

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.

Programación web

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.

Seguridad Informática

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.

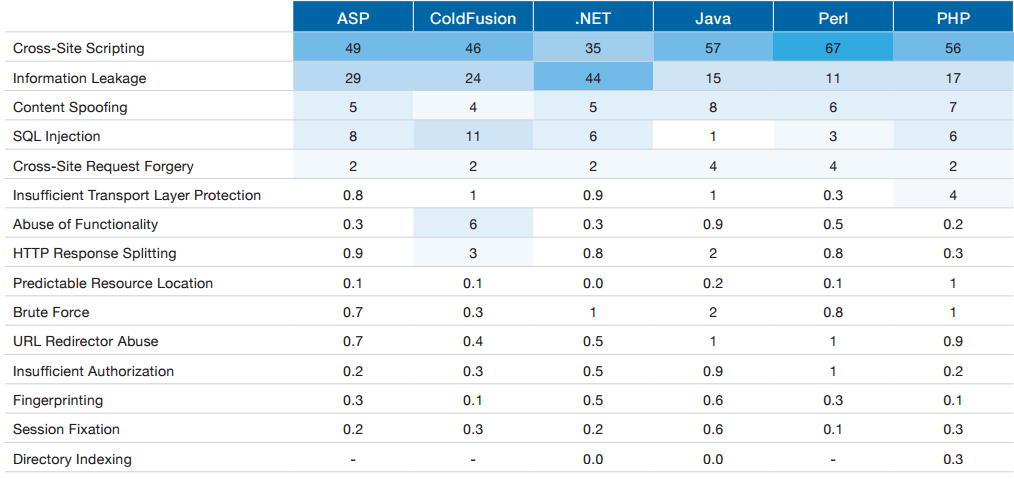
Muchas veces se confía que las aplicaciones web donde se deposita nuestra información es confiable y que los datos están seguros, aunque según una encuesta de Oracle [7] se puede observar que la mayoría de las bases de datos de las organizaciones carecen de seguridad suficiente. Pero esta confianza está 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.

Seguridad Orientada a XSS

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 vulnerabilidad 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].

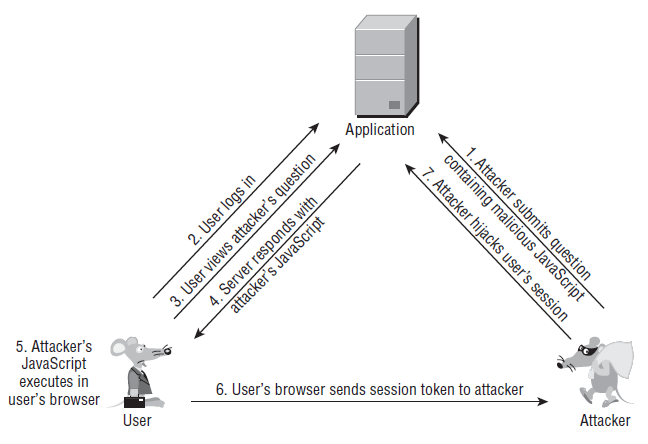
Las vulnerabilidades XSS se dividen en tres tipos [10]:

***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]



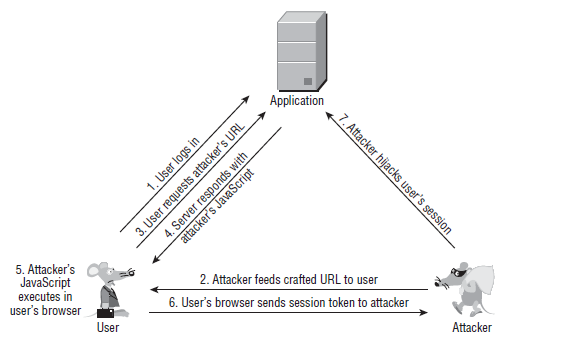
***Los XSS indirectos o reflejados*** ocurren en las aplicaciones web donde las entradas no confiables, son incluidas inmediatamente en la salida, y reflejadas del servidor web directamente en el navegador. Ocurren principalmente en mensajes de error, peticiones a motores de búsqueda o pre visualización 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.

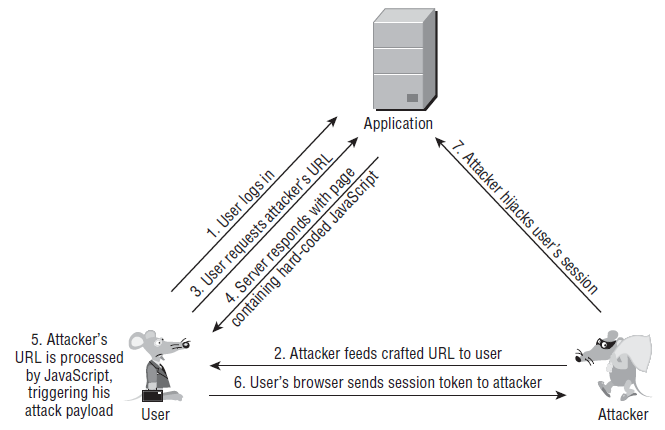
Son explotables en sitios donde buscando dinamismo se modifica el DOM del sitio desde la URL. La porción de la URL antes de “&” se utiliza para indicar al navegador a que parte del sitio dirigirse, y la porción adelante del signo queda en el DOM. La diferencia con los XSS reflejados es que estas no son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

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

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

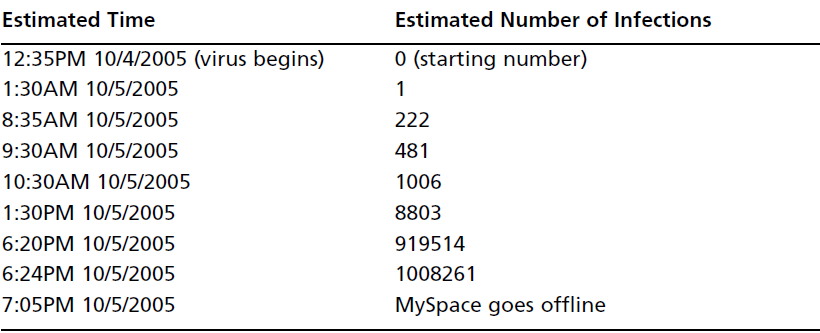
El modo de operar con esta técnica es muy parecido al de los ataques reflejados, se envía una URL especialmente diseñada que contenga un script con un código malicioso, el cual al ser ejecutado enviara el token de sesión al atacante, ver Fig. 4.

Fig. 4 XSS basados en DOM [11]



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 “héroes” 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].

Encadenamiento de ataques

En escenarios reales los ataques XSS rara vez son utilizados solos. Entendiendo que XSS son un conjunto de técnicas de ataque en las que se fuerza a un navegador a ejecutar código remoto, se puede decir que por si mismos podrían no realizar mucho daño al visitante del sitio, como también que el resultado de un ataque exitoso podría tener repercusiones muy severas.

En los ejemplos anteriores se ejemplifico como se podría explotar los tipos de tres vulnerabilidades para robar un token de sesión de usuario, pero los ataques XSS pueden variar enormemente dependiendo del contenido de su payload o carga útil. El ataque XSS posibilita al atacante tener control del sitio web, el payload del ataque podría contener otro tipo de ataque: si se combina pueden ser codificados para mostrar una simple publicidad(Addware) o redireccionar a un sitio web determinado(Defacing), como también para actuar como software malicioso basado en web (malware), gusanos basados en AJAX(Worms), 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].

En el caso de Sammy por subestimar el efecto del gusano provoco que uno de los sitios mas visitados de internet fuese fuera de línea, pero con otras intenciones prodria tomarse el control de multiples computadoras mediante ejecución de código remoto y controlarlas como zombies para realizar ataques DoS o para romper hashes de forma distribuida como puede realizarse utilizando Ravan [16] o herramientas similares.

No siempre este tipo de ataques es el final de la cadena, muchas veces no existe una vulnerabilidad directa para realizar el ataque, lo que hace imposible realizar un XSS reflejado o uno basado en DOM, pero los almacenados son un caso distinto, puede que el sitio tenga otro tipo de vulnerabilidad como ser inyecciones SQL [referencia] lo que permitiría a un atacante insertar directamente en una tabla de la base de datos un script sin tener que preocuparse por filtros de los campos de entrada del sitio, o utilizar ingeniería social para obtener acceso al servidor e importar el script directamente como un archivo mas del sitio.

Medidas de prevención

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].

