CIMAT Unidad Zacatecas

Análisis de Seguridad Web para Prevenir Cross-Site Scripting (XSS) en Aplicaciones PHP

REPORTE DE INVESTIGACIÓN

I.S.C. José Enrique Miramontes Medrano

Temas de investigación:

- Seguridad en Aplicaciones Web
- Cross-Site Scripting (XSS)
- Seguridad en PHP

Lugar y Fecha
Zacatecas, Zac., julio de 2011

Contenido pag.

Índice de Tablas

Índice de Figuras

I. INTRODUCCIÓN

Las aplicaciones web son aquellas aplicaciones que los usuarios pueden utilizar accediendo a un servidor web a través de Internet mediante un navegador que interpreta el código independientemente del sistema operativo (Wikipedia, 2011), existen aplicaciones como webmails, tiendas en línea, blogs, chats, etc., en la actualidad la mayoría de las empresas cuentan con aplicaciones web donde promocionan sus productos, muestran información acerca de los servicios que ofrecen, realizan compra/venta de productos, bienes y servicios. Para realizar sus funciones estos sitios deben interactuar con el usuario para poder hacer el servicio más dinámico y ser una ventaja competitiva ante otras empresas. Por lo tanto, la web está poblada de aplicaciones para todo tipo de negocios y día con día se desarrollan más y más que se integran a la red y quedan a disposición de las personas que las soliciten.

Todo parece marchar correctamente, pero existen en el mundo de las computadoras personas capaces de utilizar cualquier tecnología para causar daño o para obtener beneficios personales a través de la explotación de los errores de las aplicaciones que hay en Internet. En la actualidad existen diferentes tipos de vulnerabilidades de seguridad que son comunes para las aplicaciones web y cada una de ellas representa una oportunidad para personas con intenciones de robar información confidencial, destruir el sitio web, incluso robar dinero de las cuentas bancarias. Es por eso que la seguridad en las aplicaciones web no debe tomarse a la ligera e integrar las mejores prácticas en el desarrollo de software para evitar este tipo de defecto en nuestras aplicaciones.

Si las grandes compañías que invierten en investigaciones de seguridad de sus aplicaciones han sido víctimas de ataques a través de las vulnerabilidades más comunes como XSS y SQL Injection (WASC, 2011), que se esperaría de pequeñas empresas o desarrolladores independientes que no tienen tantos recursos como para invertir en seguridad.

De estas vulnerabilidades las más comunes y que han existido desde los primeros años en que se creó la Web son en primer lugar, la inyección de comandos SQL que debe su posición al impacto tan grave que representa este ataque ya que accede directamente a los datos almacenados de la aplicación (WASC, 2011), y en segundo lugar se encuentra la inserción de secuencias de comandos mejor conocido como Cross-Site Scripting (XSS) que es de menor impacto pero sus consecuencias como la deformación de sitios web y la inserción de Malware están en primer lugar según la base de datos de incidentes en aplicaciones web de los años 2007 al 2011 recolectados por la Web Application Security Consortium durante estos últimos 5 años.

Es entonces, Cross-Site Scripting (XSS) el tema que tratará este estudio, ya que siempre se le ha dado mayor importancia a la inyección SQL por obvias razones, pero Cross-Site Scripting ha permanecido presente y sus efectos son mucho más diversos, desde simplemente insertar una imagen en un sitio web hasta suplantar sesiones y el robo de credenciales de autentificación, que si

son bien utilizados su impacto también puede llegar a ser muy grave. Este estudio presenta los conceptos principales de XSS, las formas de ataque, las razones por las que sigue presente en las aplicaciones web, los efectos que causa, así como las mejores prácticas para evitar este tipo de ataques en aplicaciones web desarrolladas en PHP.

Cabe mencionar que las mejores prácticas recolectadas, así como la lista de verificación propuesta en este estudio no son las únicas formas de evitar Cross-Site Scripting, también existen los Sistemas de Detección de Intrusos (Kemmerer 2002, Wikipedia 2011) que ayudan a detectar accesos no autorizados a un equipo de computo o a una red a través del análisis de comportamiento y tipo de tráfico en la red, el cual puede ayudar a minimizar ataques de XSS, pero no serán tratados en este estudio ya que este tema merece una investigación más completa y el objetivo de este estudio es encontrar las mejores prácticas para evitar XSS en aplicaciones web desarrolladas en PHP.

Primeramente se describirá el tema y los motivos que llevaron a seleccionar el tema, después la metodología que se utilizó para recolectar los artículos que se leyeron para este estudio, así como las dificultades que se presentaron. En el tema II. Estado del Arte, se muestra el resumen de los artículos leídos para este estudio que componen el estado del arte y desarrollo principal de los temas; por último en el tema III. Conclusiones, se presentan las conclusiones a las que se llegó en el estudio, así como también las actividades que complementarían el estudio y proyectos que se relacionan con esta investigación.

I.1. DESCRIPCIÓN DEL TEMA SELECCIONADO

La seguridad en el área de computación es a veces considerada como un simple asunto de mantener los datos privados como privados. Eso es parte del concepto, quizá la parte más importante, pero también hay otros aspectos en el corazón de la seguridad de computadoras (Snyder, 2010):

- Secretos: Toda información propietaria como contraseñas y claves que dan acceso a los recursos del sistema, seguridad es con respecto a mantener esa información fuera de las manos equivocadas.

- Recursos escasos: Cada computadora tiene un límite de ciclos de procesamiento por segundo, un límite de memoria, un límite de disco duro, un límite de ancho de banda, entonces seguridad es también el prevenir el agotamiento de esos recursos, ya sea accidental o intencional, la necesidad de legitimar a los usuarios puede ser cumplida.

- Buena relación en la red: En un mundo en red, cada programador y administrador del sistema tienen la responsabilidad de asegurar que su sistema es libre de cualquier explotación accidental o maliciosa que pudiera comprometer otros sistemas en la red. Su reputación como buen ciudadano de la red depende entonces de la seguridad de sus sistemas.

Las aplicaciones web han tenido un gran auge sobre todo para la economía mundial (Grossman, 2007), eh ahí la importancia de las compañías que manejan sitios web el garantizar la confidencialidad y seguridad de la información de sus clientes, ya que un mal paso en la seguridad de la aplicación puede ser un paso fatal para la compañía, es entonces cuando la seguridad en aplicaciones web se vuelve un tema de mucha seriedad.

PHP ha pasado de ser un conjunto de herramientas personales para el desarrollo de páginas web a el lenguaje de programación web más popular; y junto con esa transición trajo consigo nuevas preocupaciones tales como el desempeño, mantenibilidad, escalabilidad, confiabilidad y también muy importante seguridad (Shiflett, 2005).

Los desarrolladores hoy en día se enfocan primordialmente en la funcionalidad de las aplicaciones dejando de lado aspectos tan importantes como la seguridad (Egeberg, 2008). El problema no es la falta de información porque una simple búsqueda en internet sobre seguridad en aplicaciones web muestra cientos de resultados sobre el tema, tampoco lo es la falta de herramientas que ayuden a realizar más fácil las tareas de detección de vulnerabilidades; al parecer el problema sigue siendo que los desarrolladores subestiman los efectos de las vulnerabilidades (Bodmer, 2007) como Cross-Site Scripting y SQL Injection que durante los últimos cinco años han estado presentes en las fallas de seguridad de las aplicaciones web en todo el mundo.

Afortunadamente PHP es un lenguaje extremadamente poderoso y muy adecuado para proveer seguridad (Snyder, 2010) y es precisamente el centro de interés de este estudio, el obtener las mejores prácticas en PHP para evitar dentro de lo mejor posible brechas de seguridad en nuestras aplicaciones web.

