

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 2017

Agradecimientos

Quiero agradecer principalmente a mis padres Pablo y Nora por darme la oportunidad de estudiar, por de chico dejarme jugar con sus herramientas de trabajo (su computadora y su calculadora programable) de donde adquirí mis primeros conocimientos de informática y el gusto por la misma, pero principalmente por estar siempre, junto con mis hermanas motivándome, apoyando y a mis decisiones.

A mis profesores orientadores y docentes de la catedra por ayudar en el desarrollo de la aplicación, a superar dificultades durante el mismo y por su ayuda en la confección de este documento.

Un especial agradecimiento a mi Hermana Natalia y a Sebastián Sánchez del INICNE por abrir las puertas del instituto, brindar su tiempo para explicar sus problemas y los datos para trabajar.

A mis amigos Maximiliano Céspedes, Manuel Saber. Fernando Milan y Emiliano Alegre por estar siempre alimentando mi curiosidad, proponiendo proyectos nuevos ayudando a aprender más cada día.

A mis compañeros y amigos Agustín Meza, Mariano López y Martin Rodríguez por acompañarme durante las etapas desarrollo de este trabajo, por asistir con aspectos técnicos de las mismas y ayudar a definir parte de la metodología utilizada.

Y Por último pero no menos importante a mis amigos Maximiliano Céspedes y Silvana Astrada por darme confianza, superar los miedos y ayudar a formar mi carácter en el momento que más necesitaba. Ellos me ayudaron a ver muchos errores y entender que estos son oportunidades para mejorar, y solamente son malos si los repetimos. Ciertamente nada de esto hubiese sido posible sin su ayuda.

Contenido

[Agradecimientos 2](#_Toc497664919)

[Capítulo 1. Introducción. 4](#_Toc497664920)

[1.1 Breve estado del arte. 4](#_Toc497664921)

[1.1.1 Programación web 4](#_Toc497664922)

[1.1.2 Seguridad Informática 5](#_Toc497664923)

[1.2 Objetivos 11](#_Toc497664924)

[1.3 Fundamentación 12](#_Toc497664925)

[Capítulo 2. Metodología. 14](#_Toc497664926)

[2.1 Fase 1 revisión sistemática de la literatura 14](#_Toc497664927)

[2.2 Fase 2. Aplicación de ciclo de vida 14](#_Toc497664928)

[2.2.1 Primera iteración: 17](#_Toc497664929)

[2.2.2 Segunda iteración 27](#_Toc497664930)

[Capítulo 3. Herramientas y lenguajes de programación. 29](#_Toc497664931)

[3.1 Lenguajes de programación 29](#_Toc497664932)

[3.2 Gestión de datos 30](#_Toc497664933)

[3.3 Gestión de la configuración 31](#_Toc497664934)

[3.4 Otras herramientas 31](#_Toc497664935)

[Capítulo 4. Resultados 33](#_Toc497664936)

[4.1. Documentación 33](#_Toc497664937)

[4.2 Funcionamiento del Software 49](#_Toc497664938)

[4.3 Medidas preventivas 57](#_Toc497664939)

[4.3.1 Niveles de acceso 57](#_Toc497664940)

[4.3.2 Login 58](#_Toc497664941)

[4.3.3 Sesiones 59](#_Toc497664942)

[4.3.4 Control de entradas 59](#_Toc497664943)

[4.3.5 Control de salidas 59](#_Toc497664944)

[4.3.6 Menú de altas, bajas y modificaciones 60](#_Toc497664945)

[4.4 Respuesta del software ante ataques 61](#_Toc497664946)

[Capítulo 5. Conclusiones y futuros trabajos 70](#_Toc497664947)

[Referencias 71](#_Toc497664948)

Capítulo 1. Introducción.

1.1 Breve estado del arte.

1.1.1 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.

1.1.2 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.

1.1.2.1 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.

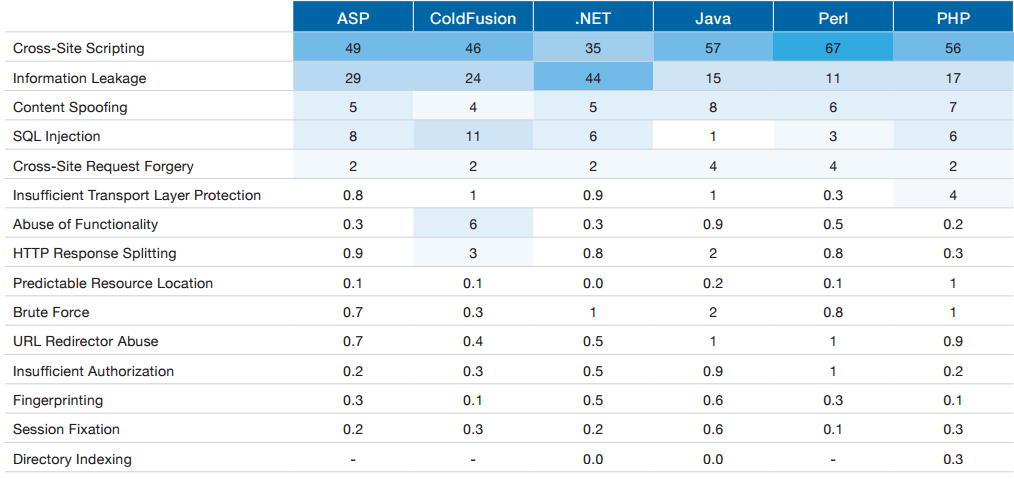


Fig. 1: porcentaje de vulnerabilidad por lenguaje [13]

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.

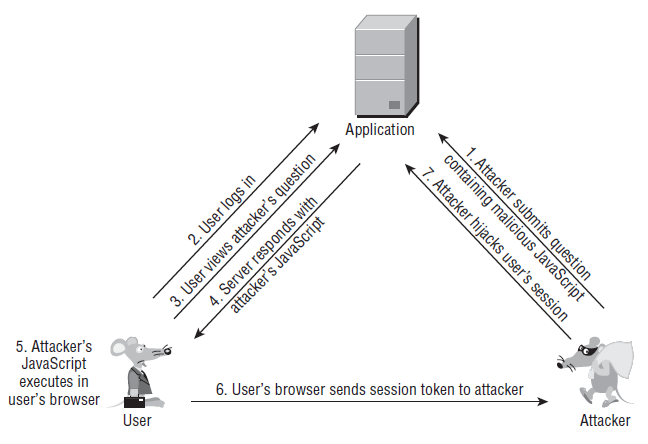


Fig. 2: XSS almacenados [11]

***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.

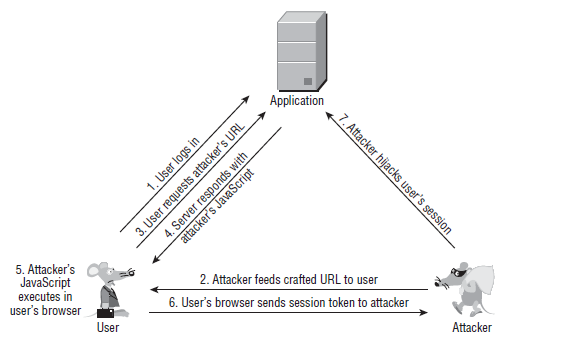


Fig. 3 XSS reflejados [11]

***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.

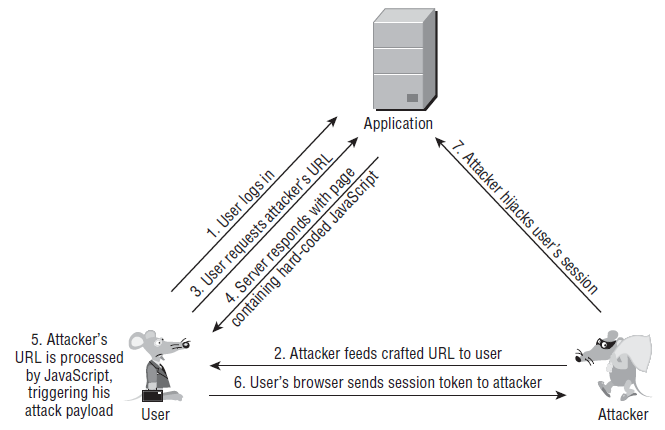


Fig. 4: XSS basados en DOM [11]

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

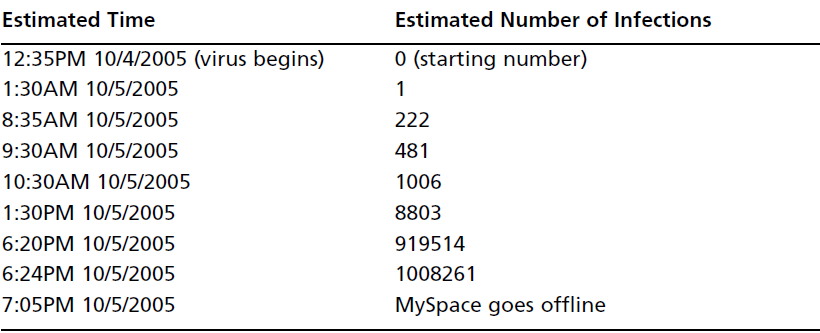


Fig. 5: Estimado de infecciones de Samy [12]

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].

1.1.2.1 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 sí 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 más visitados de internet fuese fuera de línea, pero con otras intenciones podría tomarse el control de múltiples 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 [11] 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 otra librería del sitio.

1.1.2.3 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 [17].

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] [18] [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 [19] y [20], las cuales usan métodos para detectar categorizar amenazas como STRIDE [21] y DREAD [15].

1.2 Objetivos

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.

1.3 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 más importante que la representación de los datos es su seguridad e integridad y, dado que por motivos de accesibilidad geográfica una de las formas más 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.

2.1 Fase 1 revisión sistemática de la literatura

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.