Además existen otras técnicas para mitigar este tipo de ataques como el uso de librerías, proxy de aplicación, la validación de entradas y el escape de caracteres considerados peligrosos, lo que puede ejecutarse tanto del lado del cliente como del lado del servidor. Como se puede ver en [10] [14] [17] [11] y [12], cada uno de los cuales se aplican en distintas etapas del ciclo de vida del software, como también metodologías basadas en ciclos de vida para software seguro [18], [19] y [19], las cuales usan métodos para detectar categorizar amenazas como STRIDE [20] y DREAD [15].

Va a metodología y resultados

En este proyecto se aplicaron múltiples técnicas para mitigar los ataques XSS [10] [14] [17] [11] [12], las que se emplearon en distintas etapas del ciclo de vida.

En la etapa de análisis y diseño para la detección de amenazas de la aplicación web, utilizando STRIDE [20] para categorizar las amenazas, y DREAD [15] para clasificarlas. En la codificación se controlaron las entradas a las funciones de la aplicación [21] [11] [17], siendo estas una de las vulnerabilidades más comunes en los aplicativos web.

En la trapa de pruebas,

Para realizar el control se recurrió a diferentes técnicas, como el uso de librerías, proxy de aplicación, la validación de entradas y el escape de caracteres considerados peligrosos, lo que puede ejecutarse 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 ser utilizado en 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 múltiples 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. Estas planillas poseen una redundancia de datos muy alta, la cual es copiada al final de una planilla histórica donde serán resguardados de forma permanente, y también son volcados otra planilla temporal con datos de más campañas, para luego ser enviados a distintos documentos Excel donde serán tratados y utilizados en tablas dinámicas para generar informes, un trabajo que se realiza en una semana y la probabilidad de errores y perdida de información al pasar de una planilla a otra es muy alta.

Esta carga de trabajo podría reducirse utilizando una base de datos relacional, para persistir la información de una forma más eficiente, ordenada, 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 carga de datos se puede mejorar, quitando los datos duplicados, e introduciendo únicamente los datos necesarios.

Por otro lado la información almacenada en la base de datos representa el trabajo y esfuerzo de los investigadores para obtenerla y es la base para la generación y validar hipótesis y estudios científicos, lo que implica la perdida de esta información o la manipulación indebida de la misma puede estropear años de trabajo. Lo que lleva a pensar que tanto o mas importante que la representación de los datos es su seguridad e integridad y, dado que por motivos de accesibilidad geográfica una de las formas mas eficiente de distribuir una aplicación y mantener su información segura y consistente en solo lugar es utilizando una aplicación con arquitectura cliente-servidor, por lo que se eligió desarrollar una aplicación web en donde el servidor se encarga de las tareas de autenticar y autorizar a los usuarios para brindarle los datos o recibir sus datos y almacenarlos; y el cliente de mostrar la información al usuario e interactuar con el mismo.

Esto se puede confirmar con OWASP [9], los riesgos más comunes de las aplicaciones web son las inyecciones SQL [10] y XSS, actualmente facilitadas por frameworks y programas para automatizar ataques, 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.

Fase 1 revisión sistemática de la literatura

~~Antes de iniciar con el desarrollo de la aplicación fue necesaria una lectura intensiva para determinar que metodología de ciclo de vida se utilizaría en la aplicación considerando las metodologías agiles y cuáles eran sus ventajas respecto a las tradicionales, una vez elegida las metodologías agiles se inició otra búsqueda para verificar si existían metodologías agiles con un enfoque en seguridad.~~

Al utilizar un enfoque con elementos y metodologías nuevas en lo que respecta a contenidos dados a lo largo de la carrera, fue necesaria una investigación constante para iniciarse y comprender las distintas temáticas.

A demás la literatura sobre la combinación de metodologías agiles para el desarrollo de aplicaciones en conjunto con la aplicación de métodos de seguridad para aplicaciones web carece de numerosas investigaciones lo que dificulto la selección de las mismas.

Fase 2. Aplicación de ciclo de vida

Como ciclo de vida en el desarrollo del proyecto software se utilizó una variante de XP respetando algunos de sus principios y aplicando un enfoque de seguridad basado en modelos de desarrollo de software seguro [19] y OWASP [18], añadiendo un análisis y clasificación de amenazas utilizando los métodos STRIDE [20] y DREAD [15].

En cuanto a los principios de XP se realizaron las siguientes modificaciones:

* La programación de a pares fue sustituida por la revisión de código con uno de los profesores orientadores
* Tener al cliente o a un representante (conocido como product owner) en el lugar de trabajo es bastante difícil de lograr hoy en día, en especial en este caso particular donde los clientes realizan viajes de forma periódica, por lo que se reemplazó al cliente en sitio, con entrevistas personales regulares, en las cuales se realizan las consultas necesarias.
* Debido al carácter personal del trabajo la integración continua se limito a la utilización de un servidor de versionado (git), la momento de iniciar una nueva funcionalidad se creaba una rama separada, y al terminar su desarrollo, se realizaban pruebas manuales, se verificaban los criterios de aceptación de la historia de usuario que origino dicha funcionalidad y en caso de pasar las las pruebas se lo añadia al servidor unificando la rama maestra con las funcionalidades
* Con respecto a la planeación se opto por utilizar “The Planning Game”, la cual se basa en el criterio del cliente para determinar cuáles historias de usuarios aportan mayor beneficio de negocio, para así asignar mayor prioridad al momento de desarrollar, y luego a esas historias se le asigna un tiempo estimado o costo de desarrollo. Se detallara mas adelante la metodología utilizada para decidir las asignaciones de historias de usuarios a los Sprints.
* En cuanto a las pruebas al estar este trabajo enfocado en seguridad, se limitara a realizarar pruebas de inyeccion de código JavaScript y como la aplicación responde ante código ya inyectados(almacenado).

Tambien se propone trabajar en forma comunitaria, es decir que el código pertenezca a todos los desarrolladores y no solo sea comprensible por su desarrollador, lo que permite que cualquier integrante pueda analizarlo, testearlo y hacer refactoring de ser necesario, para lo cual se establecen normas de codificación. A pesar de ser un proyecto unipersonal las normas de codificación nunca están de mas por lo que se estableció:

* Utilizar la estructura de directorios propuesta por CodeIgniter (framework PHP)
* Utilizar un contolador de AngularJS distinto para cada vista.
* Mantener trazabilidad de nombres de funciones desde los procedimientos almacenados en la base de datos, pasando por el modelo y controlador de PHP hasta el contolador de AngularJS.
* Utilizar metáforas propuestas por XP para la creación de nombre de funciones.
* Documentar funciones parámetros en caso de no ser suficientemente autodescriptivas.

Las historias de usuarios fueron divididas por temas, agrupándolas por temática. Luego, de forma independiente a los temas, las historias de usuario fueron asignadas divididas en 3 Sprints, en base el puntaje asignado a cada una de las historias de usuario.

La duración de los Sprints se definió en base a la cantidad de historias de usuario existentes al inicio del primer Sprint y la complejidad de las mismas determinada por experiencias previas en desarrollo de tareas similares.

En cuanto a las entregas se opto por realizar una muestra del producto al final del primer Sprint para establecer cuestiones de usabilidad y diseño, pero no realizar una entrega formal hasta el final del segundo Sprint, donde el software ya cuenta con mas funcionalidad y autenticación, se espero a que realicen una prueba de funcionamiento y luego se solicito una retrospectiva. En base a esta se tomaron las sugerencias y se modificaron las historias de usuario, añadiendo los nuevos criterios de aceptación.

A continuación se describe el flujo de trabajo y parte de la primier iteración:

Previo al primer Sprint se inicio con una entrevista personal con el encargado del instituto donde se buscaba comprender la situación y determinar las funciones básicas y la viabilidad del sistema.

Como resultado de esta entrevista se pudieron extraer las primeras historias de usuario junto con un documento Excel con multiples hojas, una principal conteniendo la información completa en forma de tabla única y el resto distintas agrupaciones y tablas dinámicas en base a la principal.

Posterior al análisis de la tabla se formalizaron las historias de usuario,se evaluaron las posibles lenguajes de programación y bases de datos a utilizar y se opto por el uso de PHP como backend con el framework CodeIgniter para la autenticación, autorización y acceso a datos en forma de API REST, y el frontend va a estar formado por un template básico provisto por el backend mas el contenido de las vistas que van a ser modeladas y asiganadas dinámicamente con el uso de AngularJs.

Se realizo una segunda entrevista para aclarar cuestiones, analizar las historias de usuario anadir criterios de aceptacion, crear nuevas historias y luego establecer prioridad entre las historias.