I.2. MOTIVACIÓN E INTERÉS EN LA SELECCIÓN DEL TEMA

El interés por elegir investigar sobre la seguridad en aplicaciones web y sobre todo por la vulnerabilidad de Cross-Site Scripting surgió cuando al equipo de la tercera generación de CIMAT – al cual pertenezco – se le encomendó la tarea de realizar una auditoría de seguridad web a un sistema de la empresa SoftLogik. Durante el proceso de auditoría a este sistema se adquirieron conocimientos sobre aspectos de seguridad de los cuales ni siquiera se sabía de su existencia, así como herramientas en las cuales apoyarse para detectar brechas de seguridad en los sistemas.

Es por lo anterior, que surgió aún más el interés por continuar investigando el tema de seguridad en aplicaciones web, y sobre todo en la vulnerabilidad de Cross-Site Scripting ya que es una de las más comunes y que a pesar de que es fácil evitarla y hay bastante información sobre el tema en Internet, sigue siendo una vulnerabilidad todavía presente en la mayoría de las aplicaciones web hoy en día. Así que el interés por descubrir qué factores influyen en la permanencia de esta vulnerabilidad, así como también el saber cuáles son las mejores prácticas para evitar este defecto de seguridad en las

aplicaciones web, son algunos de los objetivos principales por el cual se realizó esta investigación.

I.3. METODOLOGÍA DE CLASIFICACIÓN

Los artículos leídos para este reporte fueron encontrados a través de la biblioteca virtual de CIMAT usando principalmente las bases de datos de ACM y SpringerLink.

También se hizo una búsqueda en Internet utilizando el buscador Google.

Las palabras claves utilizadas para la búsqueda de los artículos fueron:

- Seguridad en Aplicaciones Web
- Cross-Site Scripting (XSS)
- Seguridad en PHP

En la mayoría de los artículos leídos se encontraron conceptos básicos sobre Cross-Site Scripting (XSS), pero contenían pocos ejemplos que son los que ayudan a comprender mejor la teoría presentada en cada artículo, además de que cada artículo mostraba soluciones diferentes al problema de XSS. Sin embargo el libro de "XSS Attacks: Cross Site Scripting Exploits and Defense" si contiene variedad de ejemplos y explicaciones sencillas el cual servirá como guía en la recopilación de las mejores prácticas para evitar el XSS en aplicaciones web desarrolladas con PHP.

I.4. DIFICULTADES

Las principales dificultades que se presentaron es que se encontró una gran variedad de artículos que contenían las palabras claves usadas para la búsqueda, pero casi todos tenían los mismos conceptos de XSS y mostraban ejemplos muy sencillos, que igual eran fáciles de entender pero nada elaborados de forma que mostraran realmente un daño serio a una aplicación web.

Otra dificultad fue que algunos de los artículos que mostraban estadísticas sobre las vulnerabilidades en aplicaciones web, no señalaban de manera clara como se obtuvieron los datos para las estadísticas, ya que los artículos que contenían tales gráficas y estadísticas pertenecían a empresas que desarrollan un escáner automatizado que detecta vulnerabilidades en los sitios web, por lo tanto no mostraban la información completa que es necesaria para justificar la importancia de la vulnerabilidad XSS.

II. ESTADO DEL ARTE

II.1. Seguridad en Aplicaciones Web

La seguridad en cualquier aplicación software se considera como una medida, no cómo una característica; ya que la seguridad es subjetiva tanto cómo decir si algo está caliente o no (PHP Security, 2005). No se puede decir que una aplicación es segura, incluso si se dijera algún porcentaje o un nivel, este de igual manera seguiría siendo subjetivo.

Gary McGraw apunta un dilema esencial para la seguridad. “No existe tal cosa como 100 por ciento seguro. De hecho, hay una tensión fundamental hoy en día entre tecnología funcional y seguridad” (Gilliam, 2003).

La seguridad en aplicaciones Web es un caso especial, ya que al estar alojadas en servidores que son accedidos desde cualquier parte del mundo por cualquier persona la seguridad debe tomarse muy en serio; y más aún, si se sabe que la Web es el patio de recreo de más de 800 millones de cibernautas, hogar de más de 100 millones de sitios web y transporta billones de dólares cada día. La economía internacional ha empezado a ser dependiente de la Web como un fenómeno global. Hoy en día, los usuarios mantienen en la Web sus nombres, direcciones, número de seguridad social, números de tarjetas de crédito, números de teléfono, el nombre de sus madres, salario, fecha de nacimiento, estados financieros, etc. Y también, hoy en día ocho de cada diez sitios web tienen graves problemas de seguridad que ponen en riesgo toda esta información. Incluso los sistemas más seguros están plagados de amenazas de seguridad (Grossman, 2007).

De la noche a la mañana, los sitios web pasaron de ser simples paredes de texto a comercios electrónicos globales, donde la seguridad seguía siendo pobre, y los administradores no encontraban forma de proteger sus aplicaciones web. El alcance y magnitud del software ha incrementado exponencialmente y los problemas de seguridad también se han complicado más. Ahora cientos de millones de usuarios tienen acceso directo al software y cualquier número de ellos puede ser un adversario malicioso. Términos como Cross-Site Scripting (XSS), SQL Injection y una docena más de ataques basados en web tienen que ser entendidos y tratados (Grossman, 2007).

Una de las mayores amenazas que los desarrolladores de aplicaciones web deben entender y aprender a mitigar son los ataques XSS. Mientras XSS es solo una pequeña parte de campo de seguridad en aplicaciones web es quizá el más peligroso. Un simple bug en una aplicación web puede resultar en un navegador comprometido a través del cual un atacante puede robar información (Grossman, 2007).

II.2. ¿Qué es Cross-Site Scripting XSS?

Las vulnerabilidades XSS datan desde 1996 durante los primeros años de la World Wide Web (Web). Al mismo tiempo que los e-commerce empezaron a tener éxito. Cuando miles de páginas estaban bajo construcción y usaban Hypertext Markup Language (HTML). El lenguaje de programación JavaScript apareció en la escena, un conocido precursor del cross-site scripting el cual cambió el paisaje de la seguridad en aplicaciones web para siempre. JavaScript permitía a los desarrolladores crear sitios web interactivos llenos de efectos impresionantes. Pero los crackers descubrieron un nuevo mundo de posibilidades (Grossman, 2007)

Los crackers encontraron que cuando usuarios desprevenidos visitaban sus páginas web podían cargar a la fuerza cualquier otro sitio web en un frame HTML dentro de la misma ventana del navegador. Usando JavaScript ellos podía cruzar el límite entre los dos sitios web y leer desde un frame dentro de otro. Ellos fueron capaces de robar nombres de usuario y contraseñas de los formularios HTML, robar cookies, y comprometer cualquier información confidencial desplegada en la pantalla (Grossman, 2007).

En 1999 David Ross responsable de la seguridad en Internet Explorer de Microsoft, encontró estas vulnerabilidades en el modelo de seguridad de Explorer que efectivamente permitían insertar scripts, se reportó estas vulnerabilidades en la industria y después de una reunión con otros proveedores de software se eligió el nombre de “Cross-Site Scripting” (Grossman, 2007), que se abrevia XSS ya que CSS puede ser confundido con Hojas de Estilo en Cascada y se usa la palabra scripting ya que esta vulnerabilidad permite ejecutar código de Visual Basic Script, HTML, JavaScript, ActionScript e incluso código PHP (Wikipedia, 2011).

