

Universidad Nacional Del Nordeste

Facultad de Ciencias Exactas y Naturales y Agrimensura

Licenciatura en Sistemas de Información.

Proyecto Final de Carrera

“Aplicación de medidas de seguridad contra XSS en un sitio web para el INICNE”

Autor

Silva, Pablo Ariel

pabloariel01@hotmail.com

Profesores Orientadores

Lic. Bosco, Juan Francisco

boscojfrancisco@gmail.com

Expt. Cuzziol, Juan José

jcuzziol@hotmail.com

Corrientes – Argentina – Año 2016

Capítulo 1.

Introducción.

Breve estado del arte.

Hoy en día si se queremos desarrollar una aplicación web tenemos, necesariamente, que pensar en la web 2.0 [1] donde diseño centrado en el usuario, colaboración e interoperabilidad son las bases, y los sitios antiguos que solo muestran información dejan de ser populares y son reemplazados por sitios donde los usuarios pasan a tomar un papel central, pueden interactuar entre ellos y con las aplicaciones generando nueva información y se cambia el uso de links para mostrar distintos contenidos por sitios dinámicos donde se puede mostrar distintos contenidos sin cargar otra página, permitiendo crear aplicaciones similares a las de escritorio sobre los mismos [2].

El avance de la tecnología, también ha permitido que la capacidad de conectarnos crezca exponencialmente, pasamos de usar computadoras para trabajar y conectarnos a internet, a tener múltiples dispositivos conectados y poder elegir entre ver información, en una computadora, Tablet o teléfono celular, con lo que pasamos de tener que utilizar solo la computadora, por diseñar una aplicación para computadoras y otra para móviles, a llegar a lo que hoy se conoce como responsive design, poder diseñar una sola aplicación, que se adapte a los distintos dispositivos.

Para ser posible la llegada de sitios dinámicos se emplean distintos tipos de lenguajes, por un lado se continuo usando HTML [3], un lenguaje de etiquetado que nos permiten dar una estructura básica y un código para la definición de contenido de nuestro sitio, y se lo complemento con lenguajes de Script [4], como JavaScript, el cual posee una gran variedad de librerías adicionales entre las cuales podemos resaltar a Jquery y Angular, que se pueden añadir para incorporar nuevas funcionalidades, con hojas de estilo en cascada o también conocido como CSS [5] que cumplen la función de organizar la presentación y el aspecto del sitio.

Este conjunto de tecnologías funcionando de forma coordinada para evitar los tediosos tiempos de recarga, permitiendo crear aplicaciones interactivas y fáciles de usar se conoce como AJAX (Asynchronous JavaScript and XML) [6], el cual en sus principios no tuvo mucha popularidad, pero con el tiempo eso cambio, transformándolo en un requisito fundamental para que una aplicación web tenga éxito.

Para Incorporar estas tecnologías a una aplicación web satisfactoriamente, se debe realizar modificaciones a la estructura de la misma y pasar de utilizar solamente la arquitectura cliente servidor a incluir patrones de diseño, como el patrón MVC [2] el cual permite desarrollar una aplicación web de una forma ordenada y con un código modularizado y legible.

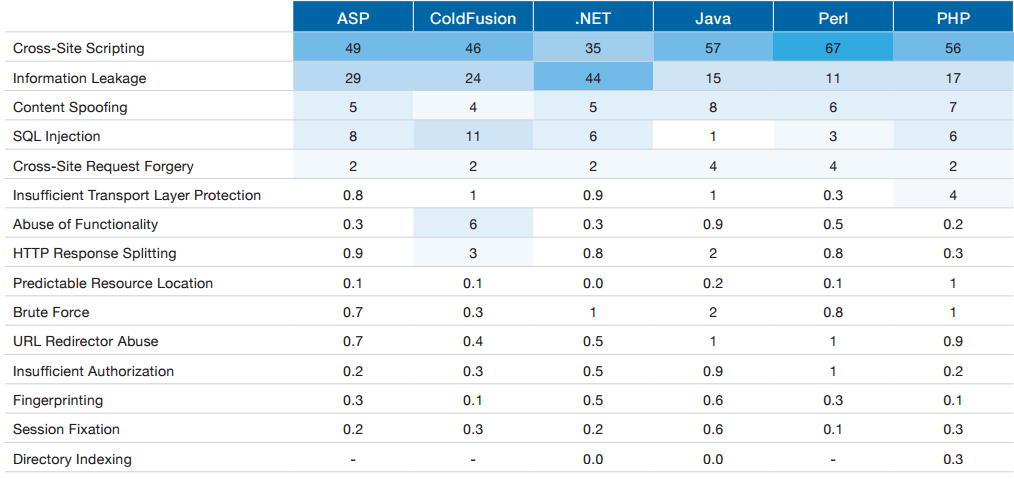
Además hay que tener en cuenta nuevas consideraciones de seguridad, el mismo dinamismo que aporta funcionalidades a las aplicaciones también agrega nuevos puntos de falla que pueden servir para fines maliciosos.