2.2 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 [20] y OWASP [19], añadiendo un análisis y clasificación de amenazas utilizando los métodos STRIDE [21] 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 limitó a la utilización de un servidor de versionado (git), al 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 pruebas se lo añadía al servidor unificando la rama maestra con las funcionalidades
* Con respecto a la planeación se optó 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 más 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 limitó a realizar pruebas de inyección de código JavaScript y SQL, como la aplicación responde ante código inyectado (almacenado) y debido a que los ataques XSS frecuentemente van de la mano con los ataques CSRF, también se harán pruebas de cómo responde ante un supuesto robo de sesión.

También 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 refactorización 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 más por lo que se estableció:

* Utilizar la estructura de directorios propuesta por CodeIgniter (framework PHP)
* Utilizar un controlador 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 controlador 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 optó 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 más funcionalidad y autenticación, se esperó a que realicen una prueba de funcionamiento y luego se solicitó 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 primera iteración:

Previo al primer Sprint se inició 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 múltiples 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 optó 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 asignadas dinámicamente con el uso de AngularJs.

Se realizó una segunda entrevista para aclarar cuestiones, analizar las historias de usuario añadir criterios de aceptación, crear nuevas historias y luego establecer prioridad entre las historias.

2.2.1 Primera iteración:



Fig. 6: Historias de Usuario de la primer Iteración

Vale aclarar que en la figura anterior solo están representadas las historias a ser incluidas en el primer Sprint, las demás se anotaron en forma de borrador y ayuda de memoria, debido a que los clientes son libres de modificarlas hasta el inicio del siguiente Sprint, donde se reevalúa la importancias de las historias y se las formaliza como documento que cumple la función de un objetivo a cumplir en el Sprint.

Al ser XP incremental, en cada iteración se agregan los elementos necesarios para completar la misma y se controlan que los elementos anteriores sigan siendo válidos. Por lo que al iniciar la primera iteración se creó el primer esquema de base de datos conteniendo solo los datos necesarios para cumplir con las historias de usuario designadas a este Sprint.

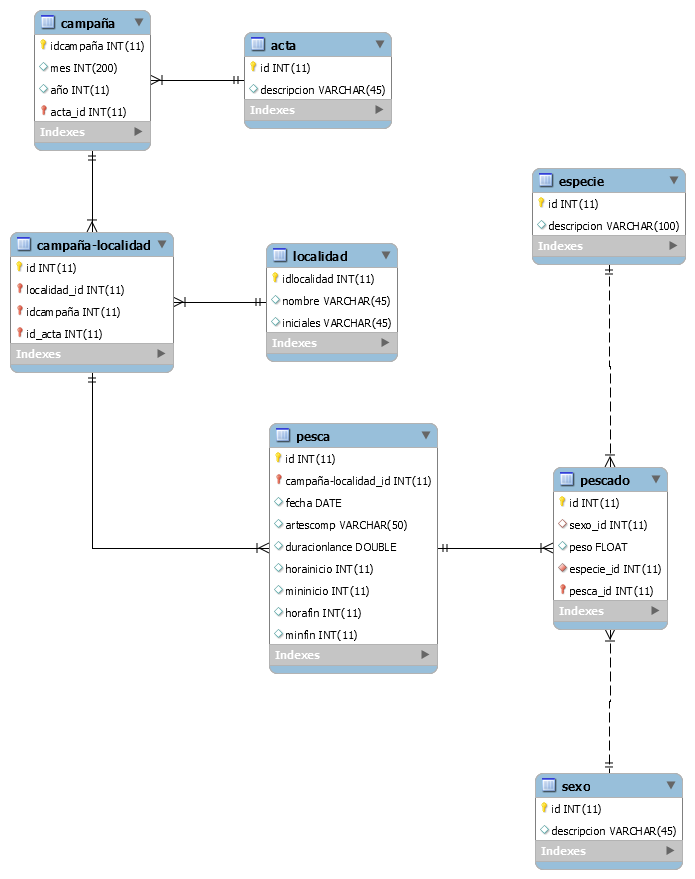


Fig. 7: Primer esquema de base de datos

Debido al volumen y la necesidad de disponer de datos de prueba se creó un proceso ETL con la herramienta Pentaho la cual permitió una extracción y volcado rápido de datos, manipulación y limpieza de datos y creación de llaves primarias y foráneas.

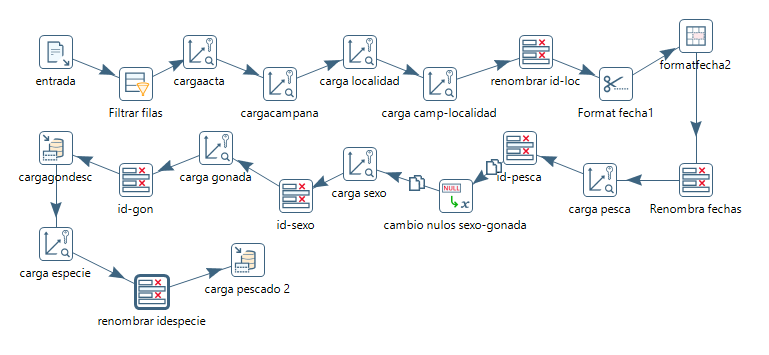


Fig. 8: Primer proceso ETL

Una vez cargados los datos se inició una etapa de análisis más profundo, enfocado a seguridad, basado en OWASP [19]. Se realizó el modelado de amenazas, iniciando por una descripción de la aplicación, como se puede ver en la Tabla 1.

Tabla 1: Análisis de amenazas

|  |  |
| --- | --- |
| Análisis de amenazas | |
| Versión de aplicación | 1.0 |
| Descripción | Sitio web para investigadores de INICNE.  Los usuarios de la aplicación serán:   1. Usuario 2. Usuario autenticado   Un usuario registrado podrá acceder a la información disponible en el sitio. |
| Participantes | Silva Pablo |
| Revisado por | Bosco Francisco |

Seguidos por un análisis de niveles de confianza, los cuales representan distintos perfiles de usuario o agentes externos a la aplicación con sus respectivos nivel de acceso, que luego serán cruzados con los puntos de entrada y los activos que posee la misma.

Tabla 2 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 | Administrador de base de datos | Tiene permisos de lectura y escritura en la base de datos que utiliza la aplicación |
| 3 | Administrador del sitio web | Puede cambiar la configuración |
| 4 | 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 |
| 5 | 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 3 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 |

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 4 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 |
| 1.1 | Filtros de búsqueda | Acepta las entradas otorgadas por los usuarios y las busca en la base de datos | 1. Usuarios |

Se definen como activos a las áreas o ítems de la aplicación que podría interesar a un atacante y son la razón de una amenaza. Se utiliza una clasificación de activos en físicos y abstractos, la información de los usuarios, sus contraseñas y los demás datos propios de la aplicación se los clasifican como activos físicos, y la disponibilidad de la aplicación, su tiempo de respuesta serian ejemplos de activos es abstractos.

Tabla 5 Activos

|  |  |  |  |
| --- | --- | --- | --- |
| Activos | | | |
| Id | Nombre | Descripción | Niveles de confianza |
| 1 | Sistema | Activos relacionados al funcionamiento del sistema | (3) Administrador del sitio web |
| 1.1 | Disponibilidad de la aplicación | La aplicación debería estar disponible todo el tiempo y por todos los usuarios autorizados | (2) Administrador de base de datos  (3) Administrador del sitio web |
| 1.2 | 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 | (3) Administrador del sitio web  (4) Usuario del servicio de servidor web |
| 1.3 | 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 | (2) Administrador de base de datos  (5) Usuario de base de datos |
| 2 | Aplicación web | Activos relacionados a la aplicación |  |
| 2.1 | 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 | (2) Administrador de base de datos |
| 2.2 | Habilidad de añadir datos | Permite a un administrador agregar datos de campañas | (2) Administrador de base de datos  (3) Administrador del sitio web |

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 relacionadas 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.

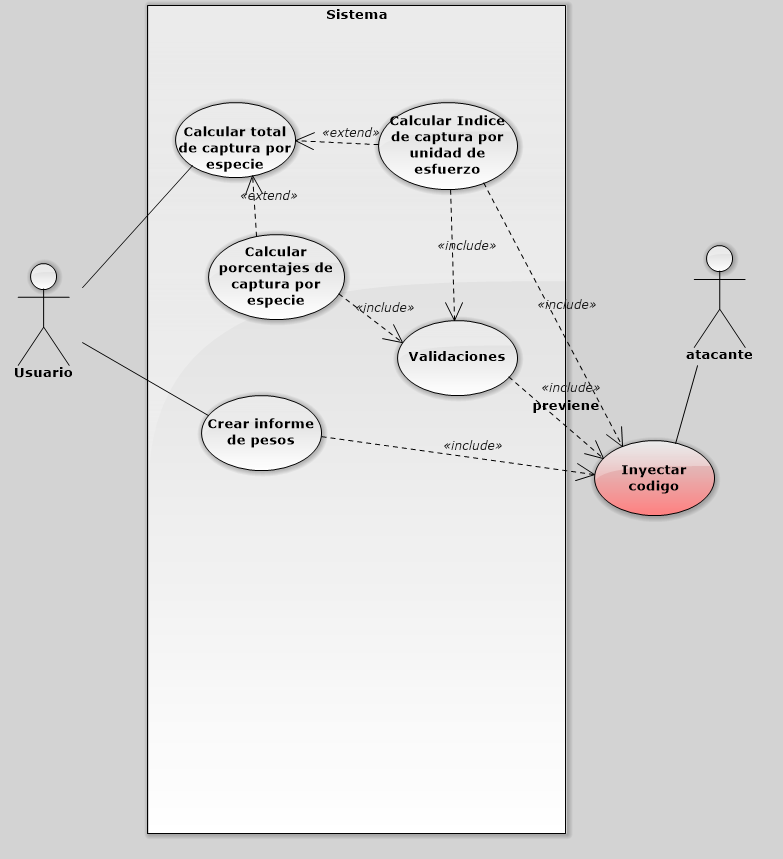


Fig. 9: Casos de abuso

Con esto se tiene una idea de las amenazas a la aplicación, pero es muy probable que no sean todas, por lo que se realizó un brainstorming de donde se extrajeron algunas otras, para anotarlas fue necesario categorizarlas, por lo que se utilizó el método STRIDE

Tabla 6: Clasificación de amenazas con STRIDE

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

De las amenazas anteriores 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.