Las Fallas XSS ocurren cuando una aplicación toma datos externos no confiables y los muestra en el navegador sin antes haberlos validado o codificado apropiadamente. Los atacantes pueden ser usuarios tanto externos como internos que puedan insertar datos no confiables al sistema (OWASP, 2010). Dicho de otra forma XSS permite inyección de código en el lado del cliente de una aplicación web (Cgisecurity, 2011).

Existen dos tipos de Cross-Site Scripting (OWASP, 2008):

- Reflejado: También conocido como tipo 1 o no persistentes y son los ataques de XSS más frecuentes en la actualidad. El atacante utiliza ingeniería social para convencer a su víctima de cargar un URL malicioso en su navegador.
  - Ejemplo: http://www.home.com?usr=<script>alert(123)</script>

- Almacenado: Es el tipo más peligroso de XSS, ya que la aplicación permite almacenar la entrada del usuario en un repositorio para su uso posterior. Como consecuencia la información aparecerá como parte del sitio y se ejecutará en el navegador del usuario con los privilegios de la aplicación.

II.3. ¿Qué puede hacer un ataque XSS?

Los ataques con XSS son muy frecuentes, pero sus consecuencias son subestimadas por los desarrolladores (Bodmer, 2007). XSS puede ser usado para robar información sensible, secuestrar sesiones de usuario, comprometer el navegador y afectar la integridad del sistema (Grossman, 2007). Así como también modificar la aplicación (deformarla), instalar código malicioso y redirigir a los usuarios a otros sitios maliciosos (OWASP, 2010).

La forma de ataque XSS más común es a través de los campos de un formulario HTML, aunque también es común que un atacante use ingeniería social a través de correos o mensajería instantánea para orillar a los usuarios a hacer clic en ligas que contienen scripts malignos y ejecutar acciones que envíen datos confidenciales al servidor del atacante (Bodmer, 2007).

El atacante puede modificar los valores que se pasan entre páginas mediante URL, agregando un script que ejecute acciones en el navegador con los permisos del usuario sin siquiera saber la contraseña, o puede ser redirigido a un sitio que le resulte familiar y pedirle que vuelva a iniciar sesión para poder robar su nombre de usuario y contraseña, a esta acción también se le conoce como “phishing” (Wikipedia, 2011).

II.4. Estadísticas de vulnerabilidades y ataques en aplicaciones Web

Existe un proyecto llamado OWASP - Top 10 cuyo propósito es crear conciencia sobre la seguridad en las aplicaciones web. Este proyecto recopila las 10 vulnerabilidades más riesgosas para aplicaciones web y las actualiza cada año, también en el sitio de la fundación OWASP se pueden encontrar diversos recursos referentes a la seguridad que ayudan a las empresas y desarrolladores a desarrollar, comprar y mantener aplicaciones web seguras (OWASP, 2010).

Al igual que OWASP, la Web Hacking Incident Database (WHID) es un proyecto dedicado a mantener una lista de incidentes de seguridad relacionados con aplicaciones web. Su objetivo es servir como herramienta para hacer conciencia sobre los problemas de seguridad de aplicaciones web y proveer información estadística de los incidentes (WASC, 2011).

La Tabla 1. Métodos de Ataque, muestra el porcentaje que ocupó en cada año desde 2007 a 2011 cada uno de los diferentes métodos de ataque a aplicaciones web, destacando en primer lugar la inyección SQL, seguida de insuficiencia de anti-automatización y después Cross-Site Scripting. Cabe mencionar que estos datos son únicamente de ataques registrados ya sea por la empresa que sufrió el ataque o por la persona que llevo a cabo el ataque, obviamente no se están mostrando en los datos todos los demás ataques que las empresas no reportaron para no dañar su reputación o que los atacantes no reportaron para no dejar pistas sobre su identidad.

Método de Ataque	2007	2008	2009	2010	2011	Promedio
	%	%	%	%	%	%
Cross-Site Scripting (XSS)	**12**	**8**	**8**	**22**	**11**	**12**
SQL Injection	20	30	19	15	21	**21**
Cross-Site Request Forgery (CSRF)	0	3	8	8	4	**6**
Insufficient Authentication	18	3	11	13	5	**10**
Configuration/Admin Error	17	5	8	7	3	**8**
Insufficient Anti-Automation			10	15	32	**19**
Unknown	15	29	11	10	8	**15**
Other	18	22	13	10	6	**14**

Tabla 1. Métodos de Ataque (WASC, 2011).

La Figura 2. Gráfica de los Métodos de Ataque muestra las tendencias de los tres principales métodos de ataque, donde se puede apreciar que la insuficiencia de anti-automatización ha ido aumentando conforme los años, Cross-Site Scripting ha tendido a aumentar pero muy poco y la inyección SQL ha ido disminuyendo también solo un poco. Lo que da a entender que si la tendencia sigue así, ahora el problema de negación de servicio estará más presente en los siguientes años y XSS e inyección SQL seguirán presentes pero en menor número.

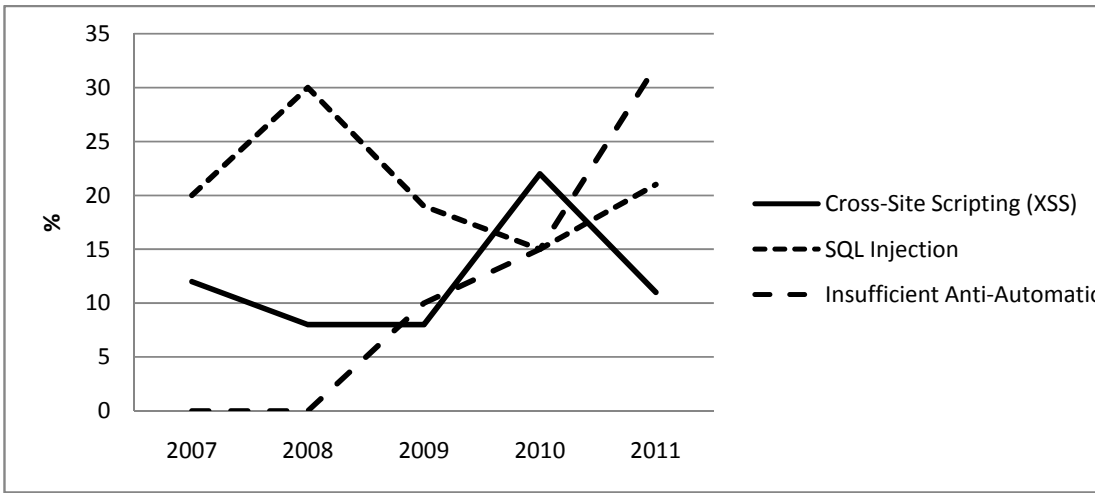

Figura 1. Gráfica de los Métodos de Ataque

La Tabla 2. Objetivos del Ataque muestra los porcentajes de lo que han sido los objetivos de los ataques de los años 2007 al 2011 destacando en primer lugar la deformación y malware que son principalmente efectos del XSS y en segundo lugar el robo de información sensible que son efectos tanto de inyección SQL como de XSS.

Objetivos del Ataque	2007	2008	2009	2010	2011	Promedio
	%	%	%	%	%	%
Robo de información y datos Sensibles	42	19	26	24	13	**25**
Deformación y Malware	35	30	28	33	24	**30**
Desinformación	1	1	19	5	9	**9**
Negación del servicio	3	8	4	12	33	**12**
Robo de dinero	3	13	11	12	4	**9**
Spam	3	8	4	2	2	**4**
Phishing	1	5	2	1	5	**3**
Otros	8	15	10	10	8	**10**

Tabla 2. Objetivos del Ataque (WASC, 2011).