Para la primera iteración:

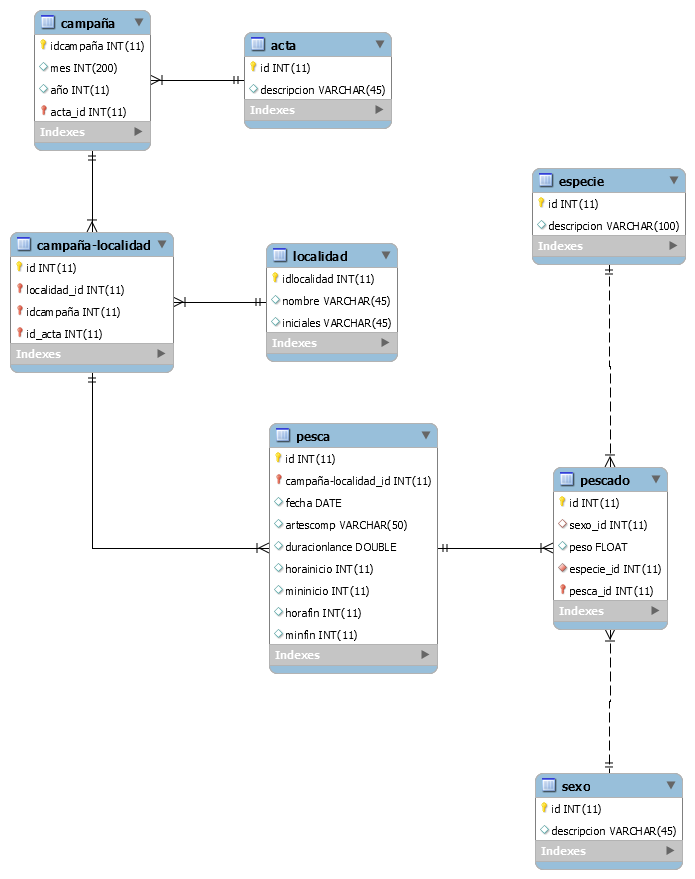
Tabla 1:Historias de Usuario de primer iteracion



Al ser XP incrementalen cada iteración se agregan los elementos necesarios para completar la misma y se controlan que los elementos anteriores sigan siendo validos.

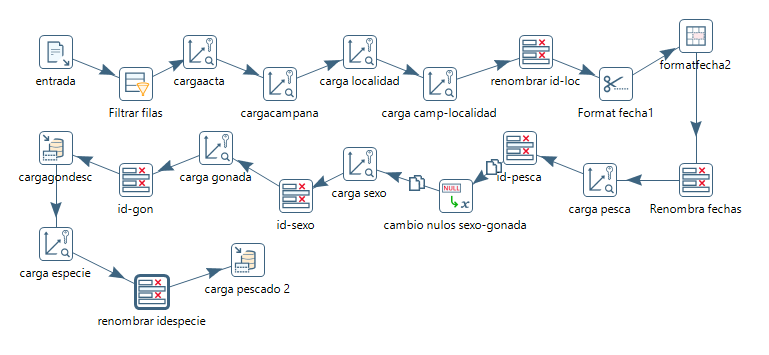
Por lo que al iniciar la primer iteración se creo el primer esquema de base de datos conteniendo solo los datos necesarios para cumplir con las historias de usuario designadas a este Sprint.

Fig. 6: Primer esquema de base de datos



Debido al volumen de datos se creo un proceso ETL con la herramienta Pentaho la cual permitio una extracción y volcado rápido de datos, manipulación y limpieza de datos y creación de llaves primarias y foráneas.

Fig. 7:Primer proceso ETL



Una vez cargados los datos se inicio el trabajo de codificación, empezando por el backend con PHP crearon los primeros modelos de datos que realizaban consultas a la base de datos y luego los controladores que tomaban los datos y los enviaban a las vitas.

Basado en OWASP [18] se realizó el modelado de amenazas, iniciando por una descripción de la aplicación sobre la cual estamos trabajando, como podemos ver en Tabla 2.

Tabla 3 Análisis de amenazas

|  |  |
| --- | --- |
| Análisis de amenazas | |
| Versión de aplicación | 2.0 |
| Descripción | Sitio web para investigadores de INICNE.  Los usuarios de la aplicación serán:   1. Administrador 2. Usuario autenticado 3. Asistente   Un administrador podrá crear, modificar y eliminar usuarios, agregar, modificar y leer datos de aplicación mientras que el usuario registrado solo podrá acceder a la información. El asistente solo podrá agregar, modificar y leer datos |
| Participantes | Silva Pablo |
| Revisado por | Bosco Francisco |

Seguidos por un análisis de niveles de confianza, los cuales representan el nivel de acceso que tendrán los agentes externos a la aplicación que serán cruzados con los puntos de entrada y los activos que posee la misma.

Tabla 4 Niveles de confianza

|  |  |  |
| --- | --- | --- |
| Niveles de confianza | | |
| Id | Nombre | Descripción |
| 1 | Usuario anónimo | Un usuario que se conectó a la aplicación, pero no inicio sesión |
| 2 | Usuario con credenciales validas | Usuario que se conectó a la aplicación, e inicio sesión con credenciales validas |
| 3 | Usuario sin credenciales validas | Usuario que intentan iniciar sesión con credenciales no validas |
| 4 | Administrador | Puede añadir nuevos usuarios, agregar datos de nuevas campañas |
| 5 | Asistente | Puede agregar y actualizar datos de las campañas |
| 6 | Administrador de base de datos | Tiene permisos de lectura y escritura en la base de datos que utiliza la aplicación |
| 7 | Administrador del sitio web | Puede cambiar la configuración |
| 8 | Usuario del servicio de servidor web | Es el usuario con el cual la aplicación ejecuta código y con el cual se conecta a la base de datos |
| 9 | Usuario de base de datos | La cuenta de usuario con la cual se conecta a la base de datos para lectura y escritura |

Las dependencias externas representas elementos externos al código de programación que pueden representar una amenaza para la aplicación. Por lo general estos ítems siguen bajo el alcance de la organización, pero fuera del alcance del equipo de desarrollo. Ver Tabla 3

Tabla 5 Dependencias

|  |  |
| --- | --- |
| Dependencias Externas | |
| Id | Descripción |
| 1 | La aplicación correrá sobre un servidor apache, montado sobre Linux, con las actualizaciones de seguridad correspondientes |
| 2 | El gestor de base de datos será MySQL, corriendo sobre un servidor Linux, con las actualizaciones de seguridad correspondientes |
| 3 | La conexión entre el servidor web y la base de datos será a través de una red privada |

Si bien se puede realizar un refuerzo aplicando medidas de seguridad (también conocidas como hardening) de estos elementos, esa tarea depende de los encargados de implementar la aplicación en el servidor del INICNE y es difícil asegurar de que la correcta implementación de estas medidas en el ambiente de producción.

Los puntos de entrada son las interfaces por las que un atacante podría interactuar con la aplicación o enviarle datos. En este caso se los describió en forma de capas ya que todos dependen del puerto HTTP para poder acceder a la misma.

Tabla 6 Puntos de entrada

|  |  |  |  |
| --- | --- | --- | --- |
| Puntos de entrada | | | |
| Id | Nombre | Descripción | Nivel de confianza |
| 1 | Puerto HTTP | El acceso a la aplicación es a través de HTTP | 1. Usuarios anónimos 2. Usuarios con credenciales validas 3. Usuarios con credenciales invalidas 4. Administradores 5. asistente |
| 1.1 | Página de login | Es la página inicial, todos los usuarios deben loguearse en la aplicación para poder utilizarla | 1. Usuarios anónimos 2. Usuarios con credenciales validas 3. Usuarios con credenciales invalidas 4. Administradores 5. Asistente |
| 1.1.1 | Función de Login | Acepta las credenciales de los usuarios, las compara con la base de datos | 1. Usuarios con credenciales validas 2. Usuarios con credenciales invalidas 3. Administradores 4. Asistente |
| 1.2 | Formularios de búsqueda | Acepta las entradas otorgadas por los usuarios y las busca en la base de datos | 1. Usuarios con credenciales validas 2. Administradores 3. Asistente |
| 1.3 | Formulario de altas y modificaciones | Acepta las entradas otorgadas por los usuarios e impacta los cambios en la base de datos | 1. Administradores 2. asistente |

Definimos como activos a las áreas o ítems de la aplicación que podría interesar a un atacante y son la razón de una amenaza. En nuestro sistema disponemos de activos tanto físicos como abstractos, por ejemplo la información de los usuarios y sus contraseñas es un activo físico, pero la disponibilidad de la aplicación es abstracto.