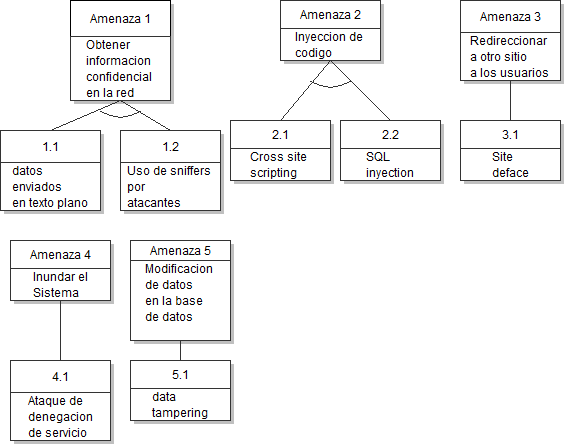


Fig. 10: Árbol de amenazas

Si bien STRIDE puede ser usado para categorizar amenazas, solo lo hace en base al impacto que se cree que va a causar una explotación de la misma, por lo que se inclinó por utilizar DREAD para llevar a cabo dicha tarea. Para determinar el factor técnico y el impacto. Se hacen 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.

Tabla 7: Calificación de amenazas con DREAD

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| id | Amenaza | D | R | E | A | D | Total | Promedio |
| 1.1 | Hombre en el medio | 2 | 2 | 3 | 4 | 5 | 16 | 3.2 |
| 2.1 | Cross site scripting | 3 | 3 | 6 | 8 | 5 | 25 | 5 |
| 2.2 | SQL inyección | 6 | 3 | 3 | 9 | 6 | 27 | 5.4 |
| 3.1 | Site deface | 5 | 2 | 4 | 7 | 5 | 23 | 4.6 |
| 4.1 | Ataque DOS | 3 | 3 | 9 | 9 | 3 | 27 | 5.4 |
| 5.1 | Data tampering | 1 | 7 | 3 | 3 | 5 | 19 | 3.8 |

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 8 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

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

Tabla 9: Estrategia de mitigación adoptada

|  |  |  |
| --- | --- | --- |
| id | Amenaza | Estrategia |
| 1.1 | Hombre en el medio | Aceptar el riesgo |
| 2.1 | Cross site scripting | Mitigar |
| 2.2 | SQL inyección | Mitigar |
| 3.1 | Site deface | Aceptar riesgo |
| 4.1 | Ataque DOS | No hacer nada |
| 5.1 | Data tampering | Aceptar el riesgo |

En base a los resultados expuestos en la Tabla 9 se tomaron las siguientes medidas: tanto para el caso de inyecciones SQL como Cross site scripting el único punto de entrada a esta versión de la aplicación es el campo donde se ingresaran los parámetros para los reportes por lo que se limitara a hacer una validación de entradas, filtrando los caracteres que podrían resultar peligrosos. Al ser una versión reducida y no ser lanzada como entrega, la implementación fue básica apuntando a mitigar los XSS reflejados, inyecciones SQL, en las siguientes iteraciones se volverá a realizar un análisis y en caso de considerar necesario se realizar una refactorización.

Finalizada las etapas de análisis y diseño se procedió a crear las pruebas que el sistema debería pasar, para asegurarse de que sea resistente a inyecciones (tanto SQL como XSS). Posteriormente se codifico siguiendo un patrón MVC a dos niveles, por un lado se crearon modelos de datos en PHP encargados de realizar las consultas a la base de datos utilizando la clase Query builder provista por CodeIgniter, la cual provee una capa de seguridad contra inyecciones SQL. Los controladores tienen dos funciones, por una lado utilizar los modelos para crear API REST que provean datos, y por otro brindar una vista con un template estático, en esta entra en juego el segundo nivel, dentro de esta vista se incluirá la una aplicación AngularJS, la cual se encargara de cargar una parte del template contenido y actualizado de forma dinámica (conocido como vistas parciales) y asociar un controlador a cada una de esas vistas, el cual se encargara de obtener los datos de forma asíncrona y de dar funcionamiento a la vista parcial.

2.2.2 Segunda iteración

Para el segundo Sprint se agregó un reporte, por lo que se amplió la base de datos y se modificó el proceso ETL para poblarlo, además se incluyó la carga de datos por medio de la aplicación, lo que introdujo nuevos puntos de acceso, en donde se debe confiar que la información ingresada sea segura y correcta. CodeIgniter sugiere realizar un control únicamente para las salidas de la base de datos, y no un control de entradas ya que su Query Builder es resistente a inyecciones SQL. Se reiteró el proceso de análisis y diseño incluyendo una modificación de la base de datos para dar soporte a la nueva información, y con una nueva fase de exploración y evaluación de amenazas, se diseñaron las pruebas y para la codificación se optó por el uso de una librería de CodeIgniter llamada Grocery Crud, la cual facilito el proceso de ABM y sobre la cual se agregaron filtros de control de entradas para detectar tipos de datos incorrectos y código malicioso. Se debió realizar modificaciones sobre la misma a causa de que los temas por defecto no superaron los test de inyección XSS, también se reemplazaron las entradas de datos para los reportes por combo box disminuyendo las posibilidades de que un usuario ingrese código malicioso por ese medio y se realizó una refactorización de la función que controlaba las entradas por una más eficiente y segura.

Una modificación importante realizada en esta iteración fue la creación de una tabla que se utiliza para probar inyecciones de código, la misma se describe a continuación:

Tabla 10: Test de inyecciones

|  |  |
| --- | --- |
| Nombre | Script |
| Inyección SQL con comillas simples | 1’; DROP TABLE usuarios;# |
| Inyección SQL con comillas dobles | 1”; DROP TABLE usuarios;# |
| XSS simple | <script>alert("xss!")</script> |
| XSS incluido en una acción | <b onmouseover=alert('XSS!')>click!</b> |
| XSS en error | <IMG SRC="" onerror="alert('XSS')"> |
| XSS como entrada | <INPUT type= "text" value = ''><SCRIPT> alert ( "XSS" ) </SCRIPT> <xss a=''> |
| XSS codificado en ASCII | <IMG SRC= &#106; &#97; &#118;&#97; &#115;&#99; &#114; &#105;&#112; &#116;&#58;&#97; &#108; &#101;  &#114; &#116; &#40; &#39; &#88; &#83; &#83; &#39;&#41;> |
| XSS codificado en Hexadecimal | <IMG SRC=&#x6A; &#x61; &#x76; &#x61; &#x73; &#x63; &#x72; &#x69; &#x70; &#x74; &#x3A; &#x61; &#x6C; &#x65; &#x72; &#x74; &#x28; &#x27; &#x58; &#x53; &#x53; &#x27 ;&#x29;> |
| XSS sin comillas | <script>alert(String.fromCharCode(88,83,83));</script> |
| XSS en iframe | <IFRAME SRC="javascript:alert('XSS');" style='display:none' ></IFRAME> |

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.

3.1 Lenguajes de programación

PHP [22] 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 [23], para dar instrucciones al servidor y comunicarse con el motor de base de datos, es la base de la aplicación que provee autenticación, niveles de acceso a usuarios, plantillas donde se mostrara información básica y estática y dará lugar a la ejecución del código JavaScript, también proveerá API REST [24] que alimentan a la aplicación.

JavaScript [25] 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 [26] como framework y se añadieron funcionalidades Jquery, ambas 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.

3.2 Gestión de datos

MySQL [27] 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 [28] 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 almacenados y triggers, y realizar los cambios por medio de una interfaz gráfica, ambos utilizan licencias GPL.

Pentaho [29] Data Integración, también llamada kettle, es un componente de la suite 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.

3.3 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 [30], con licencia GPL versión 2.

Npm [31] 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.

3.4 Otras herramientas

XAMPP [32]: 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 [33], la cual posee una estructura base sobre la cual se escriben las historias, y luego son exportadas a una planilla Excel.

Al ser una aplicación web necesita de un cliente HTTP o navegador, para probar la aplicación se utilizó Firefox [34] mayoritariamente, junto con su versión de desarrollo Firefox Quantum.

Una de las principales razones de la elección de ese navegador fue la facilidad y variedad de extensiones en forma de plugins que posee que ayudan en la etapa de diseño de interfaces y en la etapa de testing. FireBug [35] es una de ellas, que consiste en una consola orientada al desarrollo, funciona de forma inteligente y es especialmente útil al trabajar con APIs y JSON.

TamperData [36] es otra extensión de Firefox, esta fue utilizada únicamente en la etapa de testing, su funcionamiento consiste en detener las peticiones http y da la opción de modificar las mismas, se la utilizo para comprobar el comportamiento de la aplicación cuando un usuario modifica la información en un paso intermedio entre la interfaz de la misma y el backend, como también para comprobar la seguridad de las sesiones.

Google Chrome [37] también fue utilizado para realizar las pruebas de seguridad, debido a la necesidad de tener dos sesiones abiertas.

Suraksha [37] es una herramienta que nació con el propósito de ayudar en la etapa del diseño de software seguro, con la finalidad de facilitar esta etapa. Se la utilizo únicamente para la elaboración de los árboles de amenaza.

Capítulo 4. Resultados

4.1. Documentación

Una vez finalizados todos los Sprints el resultado es un conjunto de documentos que contiene la versión final de cada uno de los documentos.

Con cada sprint se añadieron las funcionalidades pertenecientes al mismo con todo lo que estas implican como ser análisis, diseño, refactorización, etc., como así también los cambios en los requerimientos y cambios en las historias de usuario.

Si bien, en XP, 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 dividen las tareas por prioridad, y se crea una estrategia de Sprints y otra de entregas que pueden coincidir o no. Se asigna las tareas de mayor prioridad al próximo Sprint sobre ellas se realiza el análisis, diseño, pruebas y codificación, pudiendo terminar el Sprint sin una entrega, de esta forma se obtiene 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.