Como era de esperarse la Figura 2. Gráfica de los Objetivos del Ataque muestra que negación del servicio que es un efecto de la falta o insuficiente anti-automatización ha ido en aumento estos últimos cinco años, aunque el robo de información y la deformación y malware han disminuido poco siguen siendo en promedio los objetivos más comunes de ataques.

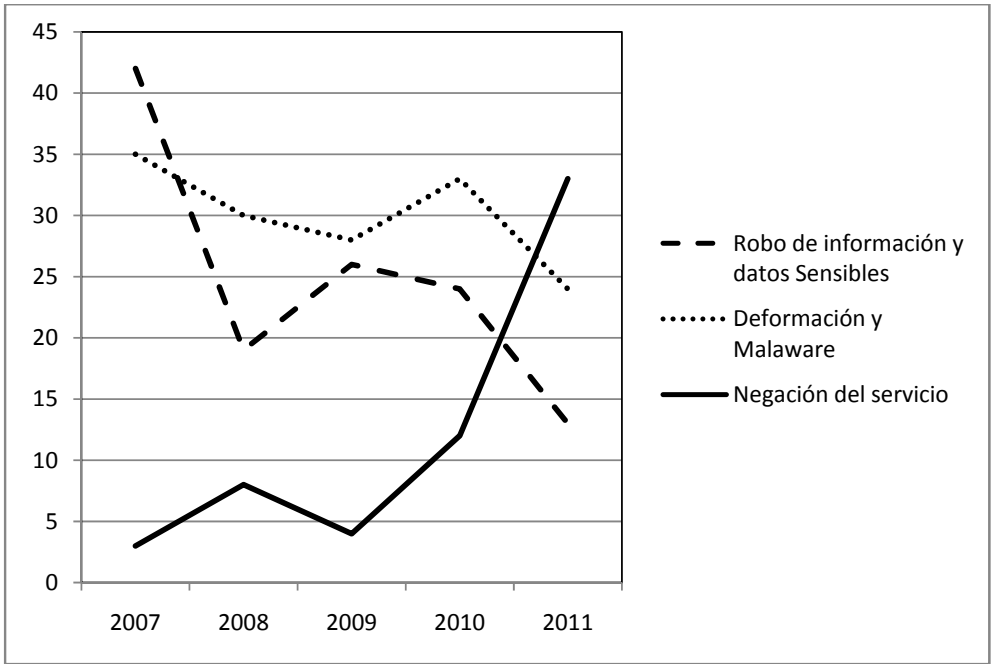

Figura 2. Gráfica de los Objetivos de Ataque

II.5. Casos reales de XSS

En octubre del 2005, el gusano Sammy, el primer gusano XSS de importancia obligó a MySpace a apagar sus servidores. La carga que generaba en el servidor era relativamente benigna, este gusano fue diseñado para esparcirse a través de los perfiles de los usuarios hasta llegar a infectar a más de un millón de usuarios en solo 24 horas (Grossman, 2007).

El 11 de agosto de 2007, un blog publicó el código del índice principal de Facebook obtenido a través de un ataque XSS, aunque no se dijeron los detalles (Egeberg, 2008).

El 20 de diciembre del 2008, el sitio web de American Express fue expuesto al encontrarse bugs de seguridad que pudieron haber permitido el acceso a la base de datos de los clientes y robado sus credenciales de acceso. El error de XSS continuó activo durante 2 semanas (Goodin, 2008).

En el sitio web de PayPal fue detectado en febrero de 2009 un bug de XSS el cual pudo ser usado para robar contraseñas de los usuarios y cookies autentificadas (Goodin, 2009).

30 de marzo del 2011, se descubrieron brechas de seguridad XSS en varios sitios de McAfee. 13 de octubre del 2010, la central de ventas de Amazon ha reportado una vulnerabilidad grave de XSS. 6 de octubre de 2010, PayPal vulnerable de XSS. 6 de octubre de 2010, se reportó que eBay tiene un XSS persistente que inyecta un iframe que roba las credenciales de los usuarios, etc., etc. (Xssed, 2011), y así continúa la lista de muchos sitios en los cuales se ha encontrado vulnerabilidades XSS.

En abril de 2011 Facebook fue víctima de 3 ataques con XSS que estuvieron activos durante un periodo de 10 días (Theharmonyguy, 2011):

- El primero fue solo una redirección que realizó un blogger para demostrar la vulnerabilidad de la aplicación de Facebook para móviles.
- El segundo ataque introducía una forma en los muros de los usuarios y se reproducía de forma viral con solo un clic pero igualmente no hacía ningún daño.
- El tercero introducía un video a través del código de las aplicaciones que permite Facebook, pero si identificaba que aún el usuario no estaba logueado cargaba una página falsa de inicio de sesión muy parecida a la de Facebook y mandaba nombre de usuario y contraseña a un log en el servidor del atacante.

II.6. Caso Práctico: Sistema PREP

La empresa desarrolladora de software SoftLogik, construyó un sistema denominado PREP (Programa de Resultados Electorales Preliminares) usando el Framework Yii de desarrollo en PHP, e identificó que como el sistema sería implementado en Internet y usado por al menos 138 personas para una elección municipal y hasta más de 130 000 personas en una elección federal para registrar los votos de las actas el día de la elección y poder calcular un resultado preliminar de los resultados de los votos antes que el IFE, era necesario llevar a cabo una auditoría de seguridad para corregir cualquier vulnerabilidad de la aplicación antes de montarla en un ambiente de producción ya que cualquier modificación no autorizada a los datos del PREP afectaría gravemente la veracidad de los resultados.

Con base en lo anterior, la tercera generación de la Maestría en Ingeniería de Software de CIMAT Zacatecas fue seleccionada para llevar a cabo la auditoría de seguridad al sistema PREP durante el mes de junio de 2011, la aplicación se encontraba en fase de desarrollo, presentando la mayoría de los módulos de captura de votos y generación de reportes por vía web, los cuales eran el objetivo principal de la auditoría de seguridad, ya que también el sistema contaría con registro de votos a través de telefonía.

Por sugerencia de la empresa SoftLogik, la auditoría de seguridad se realizó con base en el Top 10 de OWASP (Open Web Application Security Project) del 2010 que reporta las 10 vulnerabilidades más riesgosas y comunes en las aplicaciones web.

Top Ten de OWASP (OWASP, 2010):
1.- Inyección SQL
2.- Secuencia de comandos en sitios cruzados (XSS)
3.- Pérdida de autenticación y gestión de sesiones
4.- Referencia directa insegura a objetos
5.- Falsificación de peticiones en sitios cruzados (CSRF)
6.- Defectuosa configuración de seguridad
7.- Almacenamiento criptográfico inseguro
8.- Falla de restricción de acceso a URL
9.- Protección insuficiente en la capa de transporte
10.- Redirecciones y reenvíos no validados

De acuerdo al listado anterior la vulnerabilidad número dos, Secuencia de Comandos en Sitios Cruzados o Cross-Site Scripting por su nombre en inglés es la que se realizó para este caso práctico y la que se describe a continuación. La búsqueda de esta vulnerabilidad en el sistema se hizo primeramente de forma manual utilizando la hoja de trucos XSS que se encuentra en http://www.ha.ckers.org/xss.html la cual contiene más de cien formas diferentes de JavaScripts para saber si la aplicación web es vulnerable; la inserción exitosa de más de diez scripts en diferentes partes del sistema que contenía formularios HTML confirmó de manera definitiva que el sistema PREP presentaba fallas de seguridad XSS. Un ejemplo es mostrado en la Figura 3.