Tabla 7 Activos

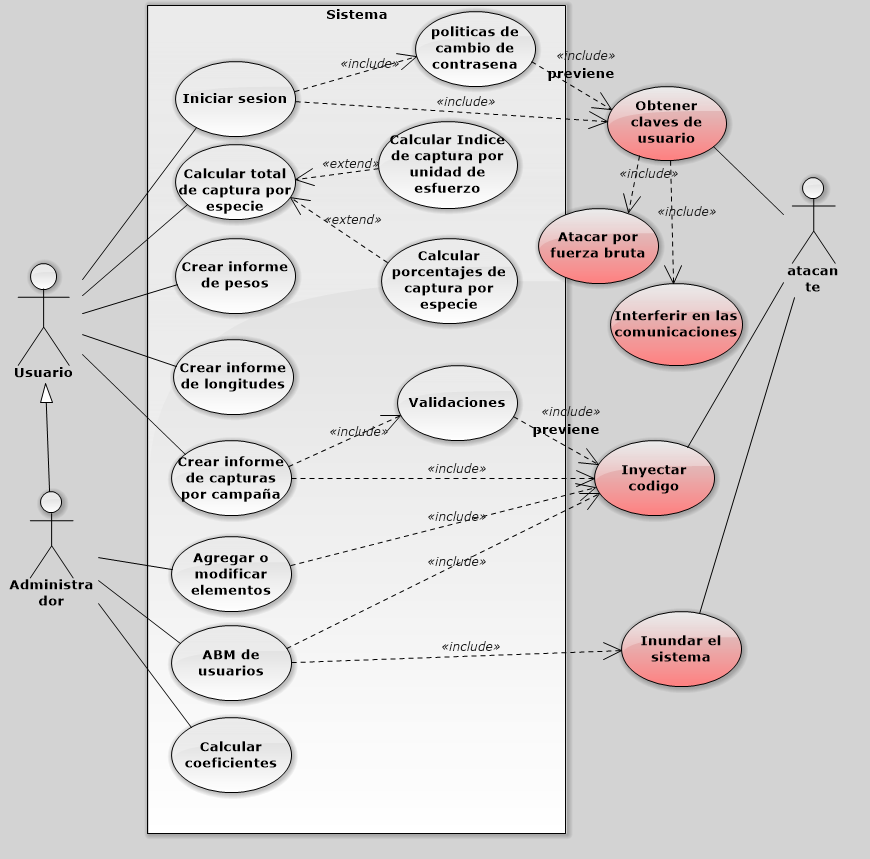
|  |  |  |  |
| --- | --- | --- | --- |
| Activos | | | |
| Id | Nombre | Descripción | Niveles de confianza |
| 1 | Usuarios y Administradores | Activos relacionados a Usuarios y Administradores |  |
| 1.1 | Login de usuario | Credenciales de inicio de sesión de usuarios e investigadores | (2) Usuario con credenciales validas (4) Administrador  (5) Administrador de base de datos (8) Usuario del servicio de servidor web (9) Usuario de base de datos |
| 1.2 | Login de Administrador | Credenciales de inicio de sesión de administradores | (4) Administrador  (6) Administrador de base de datos (8) Usuario del servicio de servidor web (9) Usuario de base de datos |
| 2 | Sistema | Activos relacionados al funcionamiento del sistema |  |
| 2.1 | Disponibilidad de la aplicación | La aplicación debería estar disponible todo el tiempo y por todos los usuarios autorizados | (6) Administrador de base de datos  (7) Administrador del sitio web |
| 2.2 | Habilidad de ejecutar código como el usuario del servicio del servidor web | Habilidad de ejecutar código fuente como el usuario del servicio del servidor web | (7) Administrador del sitio web  (8) Usuario del servicio de servidor web |
| 2.3 | Habilidad de ejecutar código SQL como usuario de base de datos | Habilidad de ejecutar código SQL en el sistema, pudiendo leer, escribir o modificar datos de la base de datos | (6) Administrador de base de datos  (9) Usuario de base de datos |
| 3 | Aplicación web | Activos relacionados a la aplicación |  |
| 3.1 | Login de sesión | Representa a una sesión activa en la aplicación, esta puede pertenecer a un usuario o administrador | (2) Usuario con credenciales validas (4) Administrador  (5) Asistente |
| 3.2 | Acceso a la base de datos | El acceso a la base de datos permite administrar la misma, accediendo a información de usuario y pertinente a las investigaciones | (6) Administrador de base de datos |
| 3.3 | Habilidad de crear usuarios | Permite a un individuo añadir nuevos usuarios a la aplicación, los cuales pueden ser usuarios o administradores | (4) Administrador  (7) Administrador del sitio web |
| 3.4 | Habilidad de añadir datos | Permite a un administrador agregar datos de campañas | (4) Administrador  (7) Administrador del sitio web  (5) Asistente |

Tanto los activos como puntos de entrada se encuentras relacionados con los usuarios que tienen contacto con los mismos lo que es importante a la hora de saber que usuarios podrían resultar afectados, o de donde puede provenir un posible ataque.

Conociendo cómo funciona la aplicación se puede modelizar las acciones que realizan los usuarios de la aplicación y cuáles son las amenazas a esas acciones. Para ello utilizamos una variación de los casos de uso llamados casos de abuso.

Aquí también podemos modelizar las acciones que se pueden tomar para mitigar ciertas amenazas.

Fig. 10 Casos de abuso



Para poder determinar las amenazas, primero fue necesario categorizarlas, por lo que se utilizó el método STRIDE []

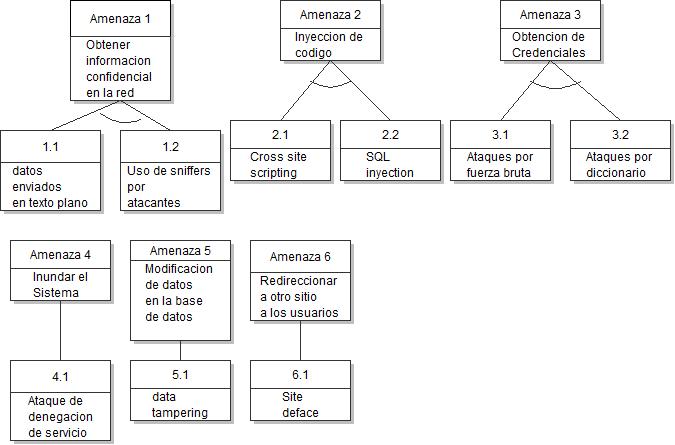
Fig. 11 Clasificación de amenazas con STRIDE

|  |  |  |
| --- | --- | --- |
| ID | Descripción | STRIDE |
| 1 | Un usuario no autorizado ve u obtiene información confidencial en la red | T,I |
| 2 | Un usuario ejecuta código que no está autorizado a hacerlo | S |
| 3 | Obtención o robo de credenciales | S,E |
| 4 | Un usuario deniega el acceso a la aplicación o base de datos inundando el sistema con peticiones | D |
| 5 | Modificación de datos directamente en la base de datos sin pasar por la aplicación | T |
| 6 | Un usuario modifica la aplicación para para redirigir a los usuarios a otro sitio | T |

Como resultado de la integración de las distintas etapas de diseño se obtuvieron los siguientes diagramas:

De las amenazas más comunes se analizó cuáles de sus variantes podrían afectar a la aplicación, esta es una etapa muy importante porque nos permite planificar qué medidas tomar para solventar las distintas amenazas en algunos casos, detectar nuevas amenazas.

Fig. 12 Árbol de amenazas



Una vez detectadas todas las amenazas las calificamos con DREAD, para determinar el factor técnico y el impacto se preguntas como: ¿qué tan grande seria el daño si un ataque tuviera éxito? Y si un ataque tuviese éxito, ¿qué porcentaje de los usuarios resultaría afectado? Correspondiendo a la primer D y a la A, mientras que para el factor de explotación se utilizan las preguntas ¿Qué tan fácil es reproducir un ataque?, ¿Qué tanto tiempo, trabajo y experiencia se necesita para llevar a cabo el ataque? Y ¿Qué tan fácil es para el atacante detectar la amenaza? Correspondientes a R, E y D respectivamente.

Fig. 13 Calificación de amenazas con DREAD

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| id | Amenaza | D | R | E | A | D | Total | Promedio |
| 1.1 | Robo de sesiones | 3 | 4 | 2 | 5 | 3 | 17 | 3.4 |
| 1.2 | Hombre en el medio | 2 | 2 | 1 | 4 | 5 | 14 | 2.8 |
| 1.3 | Petición de login no encriptada | 3 | 3 | 4 | 2 | 3 | 15 | 3 |
| 1.4 | Compartir sesiones | 3 | 5 | 5 | 4 | 6 | 21 | 4.2 |
| 2.1 | Cross site scripting | 6 | 5 | 7 | 8 | 5 | 31 | 6.2 |
| 2.1 | SQL inyección | 6 | 4 | 3 | 9 | 6 | 28 | 5.6 |
| 3.1 | Ataque por fuerza bruta | 4 | 2 | 4 | 5 | 3 | 18 | 3.6 |
| 3.2 | Ataque por diccionario | 5 | 4 | 4 | 6 | 4 | 23 | 4.6 |
| 4.1 | Ataque DOS | 3 | 3 | 2 | 6 | 2 | 16 | 3.2 |
| 5.1 | Data tampering | 5 | 7 | 3 | 3 | 5 | 23 | 4.6 |
| 6.1 | Site deface | 5 | 4 | 4 | 7 | 5 | 25 | 5 |

Así como ya existen vulnerabilidades genéricas para analizar en la mayoría de las aplicaciones, también existen contramedidas para las mismas que es bueno tenerlas en cuenta, en la Tabla 7 podemos ver algunas de las contramedidas más comunes, categorizadas según STRIDE.