Durante la producción de la aplicación se elaboraron las historias de usuario con ayuda de distintos responsable del INICNE y se realizaron múltiples entrevistas para clarificar cuestiones que particulares, a su vez a medida que se terminaba una entrega, se obtenía una retrospectiva de los usuarios, lo que en ocasiones tuvo como resultado tener que agregar historias de usuario o añadir requisitos de aceptación a otras historias, esto implica que historias que habían sido marcadas como terminadas, sean puestas en la cola de historias a hacer (To Do) antes de ser aceptada y por consiguiente una refactorización del código que daba la funcionalidad que satisfacía dicha historia.

Tabla 11: 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.

A pesar de que XP, y el paradigma ágil en general, sugieren no desarrollar ni incluir en el producto funcionalidades que no formen parte del Sprint actual, durante la última iteración se consideró que el modelo de datos no iba a recibir variaciones significativas por lo que se realizó una migración completa del Excel a la base de datos (ver Fig. 11) en el que se incluyeron datos de distintos documentos.

Otra de las razones que justificaron esta acción fue que si bien el desarrollo de la aplicación estaba terminado en términos de que se habían cumplido todas las historias de usuario, es altamente probable que a lo largo del tiempo el software requiera un mantenimiento perfectivo y el hecho de disponer de un modelo de datos completo simplificaría el trabajo de análisis y migración.

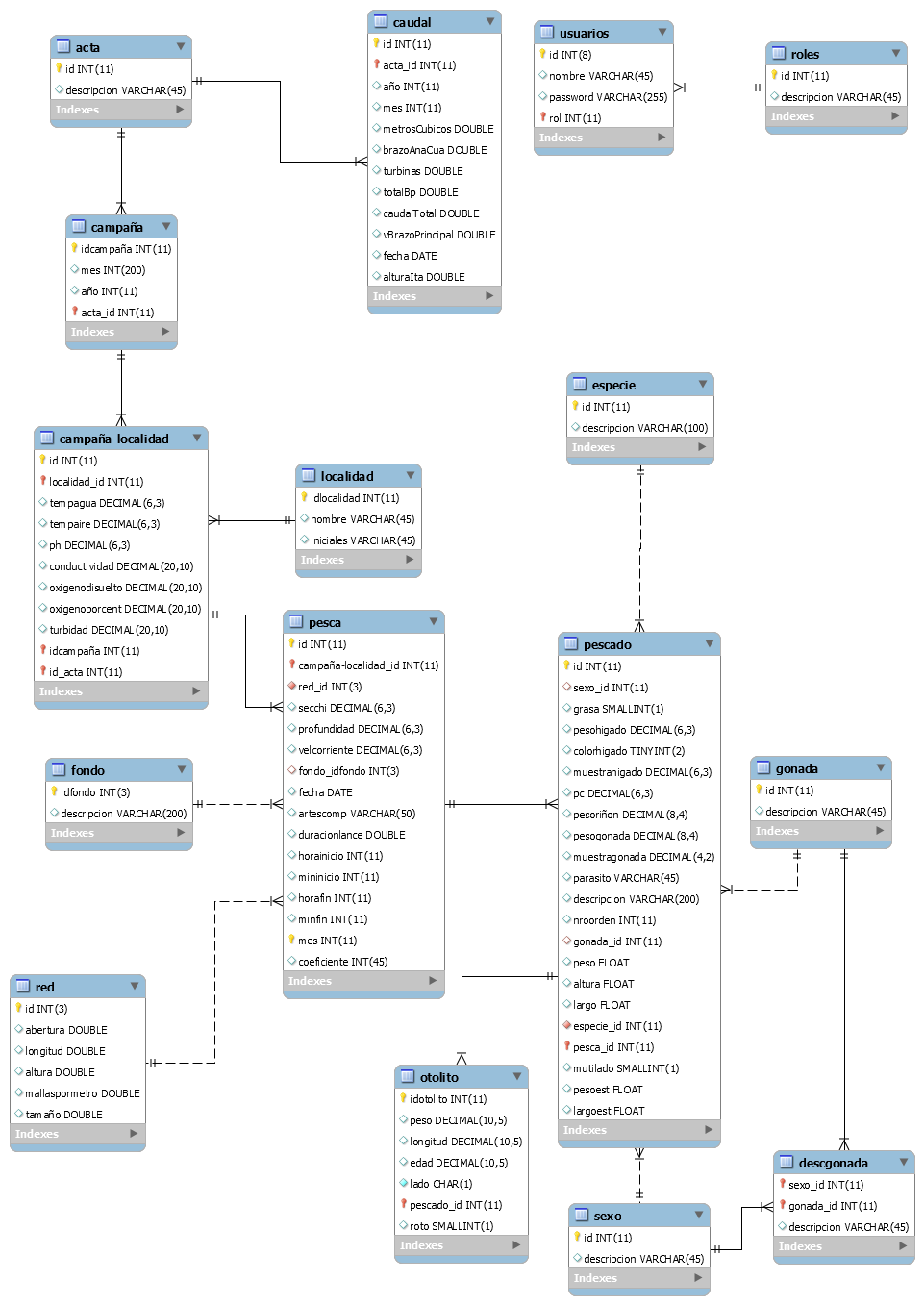


Fig. 11: Modelo de datos final

Finalizado el modelo de datos, se prosiguió a diseñar el proceso ETL final. Este incluyo nuevamente las tablas anteriores debido a la representación en forma de tabla única, lo cual implica volver a crear índices, separar valores, filtrar entradas erróneas, etc. y también debido a la incorporación de nuevos atributos en muchas de las tablas. Se inició por las tablas con menos dependencias, 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 la creación de claves, relacionar tablas con claves foráneas e integrar distintas fuentes.

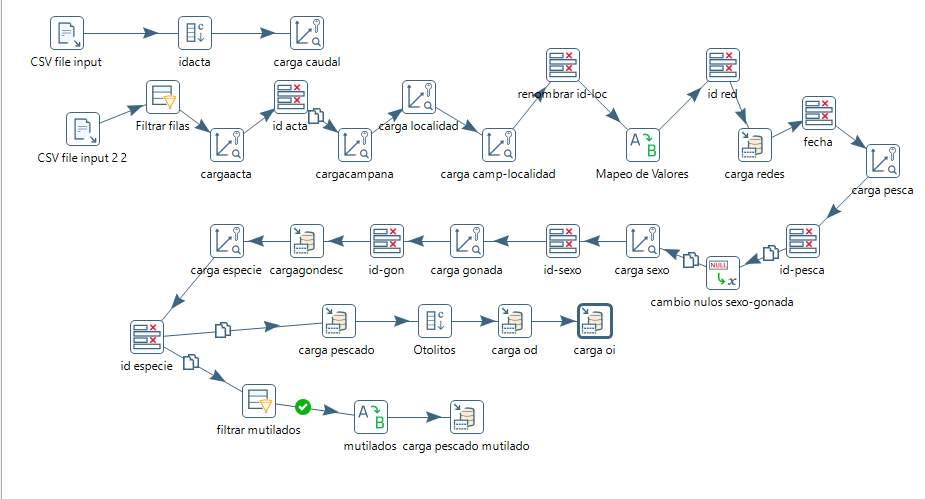


Fig. 12: Proceso ETL final

Durante los tres Sprints se siguió la con la metodología indicada en el capítulo 2, cada uno de los documentos fue reemplazado por una versión incrementada del mismo, en esta última iteración se añadieron distintos perfiles de usuario, como se ve en la Tabla 12, donde también se incluyó una descripción de los permisos que posee cada grupo.

Tabla 12 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 |

El trabajar con perfiles de usuario trae consigo una necesidad de autenticarlos, en especial cuando se incluyen distintos niveles de acceso, por lo que la tabla de niveles de confianza debió ser rediseñada de la siguiente forma:

Tabla 13 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 se mantuvieron igual a lo largo del proyecto, si bien pudieron haberse instalado actualizaciones en cualquiera de los elementos, los mismos quedan fuera del alcance de este trabajo y no se considera que representen un mayor riesgo que sus versiones anteriores.

Tabla 14 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 |

Las entradas al sistema mencionadas en el primer Sprint se ven ahora acompañadas de diversos puntos de acceso nuevos, en especial los formularios de entrada, que son un acceso fácil a la base de datos permitiendo ataques del tipo XSS almacenados, y la página de login, que es en la mayoría de los casos el punto a donde se dirigen la mayoría de los ataques de fuerza bruta e inyecciones SQL

Tabla 15 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 |

Uno de los principales objetivos de un ataque es la elevación de privilegios, poder ejecutar todas las acciones en la aplicación como si fuese un administrador para facilitar el acceso a la base de datos, por esa razón uno de los activos más importantes pasa a ser la sesión de usuario. El hecho de que un atacante pueda robar una sesión de administrador implica que potencialmente puede inyectar código XSS en la base de datos, eliminar información, agregar información errónea, y en algunas ocasiones puede escalar a robar la base de datos completa.

Tabla 16 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)Asistente (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  (5) Asistente  (7) Administrador del sitio web  (5) Asistente |

En los casos de abuso podemos ver como se expandió la aplicación a lo largo de los Sprints como también ver de una manera gráfica los accesos que tiene cada nivel de usuario, como también permite hacer un análisis rápido de posibles soluciones a los ataques que pueda llegar a recibir la aplicación. Como se puede observar la mayoría de las funciones se pueden proteger con validaciones para evitar XSS reflejados, más adelante se detallara las medidas que se tomaron para proteger contra código ya almacenado.

También se vuelve a destacar la función de login por la variedad de ataques que derivan de ella y el número de elementos a tener en cuenta para hacer la aplicación lo más segura posible.