Muchas veces se confía que las aplicaciones web donde se deposita nuestra información es confiable y que los datos están seguros, aunque según una encuesta de Oracle [7] se puede observar que la mayoría de las bases de datos de las organizaciones carecen de seguridad suficiente. Pero esta confianza está empezando a bajar debido al número de ataques a sitios web dejando expuesta sensible a manos de los atacantes, se está aumentando la presión por mejorar la seguridad de las bases de datos [8].

Actualmente hay una fundación llamada OWASP [9] la cual lleva a cabo distintos proyectos relacionados a seguridad de aplicaciones web, entre ellos desarrolla regularmente un TOP 10 de los principales riegos de las aplicaciones web, entre los que se encuentran las secuencias de comandos de sitios cruzados como unos de los más explotados.

Las secuencia de comando de sitios cruzados o mejor conocido como Cross Site Scripting (XSS) [10] [11] [12] es una de las vulnerabilidades más usuales de las aplicaciones web, se mantiene entre las primeras 5 amenazas de OWASP desde el 2005 y según Whitehat [13] ocupa el tercer puesto en probabilidad de vulnerabilidad de aplicaciones web, con un 47% y es la principal vulnerabilidad en aplicaciones web programadas en PHP, ver Fig. 1. Las vulnerabilidades XSS permiten la inyección de código por usuarios maliciosos en páginas web vistas por otros usuarios, que luego son ejecutados por navegador del cliente, con los permisos que esta esta aplicación posee.

Fig. 1 porcentaje de vulenerabilidad por lenguaje [whitehat2014]



Estos ataques son basados códigos HTML o JavaScript, que utilizan como anfitrión al sitio infectado, y se ejecuta del lado del cliente, por lo que se vuelve difícil detectarlo, permitiendo realizar distintos ataques [14].

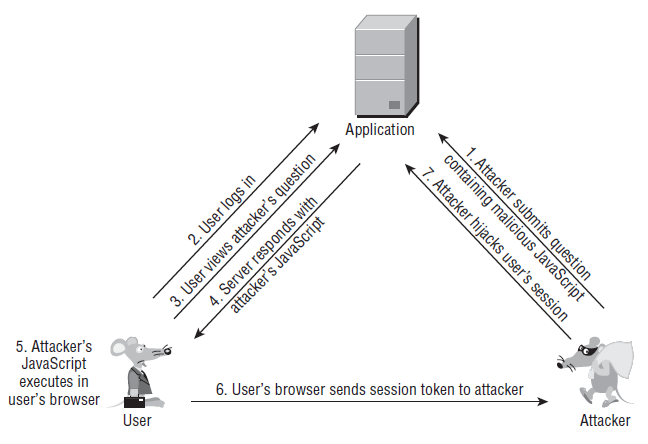
Las vulnerabilidades XSS se dividen en tres tipos [10]:

XSS almacenados o persistentes: ocurren cuando un usuario ingresa a un sitio donde se inyecto un código malicioso, y se lo dispone en el sitio en alguno de los campos, (como comentarios en un blog, o de un formulario web), el código se almacena, y se carga con cada visita, es decir, requiere de dos peticiones a la aplicación, una para el atacante donde inyecta el código y otra para el usuario en la cual se ejecuta el código.

Explotando una vulnerabilidad de este tipo de ataques un robo de un token de sesión de usuario de un blog funcionaria de la siguiente forma: el atacante forjaría una entrada al blog, con un código JavaScript escondido detrás de un mensaje o una imagen, un usuario entraría al sitio, su navegador ejecuta el código malicioso, el cual podría enviar datos de la sesión activa en el blog (ver Fig. 2).

Explotar estas vulnerabilidades permite también al atacante incluir múltiples scripts, ampliando enormemente las dimensiones del ataque y la complejidad del mismo.