Tabla 8: Técnicas de mitigación

|  |  |
| --- | --- |
| Amenazas STRIDE y contramedidas | |
| Suplantación | 1. Autenticación apropiada 2. Proteger información sensible |
| Manipulación | 1. Medidas de autenticación 2. Uso de hashes 3. Uso de protocolos resistentes a manipulación |
| Repudio | 1. Timestamps |
| Revelación de información | 1. Autorización 2. Encriptación |
| Denegación de servicio | 1. Medidas de autenticación 2. Autorización adecuada 3. Filtro de datos |
| Elevación de privilegios | 1. Ejecutar con privilegios mínimos |

Una vez finalizado el análisis y teniendo en cuenta los resultados de DREAD podemos determinar qué tan factible es la explotación de una amenaza, y si vale la pena invertir tiempo en tratar de mitigar una vulnerabilidad, porque puede que esta nunca ocurra, o que sus efectos no sean relevantes para el funcionamiento de la aplicación, las estrategias de mitigación son:

1. **No hacer nada:** no realizar ninguna acción y esperar que no cause daños
2. **Informar el riesgo:** Informar al usuario de la existencia del riesgo
3. **Mitigar el riesgo:** utilizar contramedidas
4. **Aceptar el riesgo:** actuar luego de evaluar el impacto
5. **Transferir el riesgo:** transferir el riesgo a un tercero
6. **Terminar el riesgo:** apagar, desconectar o desenchufar el active que causa problemas

Fig. 14 Estrategia de mitigación adoptada

|  |  |  |
| --- | --- | --- |
| id | Amenaza | Estrategia |
| 1.1 | Robo de sesiones | Mitigar |
| 1.2 | Hombre en el medio | Aceptar el riesgo |
| 1.3 | Petición de login no encriptada | Mitigar |
| 1.4 | Compartir sesiones de usuario | Informar del riesgo |
| 2.1 | Cross site scripting | Mitigar |
| 2.1 | SQL inyección | Mitigar |
| 3.1 | Ataque por fuerza bruta | Aceptar el riesgo |
| 3.2 | Ataque por diccionario | Aceptar el riesgo |
| 4.1 | Ataque DOS | No hacer nada |
| 5.1 | Data tampering | Mitigar |
| 6.1 | Site deface | Mitigar |

Falta completar y describir la segunda iteración y arreglar la primera

Capítulo 3. Herramientas y lenguajes de programación.

En este capítulo se describen las distintas herramientas utilizadas en la elaboración de la aplicación abarcando desde la lectura de los documentos provistos por el personal de INICNE, las herramientas y lenguajes usados en las distintas etapas del ciclo de vida del software como también herramientas auxiliares que facilitan tareas a al momento de codificar.

Se agrupan las herramientas en diferentes categorías dependiendo del fin para el cual fueron utilizadas. En una primera categoría se nombran los lenguajes de programación utilizados incluyendo los frameworks y librerías más importantes utilizadas. En la segunda se incluyen las herramientas utilizadas para la manipulación y almacenamiento de datos, luego se sigue con las herramientas auxiliares que ayudan a mantener un control de versiones de código y manejadores de dependencias, y por último se agrupan las herramientas adicionales como entornos e IDE de desarrollo, y herramientas utilizadas en las etapas de análisis y diseño.

Lenguajes de programación

PHP es un lenguaje de programación del lado de servidor o backend, utilizado para desarrollo web de contenido dinámico, su funcionamiento se basa en un intérprete PHP que recibe una solicitud de procesamiento de un script enviado por un cliente, el intérprete ejecuta el script, genera contenido de forma dinámica y se lo envía al cliente.

PHP utiliza una licencia PHP License v3.01.

En este trabajo se lo utilizo bajo el framework Codeigniter, para dar instrucciones al servidor y comunicarse con el motor de base de datos con el objetivo crear API REST [22] que alimentan a la aplicación.

JavaScript tuvo sus comienzos como parte del navegador Netscape para ayudar a los desarrolladores y diseñadores a crear applets sin precisar de conocimientos de Java, por lo que se crea un lenguaje de script interpretado, débilmente tipado y dinámico que actualmente tomo mucho más relevancia que en sus principios, ya no está limitado únicamente a navegadores web, sino que también se lo utiliza para funcionar del lado del servidor con frameworks como NodeJS y aplicaciones de escritorio con Electron.js entre otros.

Aquí se lo utilizo del lado del cliente con dos librerías: AngularJS como framework y se añadieron funcionalidades Jquery, ambas librerías bajo licencias MIT. Esto permitió que la aplicación funcione dinámicamente utilizando AJAX y evitando refrescos de páginas innecesarios.

HTML es un lenguaje de marcado de hipertexto, utilizado para dar estructura a los sitios web y sus elementos, para este trabajo se utilizó su versión 5 conocida como HTML5.

CSS es el lenguaje utilizado para describir la presentación de las páginas web, incluyendo colores, diseño y fuentes. Permite adaptar la presentación a distintos tipos de dispositivos y dispositivos con distintos tamaños de pantalla. Aquí se utilizó CSS3, con ayuda de un framework conocido como Bootstrap, el cual utiliza licencia MIT.

Gestión de datos

MySQL uno de los sistemas de gestión de base de datos relacionales open source más utilizados del mundo y uno de los más populares junto a Oracle y Microsoft SQL Server, y es uno de los componentes fundamentales en ambientes de desarrollo XAMPP, LAMP y WAMP.

MySQL Workbench es una herramienta visual de diseño de bases de datos que integra desarrollo de software, Administración de bases de datos, diseño de bases de datos, creación y mantenimiento para el sistema de base de datos [MySQL](https://es.wikipedia.org/wiki/MySQL).

Se escogió MySQL como motor de base de datos y se utilizó MySQL Workbench para modelar, diseñar, generar procedimientos y funciones almacenadas y realizar los cambios por medio de una interfaz gráfica, ambos utilizan licencias GPL.

Pentaho Data Integración, también llamada kettle, es un componente de la suit de herramientas Pentaho responsable del proceso de extracción, transformación y carga, utiliza una licencia Apache, Versión 2.0.

Si bien los procesos ETL se utilizan principalmente en ambientes de data Warehouse, aquí se lo utilizo para realizar la migración de datos de las planillas Excel hacia la base de datos.

Gestión de la configuración

Debido a los cambios contantes en el código de la aplicación, es importante llevar un seguimiento de los cambios realizados y resulta muy ventajoso poder volver a una versión anterior de la aplicación en caso de que un cambio realizado no funcione correctamente. Para eso se utilizó Git, con licencia GPL versión 2.

Npm es un manejador de paquetes para entornos JavaScript bajo Artistic License 2.0 que permite compartir y reutilizar código y librerías, en este proyecto se lo utilizo como manejador de dependencias, y como repositorio para descargar las librerías JavaScript utilizadas.

Otras herramientas

XAMPP: es un paquete de aplicaciones para el entorno de Windows, incluye Apache, MySQL, PHP y Perl. El nombre viene del acrónimo de X (multiplataforma) y las siglas de los paquetes que incluyen mencionados anteriormente.

Como entorno de desarrollo de PHP se utilizó XAMPP, bajo licencia GPL, también se lo utilizo como servidor de MySQL

Al momento de codificar, si bien al ser php y JavaScript lenguajes interpretados se puede utilizar cualquier editor de texto, se optó por Atom.io por su facilidad para manjar distintos lenguajes, resaltar sus sintaxis y marcas, y su facilidad para vincular con repositorios para poder seguir su versionado.

Microsoft Excel, Se utilizó para la lectura de datos y gráficos almacenados en hojas de cálculo y planillas elaboradas por los investigadores

Por último, existen muchas formas de anotar las historias de usuario, la más práctica se basa en la escritura simple y con lenguaje natural sobre Post-it, pero para este trabajo se utilizó una aplicación llamada Easybacklog, la cual posee una estructura base sobre la cual se escriben las historias, y luego son exportadas a una planilla Excel.

Capítulo 4. Resultados.

