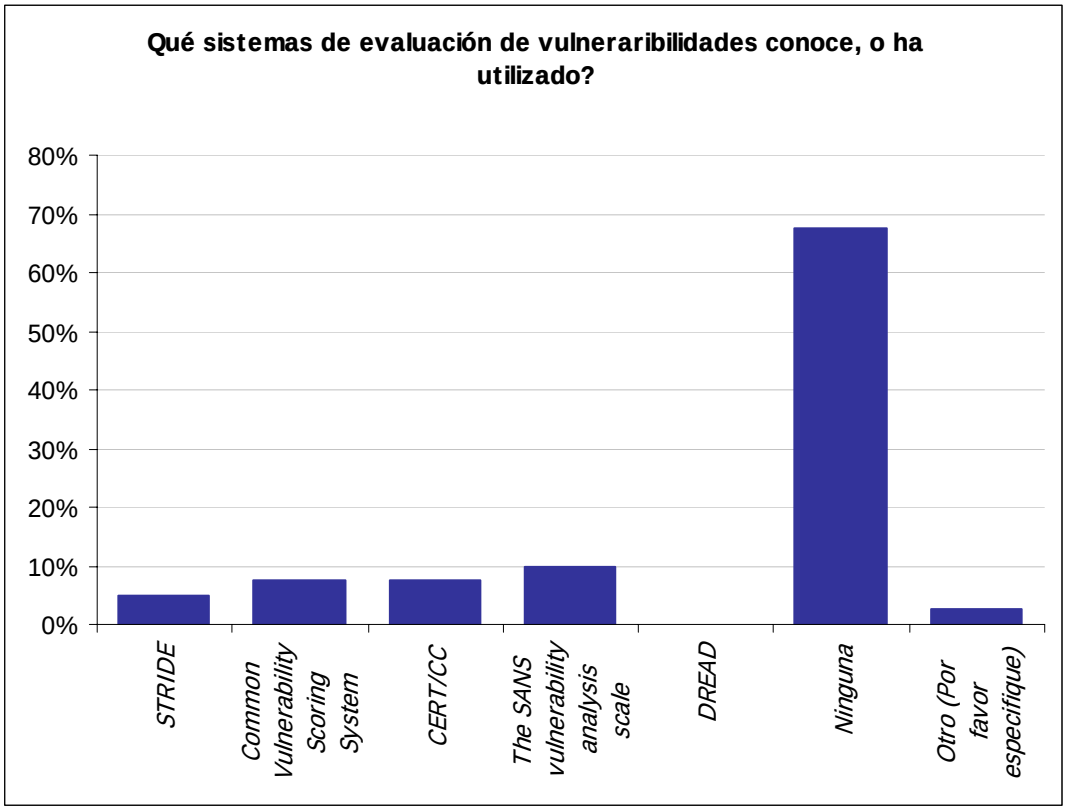
Figura 18. **Técnicas utilizadas para el desarrollo de código seguro**




Fuente: elaboración propia.

El último punto evaluado en la encuesta es la utilización de técnicas para evaluar las vulnerabilidades dentro de un sistema. Debido a lo nuevo del tema, se detectó que casi un 70% de la población evaluada no conoce ningún método de evaluación. Y solo en una muy pequeña medida conoce alguno de los mencionados.

Figura 19. **Sistemas de evaluación de vulnerabilidades**




Fuente: elaboración propia.

# CONCLUSIONES

1. Las organizaciones deben tener la capacidad de reconocerse vulnerables y como tales, buscar el apoyo técnico que les permita ser y sentirse menos inseguras; dicho apoyo consistirá en modelos que impulsen la aplicación de estándares y metodologías que mediante gobierno, administración y operación de los recursos participantes y el capital intelectual dispuesto para ello, logren mantener estables los procesos, salvaguardar los recursos y por ende, garantizar la continuidad del negocio.

2. La diversidad de estándares que se ofrecen actualmente, permite a los responsables de la seguridad informática pasar de una protección mínima o nula a una protección de verificación; actividad que ya es llevada a cabo por la mayoría de empresas, esto es, el interés y necesidad de normar procesos internos, y de considerar los procesos de TI con mejores niveles de calidad.

3. Las estadísticas muestran un crecimiento acelerado en los ataques a sistemas; para contrarrestarlos se debe de utilizar todos los métodos y herramientas que se tengan a mano. Los escáneres de vulnerabilidades son una herramienta disponible para garantizar seguridad; deben de utilizarse en complemento con *firewalls*, herramientas de detección de intrusos, buenas políticas de seguridad y otras prácticas que lo ameriten. Toda esta labor debe venir complementada por un buen plan de corrección de todas las vulnerabilidades detectadas.