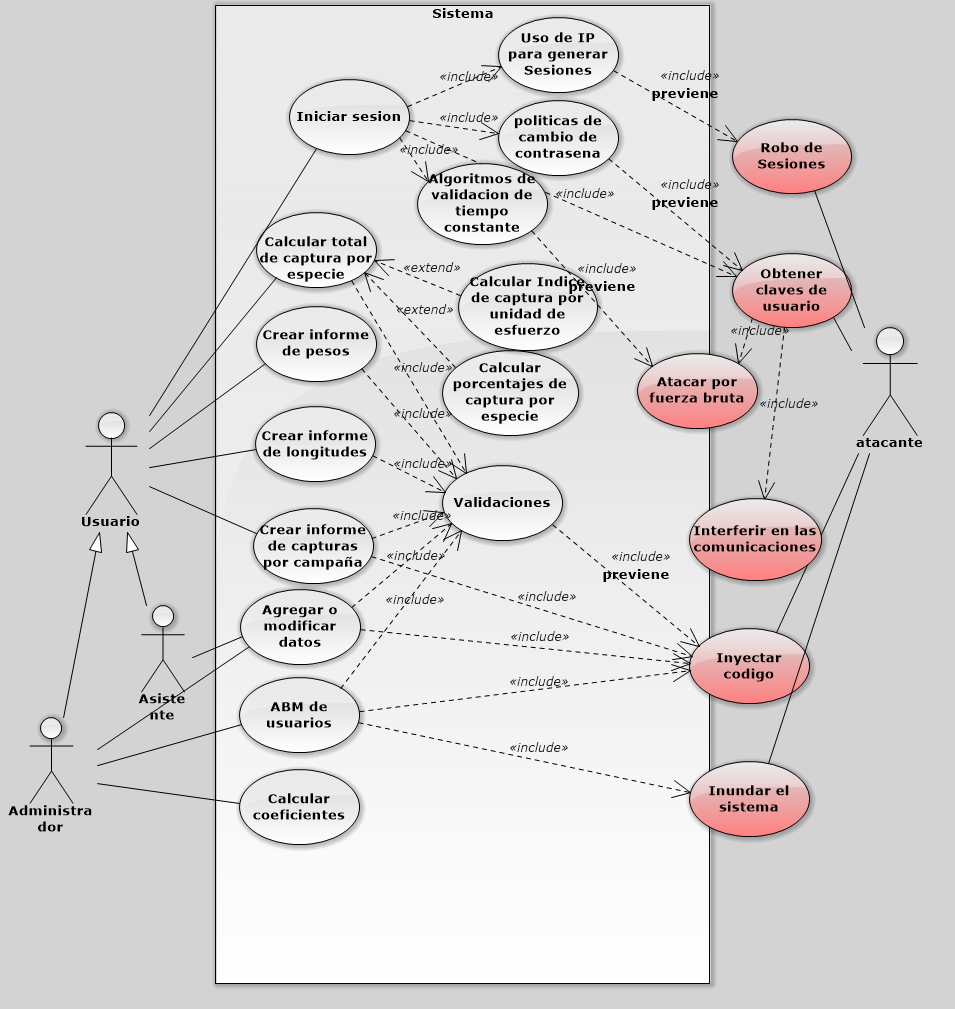


Fig. 13: Casos de abuso

Se volvieron a clasificar las amenazas utilizando STRIDE

Tabla 17: 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 |

Y como del análisis anterior se diseñó un nuevo árbol de amenazas teniendo en cuenta cuáles de sus variantes podrían afectar a la aplicación

.

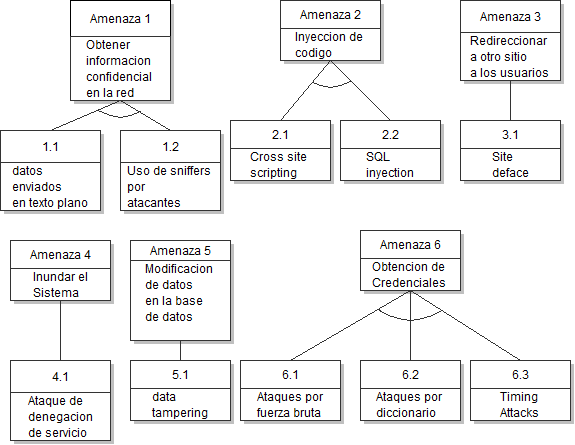


Fig. 14: Árbol de amenazas

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. 15 Calificación de amenazas con DREAD

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| id | Amenaza | D | R | E | A | D | Total | Promedio |
| 1.1 | Robo de sesiones | 7 | 5 | 5 | 7 | 5 | 29 | 5.8 |
| 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 |

Terminadas las etapas anteriores 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:

Fig. 16 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 |

Previo a la codificación de la última iteración se volvieron diseñar las pruebas a realizar. Por un lado se utilizaron las mismas inyecciones que en la segunda iteración

Tabla 18: inyecciones

|  |  |
| --- | --- |
| Nombre | Script |
| Inyección SQL con comillas simples | 1’; DROP TABLE usuarios;# |
| Inyección SQL con comillas dobles | 1”; DROP TABLE usuarios;# |
| XSS simple | <script>alert("xss!")</script> |
| XSS incluido en una acción | <b onmouseover=alert('XSS!')>click!</b> |
| XSS en error | <IMG SRC="" onerror="alert('XSS')"> |
| XSS como entrada | <INPUT type= "text" value = ''><SCRIPT> alert ( "XSS" ) </SCRIPT> <xss a=''> |
| XSS codificado en ASCII | <IMG SRC= &#106; &#97; &#118;&#97; &#115;&#99; &#114; &#105;&#112; &#116;&#58;&#97; &#108; &#101;  &#114; &#116; &#40; &#39; &#88; &#83; &#83; &#39;&#41;> |
| XSS codificado en Hexadecimal | <IMG SRC=&#x6A; &#x61; &#x76; &#x61; &#x73; &#x63; &#x72; &#x69; &#x70; &#x74; &#x3A; &#x61; &#x6C; &#x65; &#x72; &#x74; &#x28; &#x27; &#x58; &#x53; &#x53; &#x27 ;&#x29;> |
| XSS sin comillas | <script>alert(String.fromCharCode(88,83,83));</script> |
| XSS en iframe | <IFRAME SRC="javascript:alert('XSS');" style='display:none' ></IFRAME> |
| XSS deface | <script>window.location.href="https://google.com" </script> |
| Robo de cookie | <script>alert(document.cookie);</script> |

En los casos de prueba se utilizaron Scripts básicos, ya que de no funcionar estos tampoco lo iban a hacer otros mas elaborados, y a fines de testing, emitir un alerta y capturar un valor utilizando otro sitio web representan la misma amenaza.

Se añadieron también pruebas de robo de sesiones y de manipulación de información, las cuales van a ser descriptas en detalle más adelante en este capítulo.

4.2 Funcionamiento del Software

Al acceder al sitio con lo primero que nos topamos es la ventana de inicio, solicitando credenciales de usuario de la aplicación.

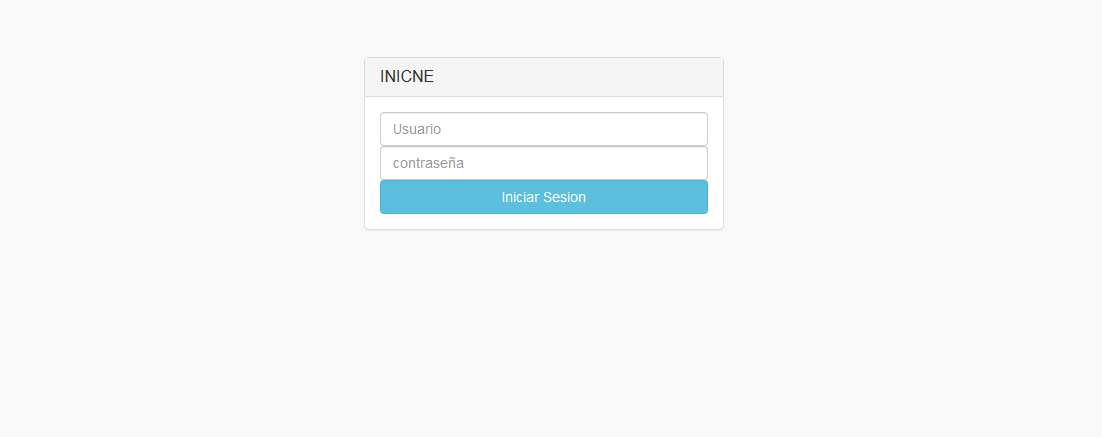


Fig. 17: ventana de inicio

Dependiendo del rol asignado al usuario se van a tener más o menos opciones en el menú lateral, los usuarios sin permisos especiales solo pueden generar informes. En la pantalla principal una vez dentro de la aplicación se puede ver un menú lateral con los informes que puede generar, y un botón en la parte superior derecha para cerrar la sesión.