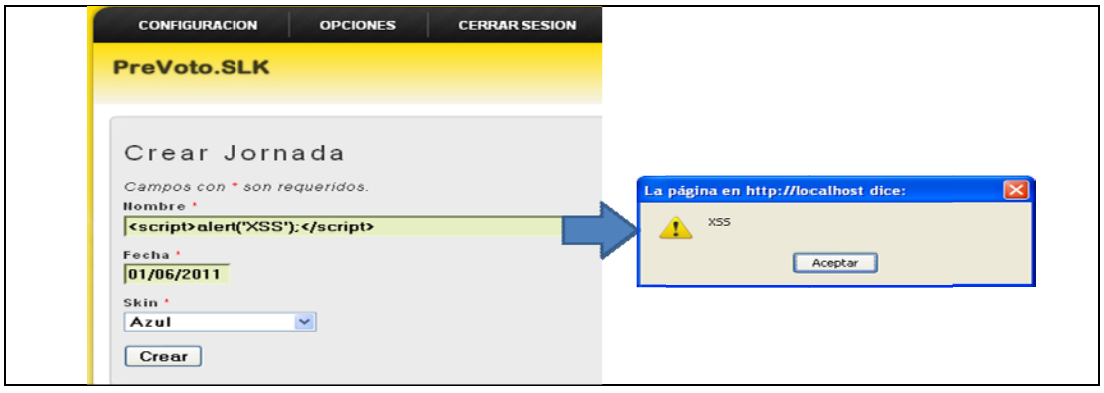

Figura 3. Inserción XSS exitosa en el PREP

El siguiente paso fue reportar la presencia de la vulnerabilidad al equipo de desarrollo del sistema PREP para que tomara las acciones necesarias para corregir esta brecha de seguridad; la cual fue corregida casi de manera inmediata al agregar un modulo de seguridad a las librerías de Yii y activando el módulo en cada una de las clases Controlador. La Figura 4 muestra el intento de ingresar código XSS después de las correcciones realizadas.

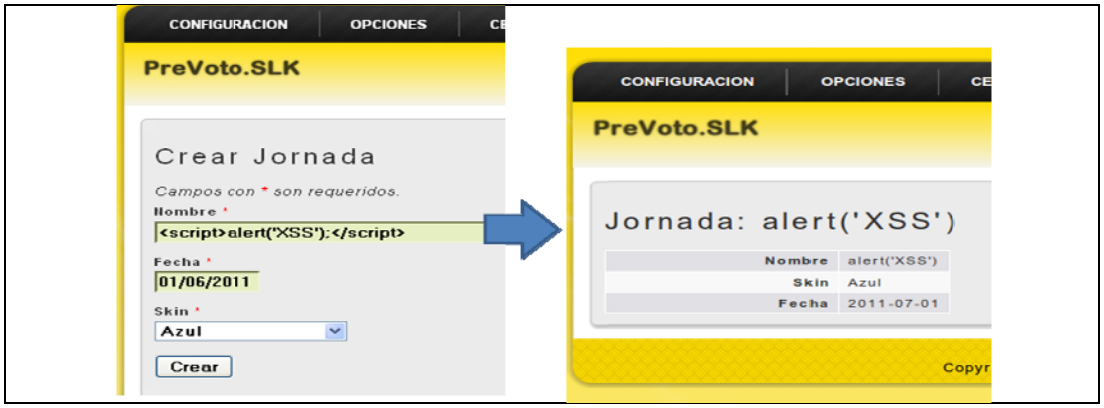

Figura 4. Inserción XSS fallida

Al reportarse que ya se había realizado la corrección de la vulnerabilidad, se intentó nuevamente de manera manual usando la hoja de trucos mencionada anteriormente y encontrando que ninguno de los scripts tuvo éxito en los formularios del PREP, lo cual indica que la vulnerabilidad fue cubierta satisfactoriamente.

No quedando satisfechos con los resultados el equipo de auditoría se dio a la tarea de buscar herramientas que realizaran de manera automática la búsqueda de vulnerabilidades XSS en el sistema, encontrando dos herramientas: Acunetix Web Vulnerability Scanner y XSS Me que fueron elegidas por la facilidad de uso y su popularidad; las cuales fueron ejecutadas en el sistema PREP y obteniendo como resultado que ninguna de las dos herramientas encontró vulnerabilidades XSS, lo cual sirvió para corroborar que el sistema estaba libre de esta falla de seguridad. Las Figuras 5 y 6 muestran el resultado arrojado por cada herramienta después de realizarse las correcciones al sistema.

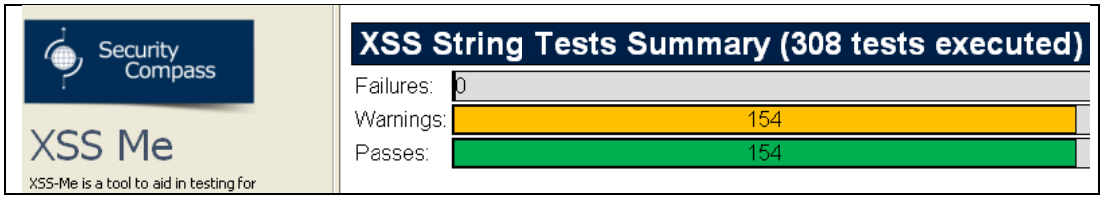

Figura 5. Resultados de la herramienta XSS Me

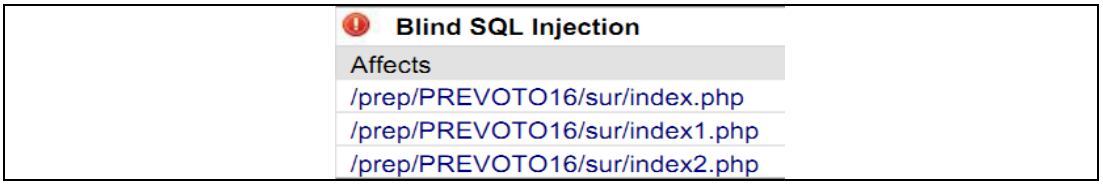

Figura 6. Resultados de la herramienta Acunetix

Las condiciones tanto de hardware como de software sobre las cuales estaba montado el sistema PREP son las siguientes:

- Sistema Operativo: Ubuntu Server 11.04 – Linux 2.6.38
- Procesador: 3.0 GHz
- Memoria RAM: 4 Gb
- Puerto de Acceso a la aplicación: Puerto 80
- Framework de Desarrollo: Yii Framework 1.6
- PHP 5.3.2
- Apache 2.0.64
- MySQL 5.0

Cabe destacar que tanto las pruebas como los resultados pueden variar con otras versiones de los programas, por lo que esta información se debe tener en cuenta al momento de aplicar las recomendaciones y el checklist presentados en este estudio.

II.7. Factores por los cuales las aplicaciones web continúan teniendo esta vulnerabilidad

- Los desarrolladores se preocupan más por poner Firewalls y antivirus o adquirir certificados SSL, pero ponen poca atención a XSS ya que creen que JavaScript no representa gran peligro para sus aplicaciones (Grossman, 2007).

- Los desarrolladores principiantes se enfocan en implementar la funcionalidad de la aplicación dejando de lado un aspecto importante como lo es la seguridad (Egeberg, 2008).

- Tanto los usuarios como los desarrolladores a veces no actualizan el software que usan, cuyas actualizaciones han reparado bugs encontrados en versiones anteriores (OWASP, 2008).

- Un ataque XSS es fácil de llevar a cabo, no se necesitan conocimientos avanzados en computación y redes, solo una web dinámica que permita a los usuarios introducir datos (Acunetix, 2011).

- Un desarrollador típico de PHP, trabajando en un proyecto típico, por lo general deja para el último tres cosas: Usabilidad, documentación y seguridad (Snyder, 2010).