Una vez finalizados todos los Sprints el resultado es un conjunto de documentos que contiene la versión final de todos los documentos, más sus versiones anteriores.

Con cada sprint a los se les añade las funcionalidades tratadas en este con todo lo que estas implican, considerando también la posibilidad de cambios en los requerimientos o cambios en las historias de usuario.

En XP si bien un sprint puede tener la misma duración que una iteración de una metodología incremental, la forma de trabajar es muy diferente. Mientras que en las metodologías incrementales se toma un conjunto de funcionalidades para ser implementadas, y se las considera como un todo, realizando sobre todas ellas un análisis, diseño, codificación y pruebas, al igual que la metodología cascada, pero realizando entregas parciales que llevan menos tiempo que el sistema completo. En cambio en XP se toma a cada tarea y sobre ella se realiza el análisis, diseño, pruebas y codificación, obteniendo mayor cantidad iteraciones mucho más cortas y se realiza integración continua, lo que significa que a medida se completa una tarea, esta debe ser incorporada al sistema para formar parte de un sistema potencialmente funcional permitiendo entregas más rápidas.

Aquí se elaboraron las historias de usuario con ayuda del cliente (responsable del INICNE) y se realizaron entrevistas para clarificar cuestiones que no quedaron del todo claro en las historias de usuario creadas inicialmente no obstante las mismas no sufrieron modificaciones significativas

/////// agregar **password\_hash en php** algoritmo bcrypt, y decripta con **password\_verify**

tiempos constants, evitar timing attack

Tabla 2 Historias de usuario

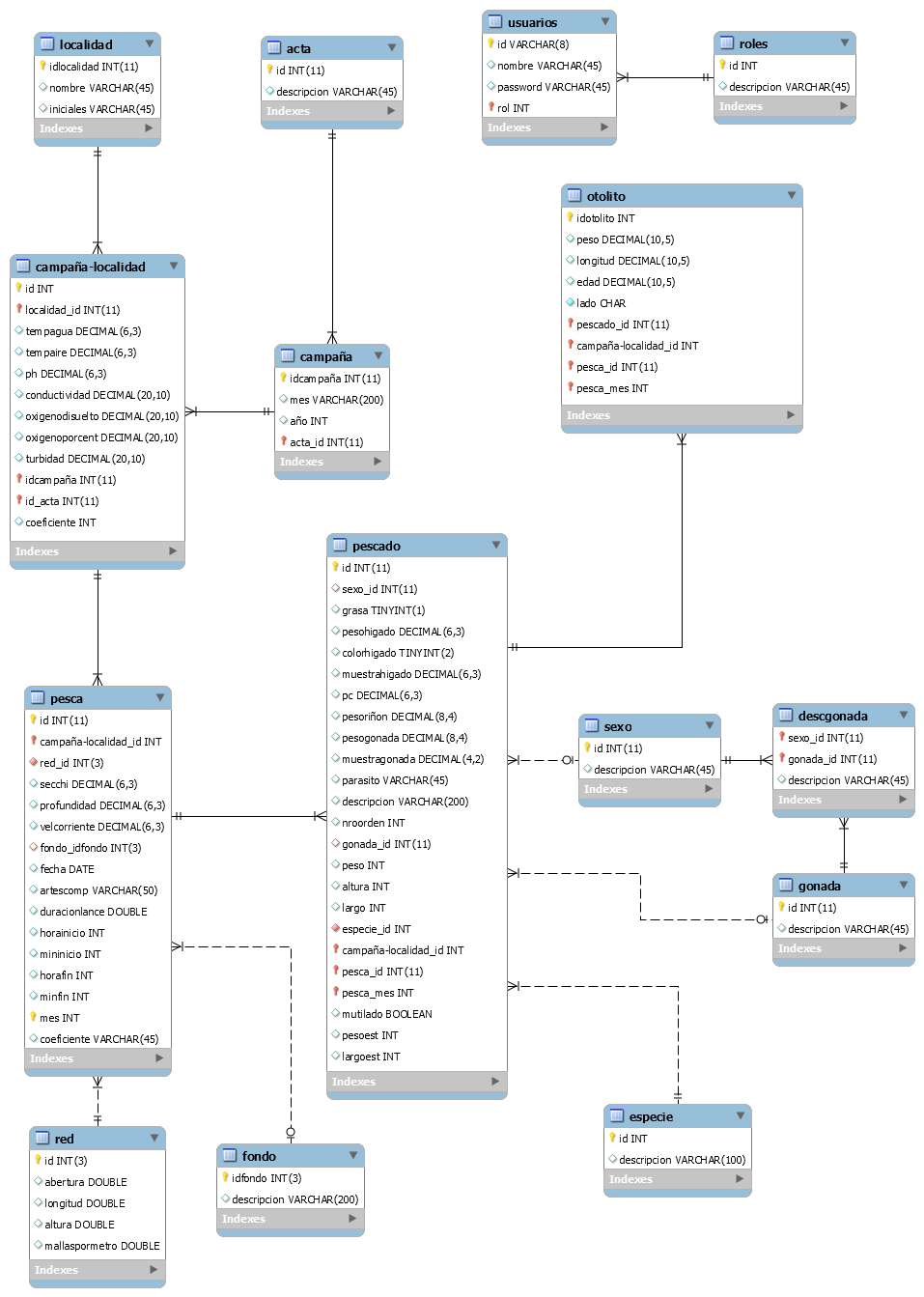


~~Una vez obtenidas las historias de usuario, se analizó el documento Excel que utilizaban para resguardar los datos, como resultado se obtuvo el modelo de datos representado en Fig. 6~~

A lo largo de las iteraciones se modificó el modelo de datos y se fueron añadiendo entidades y atributos a medida que eran necesarios hasta obtener una versión que cumpla con los requisitos (ver Fig. 6).

Por otra parte, también se adicionaron tablas que si bien no son necesarias para el funcionamiento de la aplicación, ayudan al mantenimiento y auditoria, como también a mantener la aplicación segura.

Fig. 8 Modelo de datos

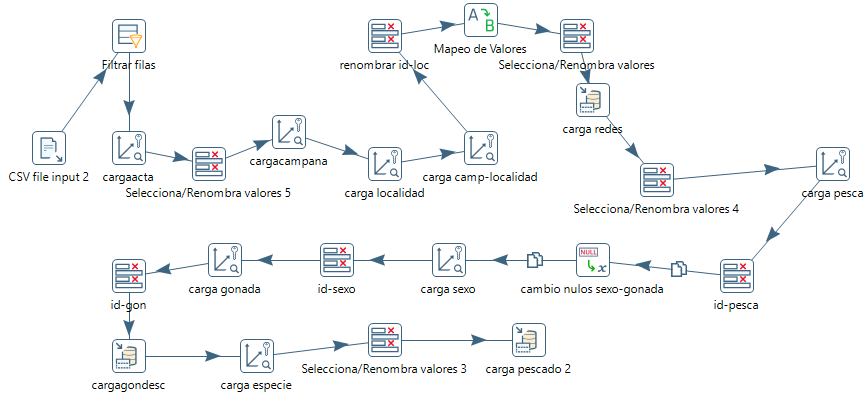


~~Debido a la existencia de datos en tablas Excel y considerando la dificultad que esto presenta para compartir y utilizarlos de forma concurrente los datos entre distintos usuarios se optaron por realizar un proceso ETL en el cual se migraron los datos de tablas a una base de datos, ver Fig. 13~~ ////////METODOLOGIA

Una vez obtenido el modelo de datos final, se prosiguió al análisis de las planillas Excel para diseñar el proceso ETL, se inició por las tablas con menos dependencias de la base de datos, y a medida que están se terminaban de cargar, se pasaba a las que tenían más dependencias.

Una de las ventajas de la herramienta utilizada para este proceso es que nos permite trabajar como si estuviésemos utilizando almacenes de datos lo que facilito el uso de claves foráneas.

Fig. 9 Proceso ETL



~~Se aplicó un análisis de amenazas basado en la guía de modelado de amenazas de OWASP [18] la cual inicia con una descripción del sistema, avanzando por elementos que podría identificar un posible atacante, y los clasifica utilizando los métodos STRIDE y DREAD~~

Basado en OWASP [18] se realizó el modelado de amenazas, iniciando por una descripción de la aplicación sobre la cual estamos trabajando, como podemos ver en Tabla 2.