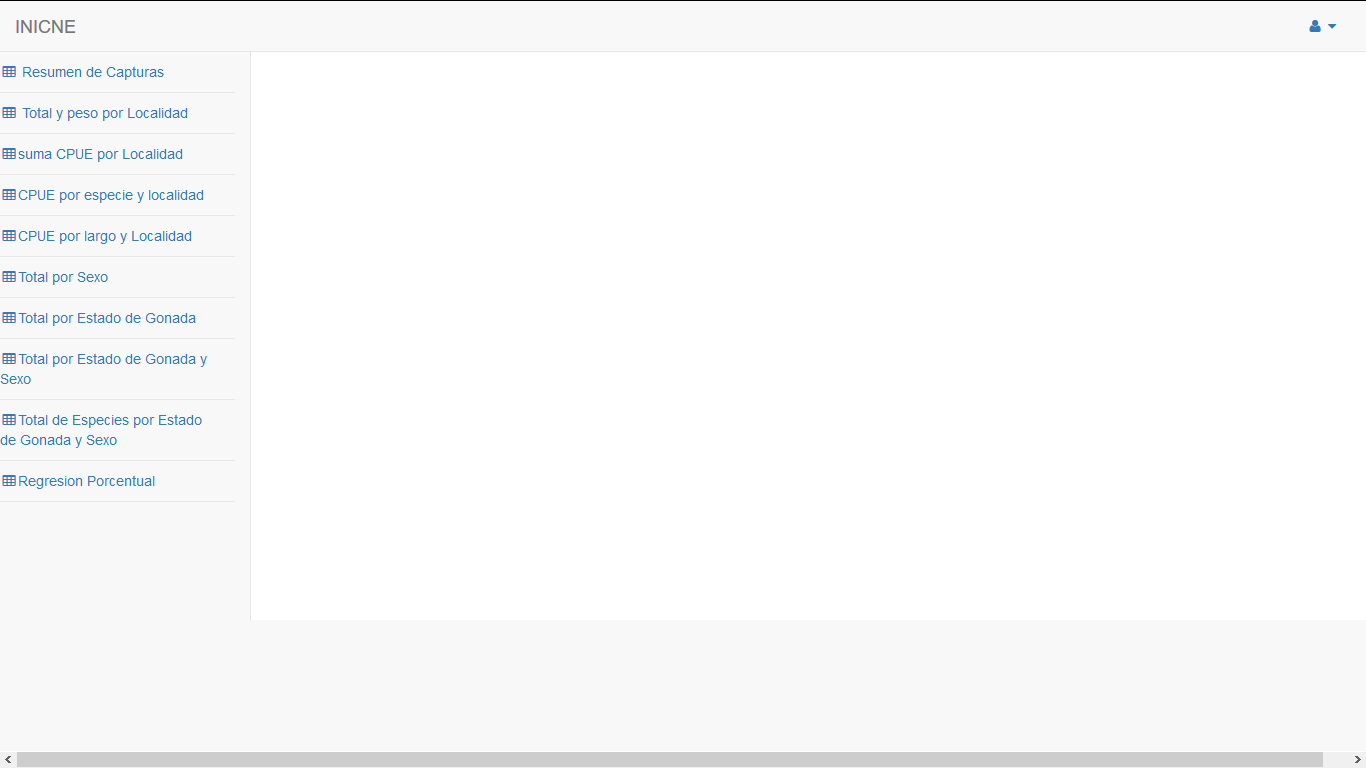


Fig. 18: Menú de Usuario

Cada elementos del menú genera un informe distinto, todos los informes están parametrizados por al menos un aspecto, y representan la información en forma de tabla y algunos de forma gráfica con la posibilidad de ser exportados.

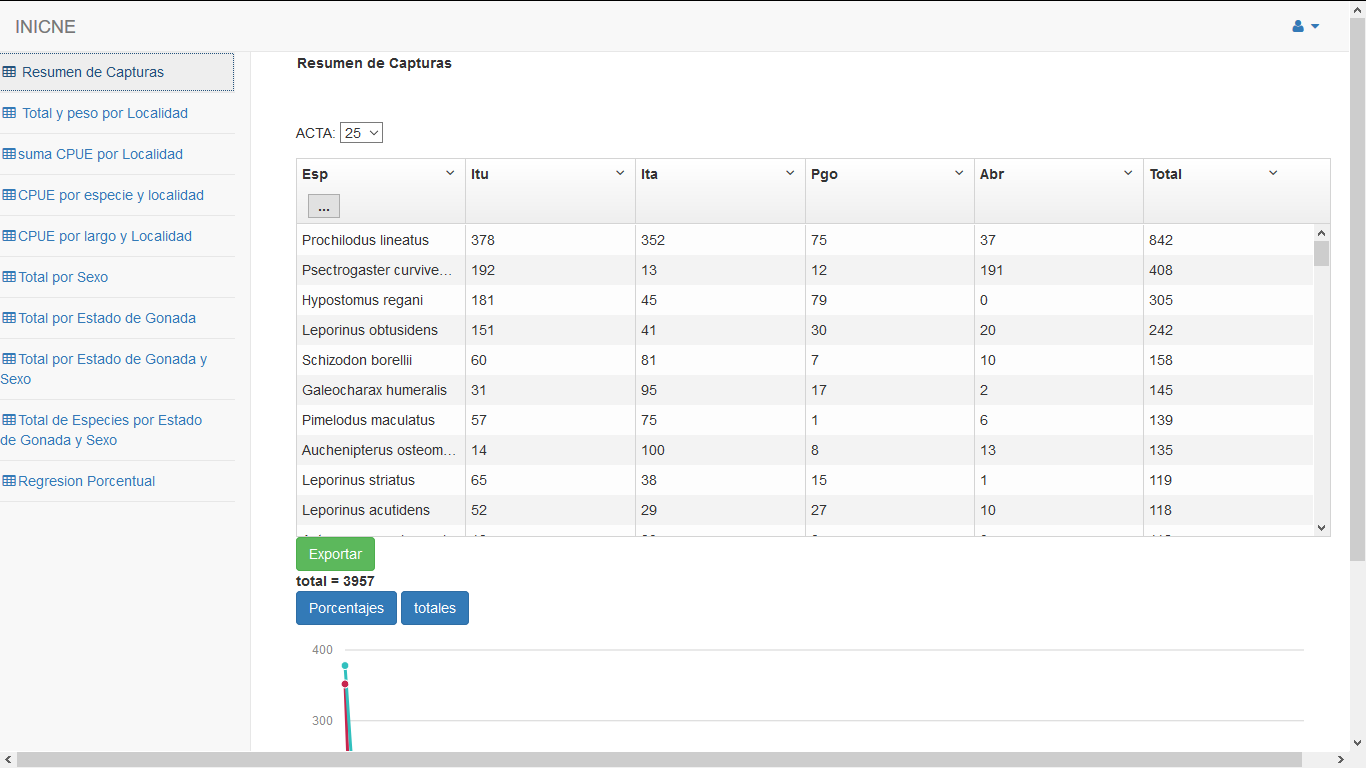


Fig. 19: Tabla resumen de capturas

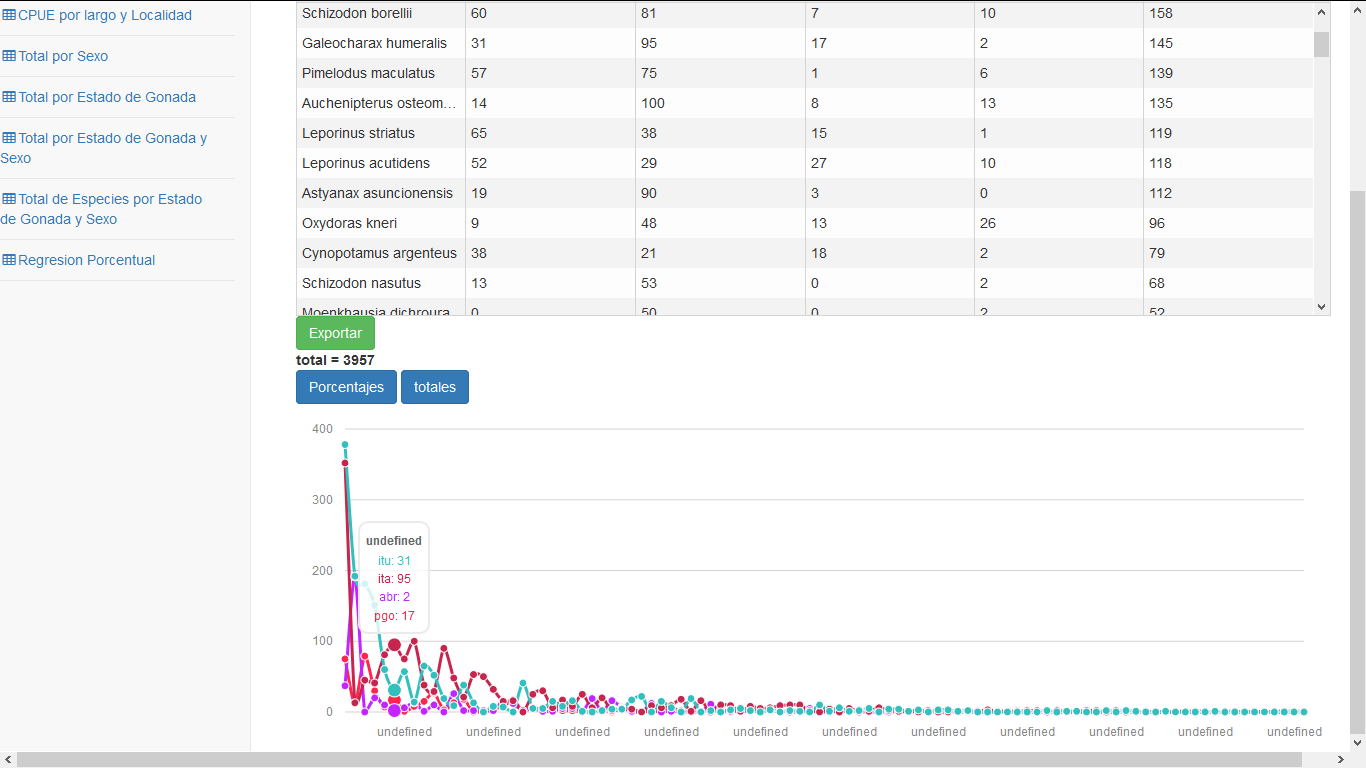


Fig. 20: Grafico de resumen de capturas

Las dos anteriores figuras pertenecen al primer informe del menú, perteneciente al primer Sprint. Al momento de desarrollar este módulo la metodología de trabajo era distinta se basaba en dejar toda la lógica de negocio en PHP, mientras que la base de datos solo almacenaba los resultado, si bien XP sugiere realizar refactorizaciones de código siempre que sea posible, para poder comparar con los resultados obtenidos delegando la lógica a la base de datos se optó por que permanezca así. Para cargar la tabla completa se realiza una transferencia de 7.14KB y tarda en promedio 340 ms. trabajando de forma local.