4. Dentro del ambiente guatemalteco, existe muy poco conocimiento sobre los diferentes estándares internacionales de seguridad que deben aplicarse en todo el proceso del desarrollo de *software* seguro, desde el inicio con la etapa de análisis, pasando por la etapa de desarrollo y finalizando con el testeo de los mismos antes de salir a producción.

5. La evaluación de vulnerabilidades, es hasta cierto punto muy subjetiva, debido a que se basa en los diferentes puntos de vista de las personas que participan en ella; pero ayuda de manera significativa a la detección, revisión y evaluación de los diferentes puntos que pueden convertirse en amenazas para el sistema evaluado.

6. Las diferentes prácticas utilizadas en la actualidad, ofrecen una gama bastante amplia de caminos a seguir, pero será cada empresa la responsable de adoptar aquellas que más le convengan; por otro lado, podrán hacer las adaptaciones o modificaciones que consideren necesarias para alcanzar el grado de seguridad deseado.

# RECOMENDACIONES

1. Cada una de las técnicas presentadas en el desarrollo de sistemas posee sus propias características, asi como sus ventajas y desventajas, antes de optar por cualquiera técnica, el departamento o grupo encargado de la seguridad informática debe evaluarlo y ver si cumple con los estándares mínimos que se tienen planteados.

2. Una correcta definición de un modelo de amenazas, durante el diseño de la aplicación puede ayudar a reducir de manera significativa, el tiempo invertido en tener que modificar una aplicación que ya está en funcionamiento y que sufrió de alguna intrusión.

3. La priorización durante el proceso de corrección de las vulnerabilidades detectadas, debe hacerse tomando en cuenta todos los aspectos vistos, desde la probabilidad que una vulnerabilidad pueda llegarse a activar, hasta el daño que la misma pueda causar.

4. Dentro de la evaluación o testeo de aplicaciones antes de salir a producción, debe incluirse pruebas que determinen la robustez del sistema contra ataques maliciosos. Se podrán aplicar pruebas de penetración o pruebas de *Fuzzing*, y preferentemente que sean realizadas por personas con experiencia en el tema.

# BIBLIOGRAFÍA


1. Advanced Research Corporation ARC*. Security Auditors Research assistant SARA* [en línea]. *Herramienta detección de vulnerabilidades.* [ref. de 5 de abril 2011]. Disponible en Web: <http://www.www-arc.com/sara>.

2. Agile Modeling. *Introducción a los modelos de Amenazas*. [en línea]. [ref. de 15 de marzo 2011]. Disponible en Web: <http://www.agilemodeling.com/artifacts/securityThreatModel .htm>.

3. BAIER, Dominick. *Principio de menos privilegios* [en línea]. [ref. de 22 de enero de 2011]. Disponible en Web: <http://www.leastprivilege.com>.

4. Cert coordination center. *Mejoramiento de Seguridad* [en línea]. [ref. de 15 de enero 2011]. Disponible en Web: <http://www.cert.org/archive/pdf/SecurityMeasurement andAnalysis.pdf>.

5. FRIEDL, Steve. *SQL injection* [en línea]. [ref. de 25 de enero de 2011]. Disponible en Web: <http://www.unixwiz.net/techtips/sql-injection.html>.

6. Microsoft. *Guía Modelo de Amenazas* [en línea]. [ref. de 10 de enero 2011]. Disponible en Web: <http://www.microsoft.com/technet/SolutionAccelerators>.

7. Scribd. *Algoritmos de Criptografía*. [en linea]. [ref. de 15 de diciembre de 2010]. Disponible en Web: <http://es.scribd.com/doc/7448828/Algoritmos-de-Criptografia>.

8. SINGH, Amit. *Sandboxing.* [en línea]. [ref. de 12 de febrero de 2011]. Disponible en Web: <http//www.kernelthread.com/publications/security/ sandboxing.html>.

9. SUN Developer Network. *Logging and traccing.* [en línea]. [ref. de 20 de enero de 2011]. Disponible en Web: <http://developers.sun.com/solaris/articles/logging.html>.

10. SUTTON, Michael. *Fuzz testing: Brute force vulnerability discovery*. [en línea]. [ref. de 2 de mayo de 2011]. Disponible en Web: <http://www.fuzzing.org>.

11. The Mitre Corporation. *Vulnerabilidades más comunes* [en línea]. [ref. de 22 de febrero 2011]. Disponible en Web: <http://cve.mitre.org/news>.

12. The Software Assurance Forum for Excellence in Code (SAFECode). *Prácticas Fundamentales para el desarrollo de Software Seguro* [en línea]. [ref. de 20 de marzo 2011]. Disponible en Web: <http://www.safecode.org/publications/SAFECode_Dev_ Practices0211.pdf>.

13. Wikipedia. *Concepto Sandboxing.* [en línea]. [ref. de 30 de marzo de 2011]. Disponible en Web: <http://en.wikipedia.org/wiki/Sandbox_%28computer_security%29>.