II.8. Mejores prácticas en PHP para evitar el Cross-Site Scripting (XSS) en aplicaciones web

Primero se debe hacer conciencia sobre la seguridad en aplicaciones web, para esto Snyder (2010) menciona 5 hábitos que se deben tener siempre en cuenta:

1. Nada es 100% seguro
   No se trata de desanimar a nadie, se trata de hacer ver que la seguridad no es algo simple que se activa y queda protegida la aplicación, sino que son muchas cosas pequeñas que hay que hacer para proteger lo mejor posible la aplicación.

2. Nunca confiar en los datos del usuario
   La mayoría de los usuarios usaran las aplicaciones web como estas se diseñaron para que fueran usadas, pero hay una minoría de usuarios curiosos y otros con malas intenciones que querrán destruir la aplicación, subir archivos dañinos, insertar malware, etc., por eso no se debe confiar nunca en los datos de los usuarios.

   El filtrado de datos es la piedra angular de la seguridad de aplicaciones web en cualquier lenguaje o cualquier plataforma. Usar white list es mejor que black list, es decir, considerar todos los datos inválidos a menos que sean aprobados y no al revés (PHP Security, 2005)

   - Siempre sanitizar los parámetros que se envían en las URL
   - Siempre sanitizar cada elemento del formulario, incluyendo los campos ocultos. Checar la longitud y tipo de los datos y remover las etiquetas HTML.
   - Es buena idea generar un token del lado del servidor y verificarlo en el formulario para aceptar solo POST del mismo dominio.

3. Defensa en profundidad es la única defensa
   Se debe poner todo lo posible para controlar la seguridad de la aplicación, una simple forma de login no bastará.

4. Lo más simple es lo más fácil de asegurar
   Asegurar las funciones simples, no tratar de asegurar funciones complejas ya que también será complejo asegurarlas. Utilizar las funciones nativas de PHP no reinventar la rueda.

5. Revisión en pares es crítico para la seguridad
   Las revisiones de otros siempre mejora la seguridad.

II.8.1. Deshabilitar las variables globales

Se debe configurar el archivo php.ini y cambiar el valor de la variable **register-globals** a **off** (Snyder, 2010).

Ejemplo: En el siguiente código si la variable **register-globals** esta en **on**, un atacante puede agregar a la URL **?admin=1** y tendría acceso a las tareas administrativas.

```
<?php
   // set admin flag
   if ( $auth->isAdmin() ){
      $admin = TRUE;
   }
   if ( $admin ) {
      // do administrative tasks
   }
?>
```

II.8.2. Permitir solo las entradas esperadas

Se puede especificar explícitamente las variables que se desea recibir, de esta forma si el atacante ingresa otras variables no se tomarán en cuenta (Snyder, 2010).

Ejemplo: De esta forma se obtendrán las variables que esperábamos y cualquier otra será ignorada.

```
<?php
   $expected = array( 'modelo', 'anio', 'estilo' );
   foreach( $expected AS $key ) {
       if ( !empty( $_POST[ $key ] ) ) {
          ${$key} = $_POST[ $key ];
       }
       else {
          ${$key} = NULL;
       }
   }
?>
```

II.8.3. Verificar los tipos de datos

Siempre se debe verificar los tipos de datos que se reciben para validar que son los datos que se quieren usar. En PHP existen diversas funciones nativas que hacen esto (Snyder, 2010).

`is_string($var)` = Devuelve true si $var es una cadena y no contiene números
`empty($var)` = Devuelve true si $var está vacía
`is_int($var)` = Devuelve true si $var es un numero entero
`gettype($var)` = Devuelve el tipo de la variable $var
`is_bool($var)` = Devuelve

II.8.4. Verificar la longitud

Se debe verificar la longitud en datos que deben cumplir con cierto tamaño, de esta forma se evita que se introduzcan valores extraños o erróneos.

En PHP existe la función strlen() que devuelve la longitud de una cadena.

Ejemplo: Los valores de año, deben tener exactamente 4 caracteres por lo que se puede comprobar que:

```
if ( strlen( $anio ) != 4 ) exit ( "$anio es un valor
invalido!" );
```

II.8.5. Escapar y codificar todas las entidades HTML

Sin embargo la mejor manera de deshabilitar un ataque XSS es escapando los caracteres del texto de entrada, de esta forma se le dice al navegador que el texto debe ser tratado como tal y de ninguna otra forma. De esta forma si se introduce un script la víctima no se verá afectada ya que el navegador no lo ejecutará (Pullicino, 2011).

Hay 5 etiquetas HTML que pueden ser introducidas desde cualquier entrada de la aplicación y al mostrarse ser ejecutadas por el navegador (Snyder, 2010), estas son <script>, <object>, <applet>, <iframe> y <embed>.

PHP cuenta con la función **htmlentities()** que traduce todas las entidades HTML a entidades equivalentes que no se ejecutarán ni ocasionarán ningún daño.

Ejemplo: La función safe codifica tanto las entidades HTML como JavaScript desarmando completamente cualquier código malicioso.

```
<?php
    function safe( $value ) {
        htmlentities( $value, ENT_QUOTES, 'utf-8' );
        return $value;
    }
    $title = $_POST['title'];
    $message = $_POST['message'];
    // and display them safely
    print '<h1>' . safe( $title ) . '</h1>
    <p>' . safe( $message ) . '</p>';
?>
```

II.8.6. Otras mejores prácticas

- Considerar durante las fases de desarrollo todos los usos ilegales que la aplicación pueda tener (PHP Security, 2005).

- Realizar análisis de código para detectar posibles fallas XSS (OWASP, 2010).

- Aunque se cuente con certificados SSL/TSL no asegura que la aplicación web ya no tenga vulnerabilidades XSS. Por lo tanto, se debe encriptar toda la información crítica que maneja la aplicación (Cgisecurity, 2011).

- Usar la hoja XSS que proporciona el sitio http://www.ha.ckers.org /xss.html para probar la aplicación en desarrollo y poder encontrar posibles vulnerabilidades (Egeberg, 2008).

- Siempre actualizar el software que se está usando para desarrollar, puesto que las nuevas versiones ya han reparado bugs de versiones anteriores (OWASP, 2008).

- Una vez aplicadas todas las medidas de seguridad se debe realizar un escaneo utilizando herramientas para detectar vulnerabilidades, en caso de ser así deben corregirse los defectos encontrados y volver a ejecutar las herramientas para asegurar que no se hayan insertado nuevas vulnerabilidades (Pullicino, 2011).

II.9. Checklist de Seguridad Web

Una metodología formal de seguridad en el ciclo de vida del software es esencial para proteger los recursos de la empresa. Sin embargo, se le ha dado poca importancia a este aspecto en el desarrollo de software. Tradicionalmente la seguridad del software se ha tratado como una reflexión tardía de “penetrar y parchar” al final del desarrollo. Debido a su importancia, la seguridad debe ser integrada de manera formal. (Gilliam, 2003).

La seguridad del software también se deja al último por la dificultad de integrar este proceso al ciclo de vida del software, ya que es necesario dedicarle recursos y expertos entrenados en tema de seguridad (Gilliam, 2003).

Una forma simple pero efectiva de integrar seguridad durante el ciclo de vida del software es utilizando checklist de seguridad en cada fase del desarrollo. Siguiendo las bases para generar un checklist de seguridad de software propuestas por Gilliam (2003) y con base en el checklist de seguridad en el diseño de aplicaciones web propuesto por Zemanek (2004) y las mejores prácticas recolectadas en este estudio, se realizó el checklist mostrado en la Tabla 3, la cual muestra los puntos que se deben verificar en cada una de las

fases de desarrollo de software para prevenir la vulnerabilidad de Cross-Site Scripting (XSS) en aplicaciones web desarrolladas con PHP.