Tabla 3 Análisis de amenazas

|  |  |
| --- | --- |
| Análisis de amenazas | |
| Versión de aplicación | 2.0 |
| Descripción | Sitio web para investigadores de INICNE.  Los usuarios de la aplicación serán:   1. Administrador 2. Usuario autenticado 3. Asistente   Un administrador podrá crear, modificar y eliminar usuarios, agregar, modificar y leer datos de aplicación mientras que el usuario registrado solo podrá acceder a la información. El asistente solo podrá agregar, modificar y leer datos |
| Participantes | Silva Pablo |
| Revisado por | Bosco Francisco |

Seguidos por un análisis de niveles de confianza, los cuales representan el nivel de acceso que tendrán los agentes externos a la aplicación que serán cruzados con los puntos de entrada y los activos que posee la misma.

Tabla 4 Niveles de confianza

|  |  |  |
| --- | --- | --- |
| Niveles de confianza | | |
| Id | Nombre | Descripción |
| 1 | Usuario anónimo | Un usuario que se conectó a la aplicación, pero no inicio sesión |
| 2 | Usuario con credenciales validas | Usuario que se conectó a la aplicación, e inicio sesión con credenciales validas |
| 3 | Usuario sin credenciales validas | Usuario que intentan iniciar sesión con credenciales no validas |
| 4 | Administrador | Puede añadir nuevos usuarios, agregar datos de nuevas campañas |
| 5 | Asistente | Puede agregar y actualizar datos de las campañas |
| 6 | Administrador de base de datos | Tiene permisos de lectura y escritura en la base de datos que utiliza la aplicación |
| 7 | Administrador del sitio web | Puede cambiar la configuración |
| 8 | Usuario del servicio de servidor web | Es el usuario con el cual la aplicación ejecuta código y con el cual se conecta a la base de datos |
| 9 | Usuario de base de datos | La cuenta de usuario con la cual se conecta a la base de datos para lectura y escritura |

Las dependencias externas representas elementos externos al código de programación que pueden representar una amenaza para la aplicación. Por lo general estos ítems siguen bajo el alcance de la organización, pero fuera del alcance del equipo de desarrollo. Ver Tabla 3

Tabla 5 Dependencias

|  |  |
| --- | --- |
| Dependencias Externas | |
| Id | Descripción |
| 1 | La aplicación correrá sobre un servidor apache, montado sobre Linux, con las actualizaciones de seguridad correspondientes |
| 2 | El gestor de base de datos será MySQL, corriendo sobre un servidor Linux, con las actualizaciones de seguridad correspondientes |
| 3 | La conexión entre el servidor web y la base de datos será a través de una red privada |

Si bien se puede realizar un refuerzo aplicando medidas de seguridad (también conocidas como hardening) de estos elementos, esa tarea depende de los encargados de implementar la aplicación en el servidor del INICNE y es difícil asegurar de que la correcta implementación de estas medidas en el ambiente de producción.

Los puntos de entrada son las interfaces por las que un atacante podría interactuar con la aplicación o enviarle datos. En este caso se los describió en forma de capas ya que todos dependen del puerto HTTP para poder acceder a la misma.

Tabla 6 Puntos de entrada

|  |  |  |  |
| --- | --- | --- | --- |
| Puntos de entrada | | | |
| Id | Nombre | Descripción | Nivel de confianza |
| 1 | Puerto HTTP | El acceso a la aplicación es a través de HTTP | 1. Usuarios anónimos 2. Usuarios con credenciales validas 3. Usuarios con credenciales invalidas 4. Administradores 5. asistente |
| 1.1 | Página de login | Es la página inicial, todos los usuarios deben loguearse en la aplicación para poder utilizarla | 1. Usuarios anónimos 2. Usuarios con credenciales validas 3. Usuarios con credenciales invalidas 4. Administradores 5. Asistente |
| 1.1.1 | Función de Login | Acepta las credenciales de los usuarios, las compara con la base de datos | 1. Usuarios con credenciales validas 2. Usuarios con credenciales invalidas 3. Administradores 4. Asistente |
| 1.2 | Formularios de búsqueda | Acepta las entradas otorgadas por los usuarios y las busca en la base de datos | 1. Usuarios con credenciales validas 2. Administradores 3. Asistente |
| 1.3 | Formulario de altas y modificaciones | Acepta las entradas otorgadas por los usuarios e impacta los cambios en la base de datos | 1. Administradores 2. asistente |

Definimos como activos a las áreas o ítems de la aplicación que podría interesar a un atacante y son la razón de una amenaza. En nuestro sistema disponemos de activos tanto físicos como abstractos, por ejemplo la información de los usuarios y sus contraseñas es un activo físico, pero la disponibilidad de la aplicación es abstracto.

Tabla 7 Activos

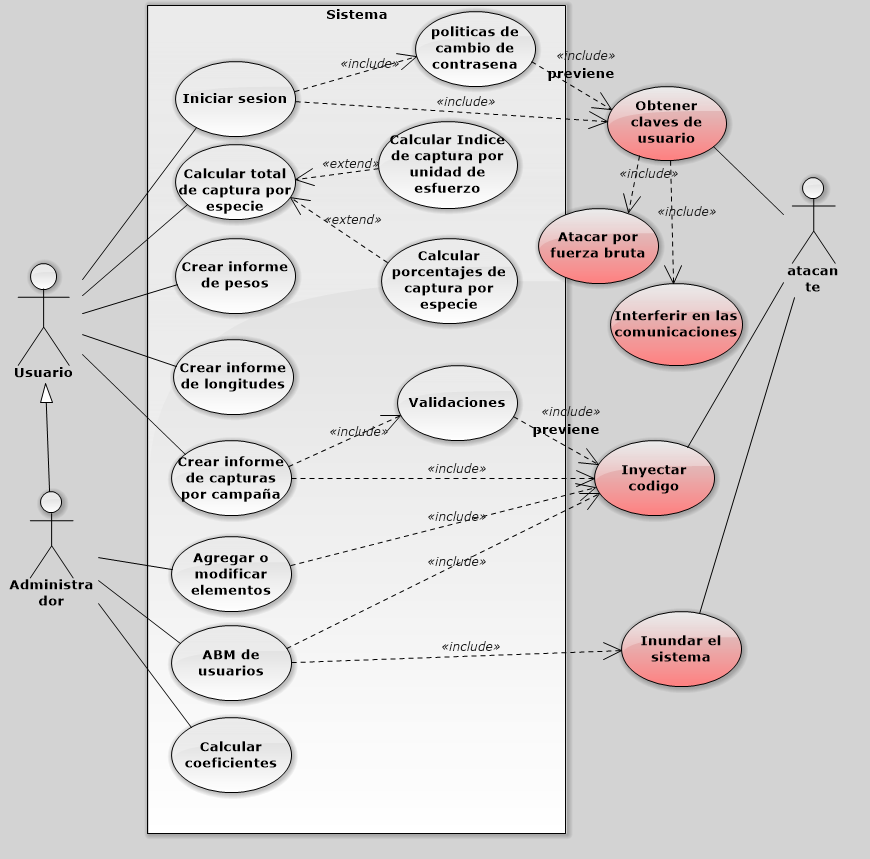
|  |  |  |  |
| --- | --- | --- | --- |
| Activos | | | |
| Id | Nombre | Descripción | Niveles de confianza |
| 1 | Usuarios y Administradores | Activos relacionados a Usuarios y Administradores |  |
| 1.1 | Login de usuario | Credenciales de inicio de sesión de usuarios e investigadores | (2) Usuario con credenciales validas (4) Administrador  (5) Administrador de base de datos (8) Usuario del servicio de servidor web (9) Usuario de base de datos |
| 1.2 | Login de Administrador | Credenciales de inicio de sesión de administradores | (4) Administrador  (6) Administrador de base de datos (8) Usuario del servicio de servidor web (9) Usuario de base de datos |
| 2 | Sistema | Activos relacionados al funcionamiento del sistema |  |
| 2.1 | Disponibilidad de la aplicación | La aplicación debería estar disponible todo el tiempo y por todos los usuarios autorizados | (6) Administrador de base de datos  (7) Administrador del sitio web |
| 2.2 | Habilidad de ejecutar código como el usuario del servicio del servidor web | Habilidad de ejecutar código fuente como el usuario del servicio del servidor web | (7) Administrador del sitio web  (8) Usuario del servicio de servidor web |
| 2.3 | Habilidad de ejecutar código SQL como usuario de base de datos | Habilidad de ejecutar código SQL en el sistema, pudiendo leer, escribir o modificar datos de la base de datos | (6) Administrador de base de datos  (9) Usuario de base de datos |
| 3 | Aplicación web | Activos relacionados a la aplicación |  |
| 3.1 | Login de sesión | Representa a una sesión activa en la aplicación, esta puede pertenecer a un usuario o administrador | (2) Usuario con credenciales validas (4) Administrador  (5) Asistente |
| 3.2 | Acceso a la base de datos | El acceso a la base de datos permite administrar la misma, accediendo a información de usuario y pertinente a las investigaciones | (6) Administrador de base de datos |
| 3.3 | Habilidad de crear usuarios | Permite a un individuo añadir nuevos usuarios a la aplicación, los cuales pueden ser usuarios o administradores | (4) Administrador  (7) Administrador del sitio web |
| 3.4 | Habilidad de añadir datos | Permite a un administrador agregar datos de campañas | (4) Administrador  (7) Administrador del sitio web  (5) Asistente |

Tanto los activos como puntos de entrada se encuentras relacionados con los usuarios que tienen contacto con los mismos lo que es importante a la hora de saber que usuarios podrían resultar afectados, o de donde puede provenir un posible ataque.