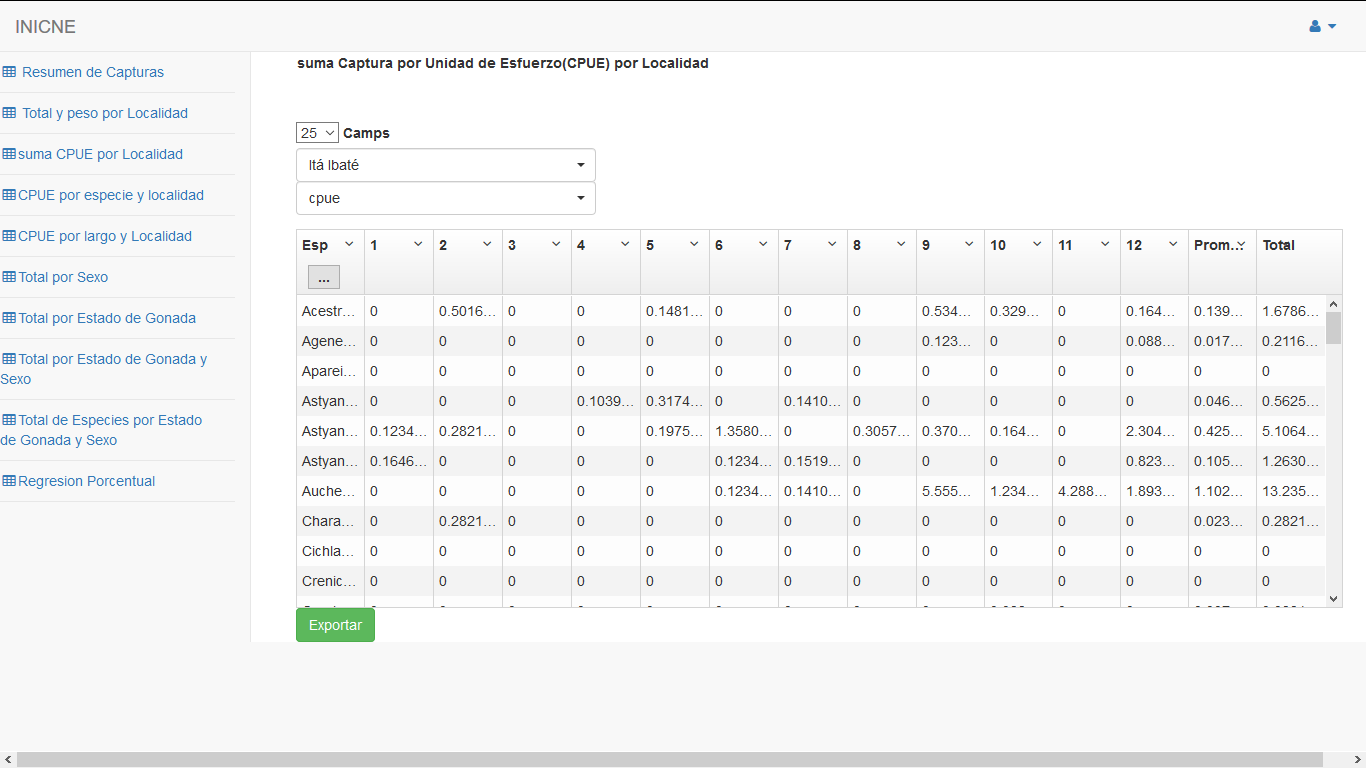


Fig. 21: Tabla de capturas por unidad de esfuerzo

La figura anterior pertenece a otro de los informes, CPUE por especie y localidad, la consulta necesaria para realizar este informe fue bastante compleja, utilizando SQL dinámico y realizando una transformación de filas en columnas para poder obtener la estructura deseada. Si bien no realiza la misma función, en complejidad es muy similar a la anterior, como así también volumen de datos que genera como salida permitiendo realizar una comparación de rendimiento. Esta última para generar transmitir 7.87KB demora en promedio 75ms, si bien la diferencia parece mínima y despreciable ya que ninguno demora más de medio segundo, a la hora de ejecutar la aplicación se puede percibir la diferencia entre ambos.

No solo se percibe la diferencia en cuanto a tiempo de respuesta, sino que, desde el punto de vista del desarrollo, el código es mucho más corto, simple, entendible y reutilizable.

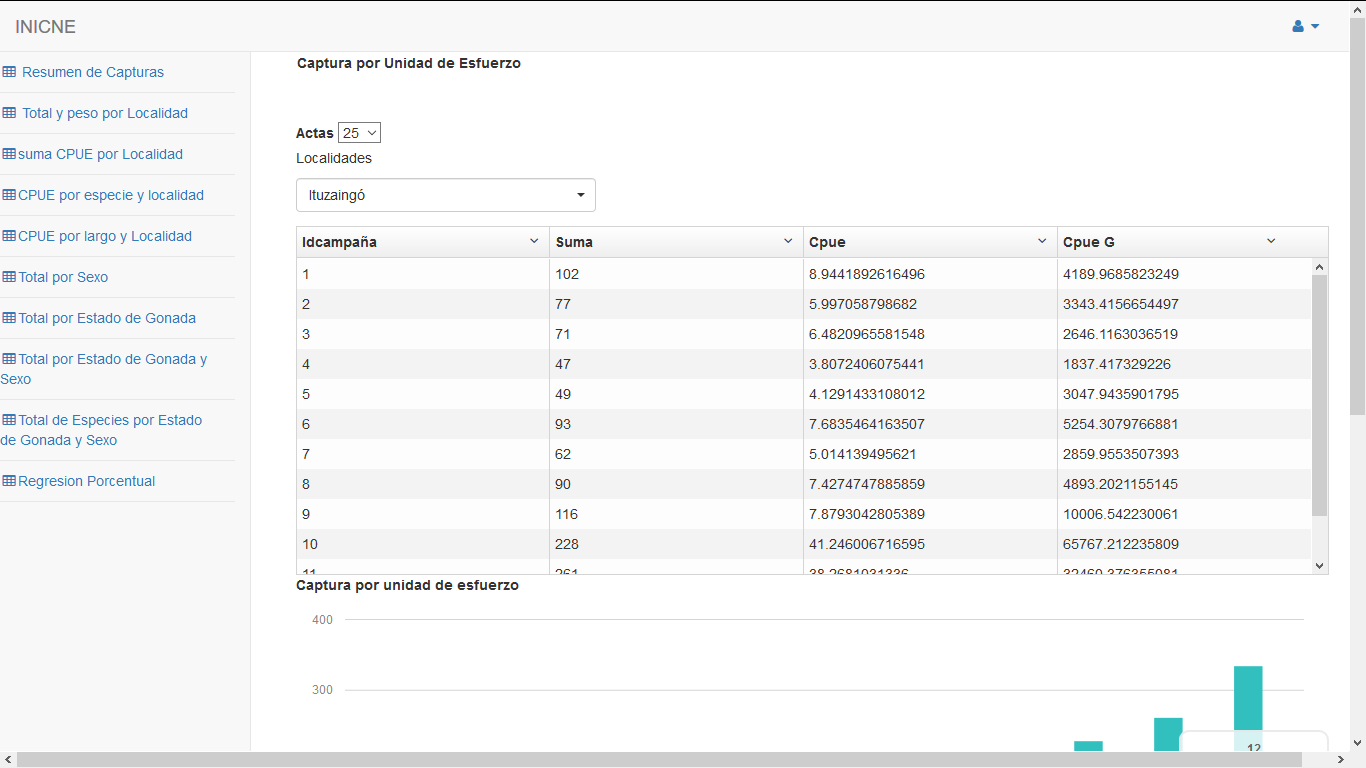


Fig. 22: Tabla de capturas por unidad de esfuerzo



Fig. 23: Grafico de capturas por unidad de esfuerzo

En las Fig. 22 y Fig. 23 se puede observar otro de los informes en el cual se puede comparar de forma gráfica el peso total de las pescas en relación a la cantidad de pescados.

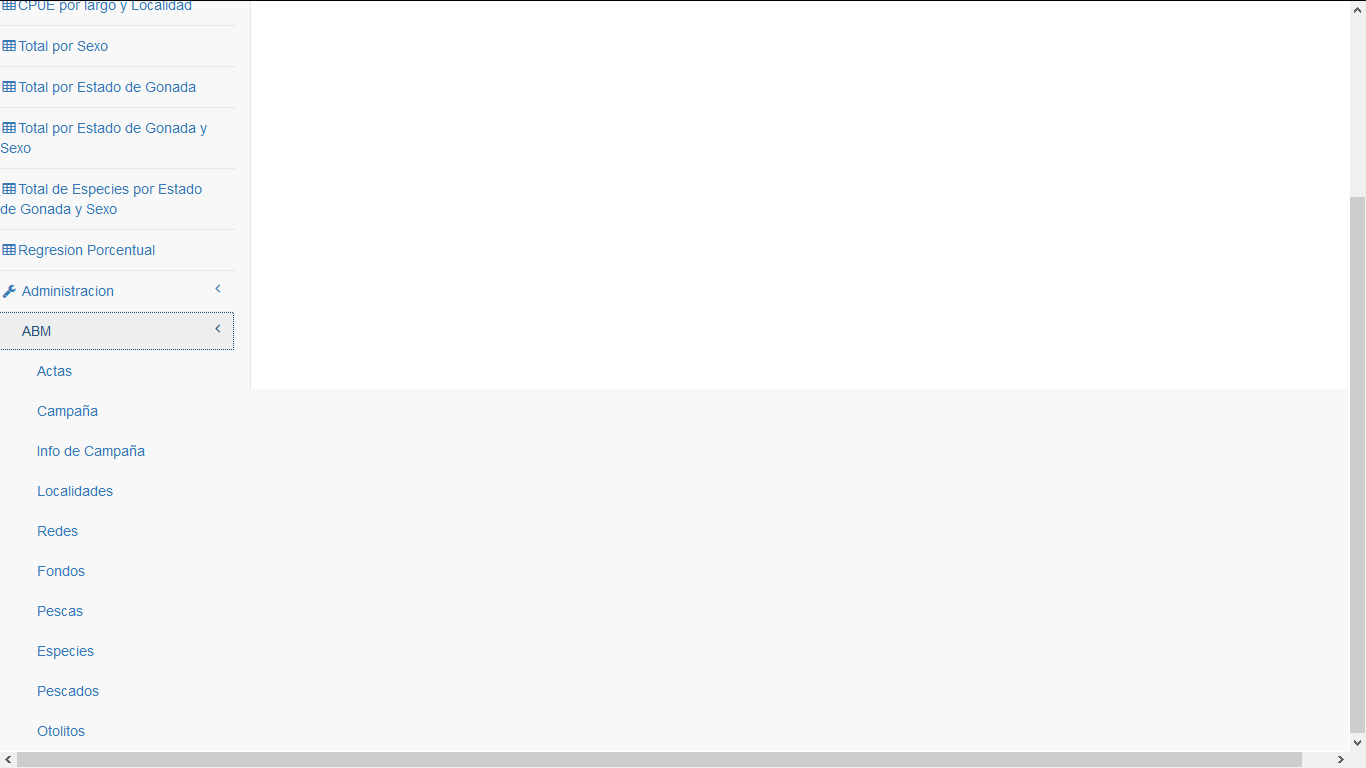


Fig. 24: Menú de asistente

En el menú de asistente se puede observar que se añaden opciones para cargar lo distintos elementos utilizados por el sistema.