Las fases de desarrollo de software que componen el checklist de la Tabla 3 se obtuvieron de Scacchi (1987) y pertenecen al clásico y tradicional Modelo en Cascada, ya que este modelo ha sido instanciado como base en casi todos los libros modernos de ingeniería de software y prácticas de programación ya que las fases que componen este modelo son las fases más comunes utilizadas por la mayoría de los modelos de desarrollo de software, y se utilizan en este checklist de tal manera que éste sea lo más flexible y genérico para que pueda adaptarse al proceso de desarrollo de software de cualquier empresa.

DEFINICIÓN DE REQUERIMIENTOS	
1. Se estableció junto con el cliente los requerimientos de seguridad de la aplicación. 2. Se estableció junto con el cliente qué información debe ser manejada con mayor seguridad en el sistema	
ANÁLISIS DE REQUERIMIENTOS	
3. Se realizó un análisis de los riesgos de seguridad que la aplicación puede presentar. (Posibles usos indebidos del sistema)	
4. Se identificaron todas las entradas y salidas de información. a) Se generó la lista de entradas y salidas de información (Lista de Entradas/Salidas).	
DISEÑO DEL SOFTWARE	
5. Se identificaron los tipos de dato permitidos para cada uno de los ítems de la Lista de Entradas/Salidas.	
6. Se identificaron las longitudes de los datos para cada uno de los ítems de la Lista de Entradas/Salidas.	
7. Se identificaron los formatos permitidos en cada uno de los ítems de la Lista de Entradas/Salidas.	
8. Se generaron las listas de tipo de dato, longitud y formato para cada uno de los ítems de la Lista de Entradas/Salidas	
9. Ningún tipo de información que permita acceso a la aplicación está almacenada en cookies.	
10. Después de cerrar la sesión, los datos son destruidos	
11. Se diseñaron soluciones óptimas para cubrir los riesgos de seguridad encontrados en la fase de requerimientos.	
IMPLEMENTACIÓN	
12. Se establecieron las validaciones de entrada para cada ítem de la Lista de Entradas/Salidas con base en el tipo de dato, longitud y formato de cada ítem.	

13. Se eliminó cualquier etiqueta de HTML o JavaScript antes de guardar los datos de entrada a la aplicación con la función `strip_tags()` o similar.	
14. Se convirtieron todas las entidades HTML y JavaScript de los datos en cada una de las salidas de información usando `htmlentities()`o similar.	
15. Se convirtieron todas las entidades HTML y JavaScript de las URLs usando `htmlentities()`o similar.	
16. En caso de que la aplicación permita subir archivos, se verificó el tipo de archivo y que el contenido no fuera malicioso.	
17. Se usaron listas blancas para permitir solo las entradas esperadas a la aplicación.	
18. Se realizó una revisión de código para detectar posibles fallas XSS.	
19. Se desarrollaron las soluciones diseñadas para los riesgos de seguridad encontrados en la fase de requerimientos.	
INTEGRACIÓN Y PRUEBAS DEL SISTEMA	
20. Se realizaron pruebas de inserción de XSS usando la hoja de trucos de http://www.ha.ckers.org/xss.html en cada campo de los formularios de la aplicación.	
21. Se utilizó al menos una herramienta de escaneo de vulnerabilidades para encontrar fallas de seguridad XSS.	
22. Se realizaron pruebas sobre las soluciones desarrolladas a los riesgos de seguridad.	
OPERACIÓN Y MANTENIMIENTO	
23. Se deshabilitó el uso de variables globales en el archivo php.ini `register_globals = off`	
24. Se utilizó al menos una herramienta de escaneo de vulnerabilidades sobre el sitio en un ambiente de producción para encontrar fallas de seguridad XSS. a. Aplicar este punto también cuando se añadan nuevos módulos o funcionalidades al sistema.	

Tabla 3. Checklist de Seguridad Web para Aplicaciones PHP

En la fase de Requerimientos del checklist anterior no se incluyó un punto donde se adquirieran los atributos de calidad requeridos por el sistema, ya que en este estudio no se está considerando a la seguridad como un atributo de calidad en sí, sino como una necesidad de cualquier aplicación web. Este estudio se centra únicamente en cómo evitar el Cross-Site Scripting en aplicaciones PHP dejando de lado el estudio de la Seguridad como atributo de calidad, ya que en ese caso sería necesaria una investigación sobre el

balanceo o Trade-offs de los atributos de calidad, el cual no es el objetivo de este estudio y mencionarlos no sería suficiente.

III. CONCLUSIONES

La vulnerabilidad Cross-Site Scripting (XSS) ha existido desde los primeros días de la Web y hoy en día continúa presente en las aplicaciones basadas en web, y seguirá siendo una de las vulnerabilidades más comunes debido a la falta de conciencia de seguridad de los desarrolladores y a que se subestima este tipo de vulnerabilidades. No significa que son malos desarrolladores, sino que la última cosa en la que piensan mientras desarrollan es en asuntos de seguridad y esto es debido a que utilizan un flujo de trabajo común: Requerimientos, análisis, diseño, codificación, pruebas, integración y despliegue.

Desarrolladores y empresas de software deben integrar la seguridad durante todo el ciclo de desarrollo de la aplicación no verlo como una característica aparte, esto puede lograrse integrando checklist de seguridad web que verifiquen en cada una de las fases si se tomaron en cuenta aspectos de seguridad.

Usuarios desprevenidos siempre habrá, de hecho se dice como una frase geek que “El eslabón más débil en la cadena de seguridad de una aplicación web es el usuario final” y lamentablemente tiene mucho de cierto ya que la ingeniería social es un arma muy certera.

Se debe tener cuidado del grado de seguridad de una aplicación, ya que las características de seguridad como cierres de sesión automático, uso de Captchas pueden dificultar la interacción con el usuario, por lo que se debe balancear los atributos de seguridad y usabilidad de la mejor manera posible.

La seguridad en aplicaciones web es un aspecto importantísimo que se está subestimando y que debe incluirse de forma efectiva desde el inicio del ciclo de desarrollo de las aplicaciones y el primer paso y más importante es crear conciencia de seguridad en desarrolladores y empresas.

III.1. Trabajo futuro

- Agregar más información a los checklist de seguridad web que cubran los demás tipos de vulnerabilidades en aplicaciones PHP.

- Integrar los checklist de seguridad web en las actividades de desarrollo de software de la empresa SoftLogik.

III.2. Beneficios obtenidos de la investigación

- Se adquirieron conocimientos sobre las vulnerabilidades que presentan actualmente las aplicaciones web, sus efectos y las formas de protegerse contra ellas.

- Se adquirieron conocimientos sobre los métodos de ataque a los sitios web y el tipo de beneficio que se trata de obtener.

- Ahora se es más consciente de la importancia de la seguridad en cualquier software, y de no dejar pasar detalles pequeños como validaciones o configuraciones del lenguaje de programación ya que cualquier falla de seguridad puede ser muy bien aprovechado por personas habilidosas para hacer daño o robar información.

III.3. Proyectos derivados

- Estudio sobre el Trade off de los atributos de seguridad y usabilidad en aplicaciones web.

- Estudio sobre el costo-beneficio de realizar actividades de integración de seguridad durante el desarrollo de la aplicación, contra el costo-beneficio de realizar actividades de detección y corrección de vulnerabilidades solo durante la fase de pruebas.