Conociendo cómo funciona la aplicación se puede modelizar las acciones que realizan los usuarios de la aplicación y cuáles son las amenazas a esas acciones. Para ello utilizamos una variación de los casos de uso llamados casos de abuso.

Aquí también podemos modelizar las acciones que se pueden tomar para mitigar ciertas amenazas.

Fig. 10 Casos de abuso



Para poder determinar las amenazas, primero fue necesario categorizarlas, por lo que se utilizó el método STRIDE []

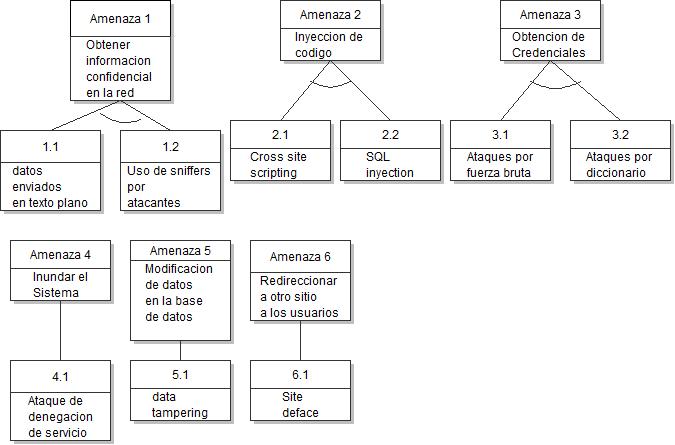
Fig. 11 Clasificación de amenazas con STRIDE

|  |  |  |
| --- | --- | --- |
| ID | Descripción | STRIDE |
| 1 | Un usuario no autorizado ve u obtiene información confidencial en la red | T,I |
| 2 | Un usuario ejecuta código que no está autorizado a hacerlo | S |
| 3 | Obtención o robo de credenciales | S,E |
| 4 | Un usuario deniega el acceso a la aplicación o base de datos inundando el sistema con peticiones | D |
| 5 | Modificación de datos directamente en la base de datos sin pasar por la aplicación | T |
| 6 | Un usuario modifica la aplicación para para redirigir a los usuarios a otro sitio | T |

Como resultado de la integración de las distintas etapas de diseño se obtuvieron los siguientes diagramas:

De las amenazas más comunes se analizó cuáles de sus variantes podrían afectar a la aplicación, esta es una etapa muy importante porque nos permite planificar qué medidas tomar para solventar las distintas amenazas en algunos casos, detectar nuevas amenazas.

Fig. 12 Árbol de amenazas



Una vez detectadas todas las amenazas las calificamos con DREAD, para determinar el factor técnico y el impacto se preguntas como: ¿qué tan grande seria el daño si un ataque tuviera éxito? Y si un ataque tuviese éxito, ¿qué porcentaje de los usuarios resultaría afectado? Correspondiendo a la primer D y a la A, mientras que para el factor de explotación se utilizan las preguntas ¿Qué tan fácil es reproducir un ataque?, ¿Qué tanto tiempo, trabajo y experiencia se necesita para llevar a cabo el ataque? Y ¿Qué tan fácil es para el atacante detectar la amenaza? Correspondientes a R, E y D respectivamente.

Fig. 13 Calificación de amenazas con DREAD

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| id | Amenaza | D | R | E | A | D | Total | Promedio |
| 1.1 | Robo de sesiones | 3 | 4 | 2 | 5 | 3 | 17 | 3.4 |
| 1.2 | Hombre en el medio | 2 | 2 | 1 | 4 | 5 | 14 | 2.8 |
| 1.3 | Petición de login no encriptada | 3 | 3 | 4 | 2 | 3 | 15 | 3 |
| 1.4 | Compartir sesiones | 3 | 5 | 5 | 4 | 6 | 21 | 4.2 |
| 2.1 | Cross site scripting | 6 | 5 | 7 | 8 | 5 | 31 | 6.2 |
| 2.1 | SQL inyección | 6 | 4 | 3 | 9 | 6 | 28 | 5.6 |
| 3.1 | Ataque por fuerza bruta | 4 | 2 | 4 | 5 | 3 | 18 | 3.6 |
| 3.2 | Ataque por diccionario | 5 | 4 | 4 | 6 | 4 | 23 | 4.6 |
| 4.1 | Ataque DOS | 3 | 3 | 2 | 6 | 2 | 16 | 3.2 |
| 5.1 | Data tampering | 5 | 7 | 3 | 3 | 5 | 23 | 4.6 |
| 6.1 | Site deface | 5 | 4 | 4 | 7 | 5 | 25 | 5 |

Así como ya existen vulnerabilidades genéricas para analizar en la mayoría de las aplicaciones, también existen contramedidas para las mismas que es bueno tenerlas en cuenta, en la Tabla 7 podemos ver algunas de las contramedidas más comunes, categorizadas según STRIDE.

Tabla 8: Técnicas de mitigación

|  |  |
| --- | --- |
| Amenazas STRIDE y contramedidas | |
| Suplantación | 1. Autenticación apropiada 2. Proteger información sensible |
| Manipulación | 1. Medidas de autenticación 2. Uso de hashes 3. Uso de protocolos resistentes a manipulación |
| Repudio | 1. Timestamps |
| Revelación de información | 1. Autorización 2. Encriptación |
| Denegación de servicio | 1. Medidas de autenticación 2. Autorización adecuada 3. Filtro de datos |
| Elevación de privilegios | 1. Ejecutar con privilegios mínimos |

Una vez finalizado el análisis y teniendo en cuenta los resultados de DREAD podemos determinar qué tan factible es la explotación de una amenaza, y si vale la pena invertir tiempo en tratar de mitigar una vulnerabilidad, porque puede que esta nunca ocurra, o que sus efectos no sean relevantes para el funcionamiento de la aplicación, las estrategias de mitigación son:

1. **No hacer nada:** no realizar ninguna acción y esperar que no cause daños
2. **Informar el riesgo:** Informar al usuario de la existencia del riesgo
3. **Mitigar el riesgo:** utilizar contramedidas
4. **Aceptar el riesgo:** actuar luego de evaluar el impacto
5. **Transferir el riesgo:** transferir el riesgo a un tercero
6. **Terminar el riesgo:** apagar, desconectar o desenchufar el active que causa problemas

Fig. 14 Estrategia de mitigación adoptada

|  |  |  |
| --- | --- | --- |
| id | Amenaza | Estrategia |
| 1.1 | Robo de sesiones | Mitigar |
| 1.2 | Hombre en el medio | Aceptar el riesgo |
| 1.3 | Petición de login no encriptada | Mitigar |
| 1.4 | Compartir sesiones de usuario | Informar del riesgo |
| 2.1 | Cross site scripting | Mitigar |
| 2.1 | SQL inyección | Mitigar |
| 3.1 | Ataque por fuerza bruta | Aceptar el riesgo |
| 3.2 | Ataque por diccionario | Aceptar el riesgo |
| 4.1 | Ataque DOS | No hacer nada |
| 5.1 | Data tampering | Mitigar |
| 6.1 | Site deface | Mitigar |

Falta testing

Capítulo 5.

Conclusiones y futuros trabajos

Trabajo futuro: agregar control de ip para evitar ataques de fuerza bruta provenientes de la misma ip y dificultar ataques DOS

# 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, «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] | OWASP, «Application Threat Modeling,» 8 Marzo 2015. [En línea]. Available: https://www.owasp.org/index.php/Application\_Threat\_Modeling#Assets. [Último acceso: 20 07 2016]. |
| [19] | M. I. Daud, «Secure Software Development Model: A Guide for Secure Software Life Cycle,» *Proceedings of the international MultiConference of Engineers and Computer Scientists,* vol. 1, pp. 17-19, 2010. |
| [20] | M. Howard y D. LeBlanc, Writing Secure Code: Practical Strategies and Proven Techniques for Building Secure Applications in a Networked World (Developer Best Practices), Microsoft Press, 2004. |
| [21] | 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. |
| [22] | C. Wells, «A Weak Foundation,» de *Securing Ajax Applications*, O’Reilly, 2007, pp. 152-154. |

Anexos.