Fig. 2 XSS almacenados [11]



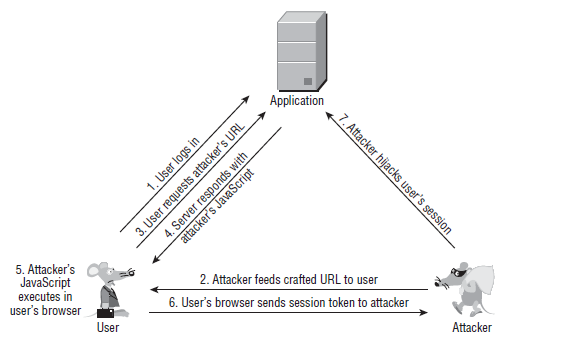
Los XSS indirectos o reflejados ocurren en las aplicaciones web donde las entradas no confiables, son incluidas inmediatamente en la salida, y reflejadas del servidor web directamente en el navegador. Ocurren principalmente en mensajes de error, peticiones a motores de búsqueda o pre visualización de comentarios. Requieren que la víctima acceda a una URL creada especialmente por el atacante y en ella estará el código malicioso [11].

En términos prácticos, para realizar un robo de sesión (véase Fig. 3) en un sitio explotando esta vulnerabilidad, el atacante debería forjar una URL a un sitio web vulnerable, dentro de la cual incluiría un código malicioso y se lo enviaría a la víctima, una vez utilizado la URL el navegado ejecuta el código y envía los datos de sesión al atacante.

En este tipo de ataques la ingeniería social juega un papel muy importante, ya que las URL muy largas resultan sospechosas a la mayoría de la gente, por lo que se utilizan distintos recursos para lograr que accedan al link. Los acortadores de URL resultan muy útiles para lograr disfrazar la URL por una más amigable, pero también se los puede esconder detrás de una imagen, o un código QR.

Es importante resaltar que para que el ataque resulte exitoso la victima debe tener una sesión activa en el sitio del cual se quiere robar la sesión, de otro modo el script resultaría inservible.

Fig. 3 XSS reflejados [11]



Basadas en el DOM de la aplicación [12], son muy parecidos a los ataques reflejados. Si un código de JavaScript accede a una URL como un parámetro de una petición al servidor y utiliza esta información para escribir HTML en la misma página sin ser codificada empleando etiquetas HTML, existe un agujero XSS, dado que estos datos escritos serán interpretados por los navegadores como código HTML que puede incluir en si código adicional del lado del cliente.

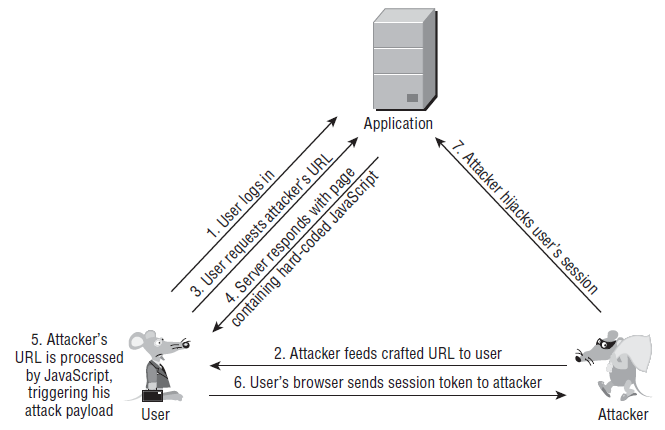
Son explotables en sitios donde buscando dinamismo se modifica el DOM del sitio desde la URL. La porción de la URL antes de “&” se utiliza para indicar al navegador a que parte del sitio dirigirse, y la porción adelante del signo queda en el DOM. La diferencia con los XSS reflejados es que estas no son enviadas al navegador, sino que quedan en el DOM. Ej.:

http://victim/promo?product\_id=100&title=Foo#<SCRIPT>alert

('XSS%20Testing')</SCRIPT>

El modo de operar con esta técnica es muy parecido al de los ataques reflejados, se envía una URL especialmente diseñada que contenga un script con un código malicioso, el cual al ser ejecutado enviara el token de sesión al atacante, ver Fig. 4.

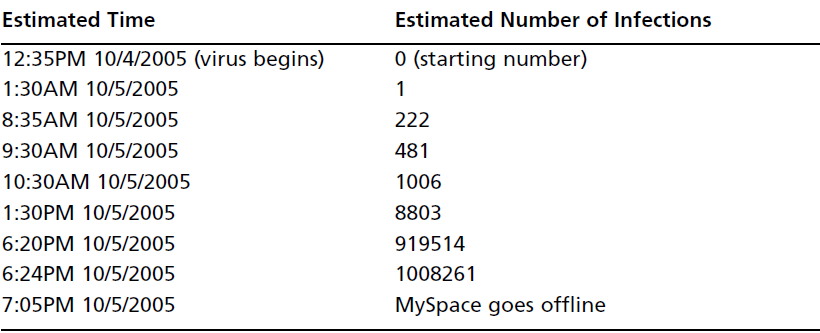
Fig. 4 XSS basados en DOM [11]



Si bien en los ejemplos anteriores se ejemplifico como se podría explotar los tipos de tres vulnerabilidades para robar un token de sesión de usuario, los ataques XSS pueden variar enormemente dependiendo de su payload, pueden ser coficados para actuar como software malicioso basado en web (malware), gusanos basados en AJAX, ataques de fuerza bruta, detección de inicio de sesión, control de zombis, escaneo de puertos o de red o tomar el control del navegador de la víctima entre otros [12].