IV. REFERENCIAS

- Acunetix (2011). “Cross-Site Scripting Attack”. Documento electrónico consultado del sitio web http://acunetix.com el 17 de junio del 2011.

- Bodmer, F. (2007). “Cross-Site Scripting (XSS)”. Computer and Network Security. Documento electrónico consultado del sitio web http://phpsec.org/library/ el 13 de junio del 2011.

- CgiSecurity (2011). “The Cross-Site Scripting (CSS) FAQ”. Documento electrónico consultado del sitio web http://www.cgisecurity.com el 10 de junio del 2011.

- Egeberg, D. (2008). “PHP Security”. PHP Freaks. Documento electrónico consultado del sitio web http://phpsec.org/library/ el 14 de junio del 2011.

- Fundación OWASP (2008). “Guía de Pruebas OWASP. Pruebas de Validación de Datos. Documento electrónico consultado del sitio web https://www.owasp.org el 15 de junio del 2011.

- Fundación OWASP (2010). “OWASP - Top 10”. Documento electrónico consultado del sitio web https://www.owasp.org el 8 de junio del 2011.

- Gilliam D., Wolfe T., Sherif J. (2003). “Software Security Checklist for the Software Life Cycle”. Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE’03).

- Goodin, D. (2008). “American Express Bitten by XSS bugs”. Documento electrónico consultado del sitio web http://acunetix.com el 23 de junio del 2011.

- Goodin, D. (2009). “XSS Bugs Crawls all Over PayPal Page”. Documento electrónico consultado del sitio web http://acunetix.com el 24 de junio del 2011.

- Grossman, J. (2007). “XSS Attacks: Cross Site Scripting Exploits and Defense”. Burlington, MA. USA. Elsevier, Inc.

- Kemmerer, R., Vigna, G. (2002). “Intrusion Detection: A Brief History and Overview”. Reliable Software Group, Computer Science Department, University of California Santa Barbara.

- PHP Security Consortium (2005). “PHP Security Guide 1.0”. Documento electrónico consultado del sitio web http://phpsec.org/library/ el 16 de junio del 2011.

- Pullicino, J. (2011). "Preventing XSS Attack". Documento electrónico consultado del sitio web http://acunetix.com el 20 de junio del 2011.

- Scacchi, W. (1987). "Models of Software Evolution: Life Cycle ad Process". Carnegie Mellon University. Software Engineering Institute.

- Shiflett, C. (2005). "Essential PHP Security". O´Reilly Media Inc. Sebastopol, CA.

- Snyder, C., Myer, T y Southwell, M. (2010). "Pro PHP Security. From Application Security Principles to the Implementation of XSS Defense". Segunda edición. Apress. USA.

- Theharmonyguy (2011). "Recent Facebook XSS Attack Show Increasing Sophistication". Documento electrónico consultado del sitio web http://theharmonyguy.com el 22 de junio del 2011.

- Turner, J. (2006). "Basic PHP Security Checklist". Documento electrónico consultado del sitio web http://webappsec.org el 14 de julio del 2011.

- Web Application Security Consortium (2011). "The Web Hacking Incidents Database 2007 - 2011". Serie de documentos electrónicos consultados del sitio web http://webappsec.org el 21 de junio del 2011.

- Wikipedia (2011). "Cross-Site Scripting". Documento electrónico consultado del sitio web http://www.wikipedia.org el 9 de junio del 2011.

- Xssed (2011). El sitio en línea que archiva la lista de sitios web vulnerables a XSS. Sitio web http://xssed.com/ visitado el día 27 de junio del 2011.

- Yang, Y. (2010). "PHP Security Guide & Checklist for Websites and Web Applications – Bottom Line for Every Good PHP Developer". Documento electrónico consultado del sitio web http://www.kavoir.com el 18 de julio del 2011.

- Zemanek, G. (2004). "A Security Checklist for Web Application Design". As part of the Information Security Reading Room. SANS Institute.

IV.1. Glosario

ActionScript Es el lenguaje de programación de la Plataforma Adobe Flash. Originalmente desarrollado como una forma para que los desarrolladores programen en forma más interactiva.

Apache Es un servidor web HTTP de código abierto para plataformas Unix (BSD, GNU/Linux, etc.), Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual.

Black List En relación con seguridad XSS, se refiere a una lista de palabras las cuales son las únicas que no son válidas.

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). Se trata de una prueba desafío-respuesta utilizada en computación para determinar cuándo el usuario es o no humano.

Cookie Es un fragmento de información que se almacena en el disco duro del visitante de una página web a través de su modo a petición del servidor de la página. Esta información puede ser luego recuperada por el servidor en posteriores visitas.

Cracker El término se utiliza para referirse a las personas que *rompen* algún sistema de seguridad.

Cross-Site Scripting Es un tipo de inseguridad informática o agujero de seguridad basado en la explotación de vulnerabilidades del sistema de validación de HTML incrustado, es abreviado como XSS para no ser confundido con las siglas CSS, las vulnerabilidades de XSS originalmente abarcaban cualquier ataque que permitiera ejecutar código de "*scripting".*

CSRF (del inglés *Cross-site request forgery* o falsificación de petición en sitios cruzados), un ataque CSRF fuerza al navegador web validado de una víctima a enviar una petición a una aplicación web vulnerable, la cual entonces realiza la acción elegida a través de la víctima. Al contrario que en los ataques XSS, los cuales explotan la confianza que un usuario tiene en un sitio en particular, el cross site request forgery explota la confianza que un sitio tiene en un usuario en particular.

e-commerce 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.

Ingeniería social Es la práctica de obtener información confidencial a través de la manipulación de usuarios legítimos. Es una técnica que pueden usar ciertas personas, tales como investigadores privados, criminales, o delincuentes computacionales, para obtener información, acceso o privilegios en sistemas de información que les permitan realizar algún acto que perjudique o exponga la persona u organismo comprometido a riesgo o abusos.

JavaScript Es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas.

Malware (Del inglés *malicious software*), también llamado badware, código maligno, software malicioso o software malintencionado es un tipo de software que tiene como objetivo infiltrarse o dañar una computadora sin el consentimiento de su propietario.

MySQL Es un sistema de gestión de bases de datos relacional, multihilo y multiusuario.

Negación de servicio En seguridad informática, un ataque de denegación de servicio, también llamado ataque DoS (de las siglas en inglés *Denial of Service*), es un ataque a un sistema de computadoras o red que causa que un servicio o recurso sea inaccesible a los usuarios legítimos. Normalmente provoca la pérdida de la conectividad de la red por el consumo del ancho de banda de la red de la víctima o sobrecarga de los recursos computacionales del sistema de la víctima.

OWASP OWASP (acrónimo de *Open Web Application Security Project*, en inglés 'Proyecto de seguridad de aplicaciones web abiertas') es un proyecto de código abierto dedicado a determinar y combatir las causas que hacen que el software sea inseguro.

PHP PHP es un lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas.

SoftLogik Empresa de desarrollo de software ubicada en el estado de Zacatecas, utiliza metodologías ágiles para el desarrollo de productos PHP.

SQL Injection Inyección SQL es un método de infiltración de código intruso que se vale de una vulnerabilidad informática presente en una aplicación en el nivel de validación de las entradas para realizar consultas a una base de datos.

WASC La Web Application Security Consortium (WASC) es un grupo sin ánimo de lucro compuesto por expertos y practicantes de la industria y representantes de organizaciones que producen código abierto y animal a utilziar las mejores prácticas de seguridad.

White List En relación con seguridad XSS, se refiere a una lista de palabras las cuáles son las únicas válidas.

Yii Framework Yii es un framework de desarrollo de aplicaciones web PHP basado en componentes y utiliza el modelo MVC (Modelo vista controlador).