Este tipo de vulnerabilidades tomo relevancia a fines del 2005, con una broma que escalo increíblemente rápido causo que los servidores de la red social MySpace tengan que ser puestos offline. Un gusano creado con código JavaScript con el objetivo de añadirse a sí mismo a la lista de “heroes” de la persona que visitaba su perfil, y luego el código se copiaba al perfil del usuario infectado, provocando una expansión exponencial del gusano, como se puede ver en Fig. 5

Fig. 5 Estimado de infecciones de Samy [12]



A medida de estas vulnerabilidades se volvían más populares y conocidas, tanto las compañías como los investigadores empezaron a desarrollar frameworks que permitan automatizar el proceso de ataque y de testeo, entre los más conocidos podemos mencionar a BeEF, AtackAPI, CAL9000, XSS-PROXY [12] y Xenotix [15].

Desde los principios de JavaScript se previeron los riesgos de enviar código ejecutable al navegador del cliente, entre ellos el principal era que un código ejecutado en una de las pestañas del navegador, podía acceder a información de las demás, por lo que se implementó la política de same-origin, la cual permite interacción entre objetos paginas mientas provengan del mismo dominio y con el mismo protocolo. A partir de ahí se implementaron se introdujeron nuevos mecanismos y políticas de control en los navegadores y lenguajes del lado del cliente, entre ellas la más destacada es el filtro contra XSS proporcionado por los navegadores.

Actualmente la mayoría de los navegadores proveen filtros básicos para evitar ataques XSS, pero para ser posible que estén activos todo el tiempo, son muy limitados y ninguno asegura que el sitio sea seguro. Considerando que si la aplicación es vulnerable en algunos navegadores, puede afectar a un gran número de usuarios, por lo que no se debería confiar únicamente que el filtro del navegador web proteja a la aplicación de todos los ataques [16].

En este proyecto se aplicaron técnicas para mitigar los ataques XSS [10] [14] [17] [11] [12], se controlaron las entradas a las funciones de la aplicación [18] [11] [17], siendo estas una de las vulnerabilidades más comunes en los aplicativos web.

Para realizar el control existen diferentes técnicas, como el uso de librerías, proxy de aplicación, la validación de entradas y el escape de caracteres considerados peligrosos, lo que puede ser hecho del lado del cliente o del servidor, siendo las primeras más seguras, debido a que no pueden ser evadidas mediante data tampering ,entre otras [17] [16].

Objetivo.

Objetivo general:

Estudiar las técnicas de Seguridad Web orientadas a la prevención de inyecciones de código malicioso y aplicarlas en el desarrollo de un sistema Web para el INICNE.

Objetivos específicos:

* Aplicar medidas de protección contra Cross Site scripting.
* Profundizar el estudio de conceptos de seguridad web.
* Implementar un framework para aumentar la seguridad, crear una aplicación estructurada, y con un código legible y mantenible.
* Diseñar y desarrollar una base de datos.

Fundamentación.

Actualmente el INICNE(Instituto de Ictiología del Nordeste) trabaja con datos de muestreos recolectados de distintas campañas en diferentes localidades, los cuales son anotados en distintas planillas en papel al momento de ser recolectados y una vez finalizada la campaña uno de los investigadores se encarga de transcribirlos a una planilla de Excel la cual esta indexada por un identificador asignado secuencialmente a cada uno de los peces muestreados y posee un alto índice de redundancia de datos, la cual es copiada al final de una planilla histórica, y otra planilla con datos de otras campañas, para luego ser enviados a distintos documentos Excel para ser tratados, formateados y utilizados en tablas dinámicas para generar informes, un trabajo que se realiza en una semana.

Este trabajo podría reducirse utilizando una base de datos relacional, para almacenar la información de una forma más eficiente, tener los datos en un solo lugar y realizar consultas SQL para recuperar los datos deseados, con el formato requerido de una forma más rápida. Como las personas que utilizan los datos desconocen dicho lenguaje y, para garantizar usabilidad y una mayor eficiencia se desarrolló una aplicación web que explote la base de datos, permita mostrar toda la información y generar los informes de una forma más sencilla. La una carga de datos se puede mejorar, quitando los datos duplicados, e introduciendo únicamente los datos necesarios.

Aun cuando los usuarios potenciales de la aplicación carecen conocimientos informáticos, se debe tener en cuenta la relativa facilidad que aportan los frameworks para prueba de XSS y al ser una aplicación web se debe prever de que otros usuarios atenten los aspectos de seguridad.

Según OWASP [9] los riesgos más comunes de las aplicaciones web son las inyecciones SQL [10] y XSS, por lo que el uso de un framework y de librerías especializadas permite la prevención en gran medida de estas amenazas. Por otro lado se utiliza un sistema de control de usuarios para garantizar que solo los usuarios deseados puedan acceder a la aplicación y a la vez poder restringir el acceso de determinados sitios de la aplicación.

Capítulo 2.

Metodología.

Se opto por utilizar un ciclo de vida basado en XP utilizando como base el trabajo presentado en [19]





Capítulo 3.

Herramientas y/o lenguajes de programación.

Capítulo 4.

Resultados.

Capítulo 5.

Conclusiones y futuros trabajos

# Referencias

|  |  |
| --- | --- |
| [1] | . T. O'Reilly, « What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software.,» *Communications & Strategies,* nº 65, pp. 17-37, 2007. |
| [2] | A. MacCaw, JavaScript Web Applications, North, Sebastopol, CA: O'Reilly Media, 2011. |
| [3] | W3C, «W3C: HTML & CSS,» [En línea]. Available: https://www.w3.org/standards/webdesign/htmlcss. |
| [4] | D. Hazaël-Massieux, «W3C,» [En línea]. Available: https://www.w3.org/standards/webdesign/script. |
| [5] | H. W. Lie y B. Bos, «W3C,» 11 4 2008. [En línea]. Available: https://www.w3.org/TR/2008/REC-CSS1-20080411/. |
| [6] | A. T. Holdener III, Ajax: The Definitive Guide, North, Sebastopol, CA: O'Reilly Media, 2008. |
| [7] | J. McKendrick, «Dbs. Security Superhero: IOUG Enterprise Data Security Survey,» Unisphere Research, New Jersey, 2014. |
| [8] | A. Bamrara, «Evaluating Database Security and Cyber Attacks: A Relational Approach,» *Journal of Internet Banking and Commerce,* vol. 20, nº 2, 2015. |
| [9] | OWASP, «OWASP Top Ten Project,» 12 junio 2013. [En línea]. Available: www.owasp.org/images/5/5f/OWASP\_Top\_10\_-\_2013\_Final\_-\_Espa%C3%B1ol.pdf. |
| [10] | E. Couture, «Web Application Injection Vulnerabilities a Web App’s Security Nemesis?,» GIAC (GWAPT) Gold Certification, SANS Institute, Mayo 2013. |
| [11] | D. Stuttard y M. Pinto, The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws, Indianapolis, Indiana: John Wiley & Sons, Inc., 2011. |
| [12] | S. Forgie, J. Grossman, R. Hansen, A. Rager y P. D. Petkov, XSS Attacks: Cross Site Scripting Exploits and Defense., Burlington, MA: Syngress Publishing, Inc., 2007. |
| [13] | WhiteHat Security, « WhiteHat Security Status Report,» 2015. [En línea]. Available: https://info.whitehatsec.com/rs/whitehatsecurity/images/2015-Stats-Report.pdf. |
| [14] | B. B. Guptaa, K. P. Meenaa, S. Guptaa, S. Gangwara y M. Kumara, «Cross-Site Scripting (XSS) Abuse and Defense: Exploitation on Several Testing Bed Environments and Its Defense,» *Journal of Information Privacy and Security,* vol. 11, nº 2, pp. 118-136, 2015. |
| [15] | «Owasp,» 20 junio 2016. [En línea]. Available: https://www.owasp.org/index.php/Main\_Page. [Último acceso: 06 julio 2016]. |
| [16] | M. Heiderich, E. A. Vela Nava, G. Heyes y D. Lindsa, Web Application Obfuscation ‘-/WAFs..Evasion..Filters//alert(/Obfuscation/)-’, Burlington, MA: Elsevier, 2011. |
| [17] | P. Brady, «Survive the Deep End: PHP Security,» 26 Octubre 2015. [En línea]. Available: http://phpsecurity.readthedocs.org/en/latest/. |
| [18] | B. Gupta, «Web Application Security – What You Need to Know,» de *Thirty-Third Annual Pacific Northwest Software Quality Conference*, World Trade Center Portland Portland, Oregon. pp 271-280, Octubre 2015. |

Anexos.