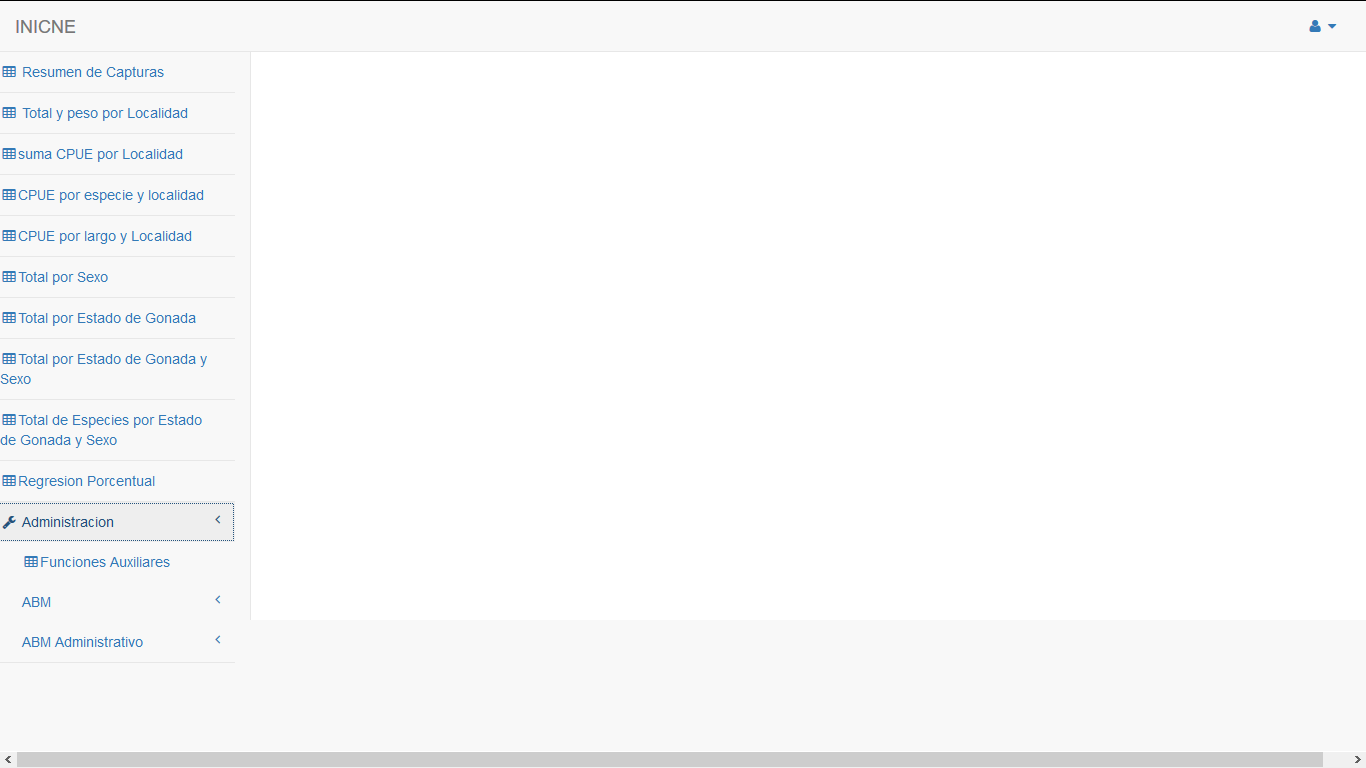


Fig. 25: Menú de administrador

El administrador posee las mismas opciones que los dos roles anteriores, incluidas la posibilidad de agregar usuarios, modificar su contraseña y realizar funciones secundarias.

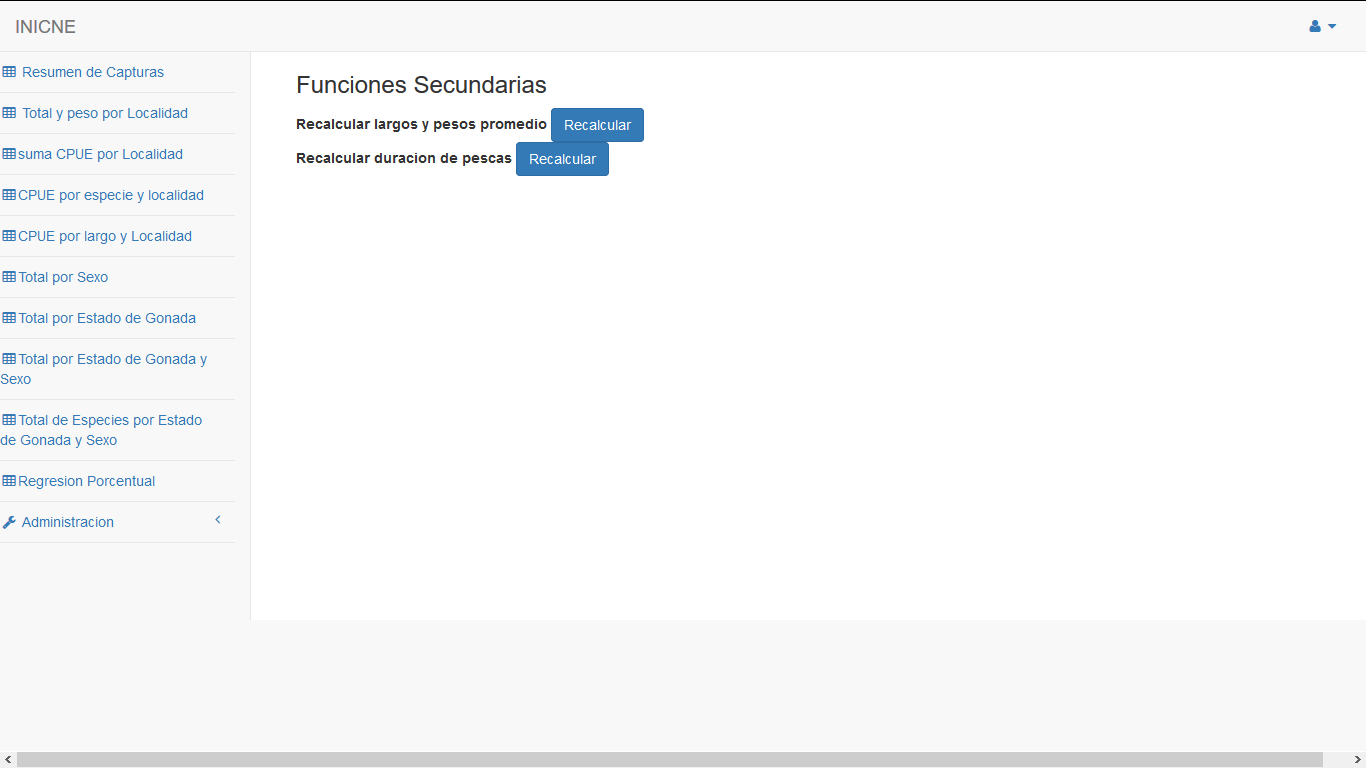


Fig. 26: Funciones secundarias

En la imagen anterior se pueden observar las funciones secundarias, la primera es una función de cálculo muy intensiva, consiste en calcular el peso y el largo promedio por especie y por red para asignar a los peses mutilados, esto sirve para tener una estimación de cuanto podrían medir pesar, y poder utilizarlo como una muestra y no tener que descartarlo a la hora de realizar los informes.

La segunda es útil en el caso de que se haya necesitado modificar la hora de inicio o finalización de alguna o varias pescas, si bien esta función esta implementada en forma de trigger en la base de datos, solo se la añadió en las altas, de esta forma los administradores tienen más control sobre las modificaciones realizadas por los asistentes.

En la Fig. 27 podemos observar uno de los menús de AMB, dado de los semas son similares se lo utilizara para generalizar el funcionamiento de los mismos.

En este menú se puede ver un listado de todos los elementos de esa clase, para mantener la pantalla lo más limpia posible parecer más llamativo se paginaron los elementos, a diez por página, lo cual puede ser modificado utilizando el botón inferior izquierdo. Debajo del título de la tabla se puede acceder al menú de altas opciones para exportar e imprimir la lista.

En la lista de elementos se puede observar las características de cada uno, y a la derecha se encuentran los botones para inspeccionar el registro, editarlo y borrarlo, acomodados en ese orden.

Debido a la cantidad de elementos de algunas clases, y el incremento que podrían llegar a recibir encontrar un elemento entre las paginas se puede volver muy tedioso, por lo que se incluyó una búsqueda situada justo debajo de la tabla, a la cual se le puede especificar porque atributo se está realizando la misma, o que la haga de forma global.

A la derecha también se puede encontrar un botón para quitar los filtros aplicados de una forma rápida.

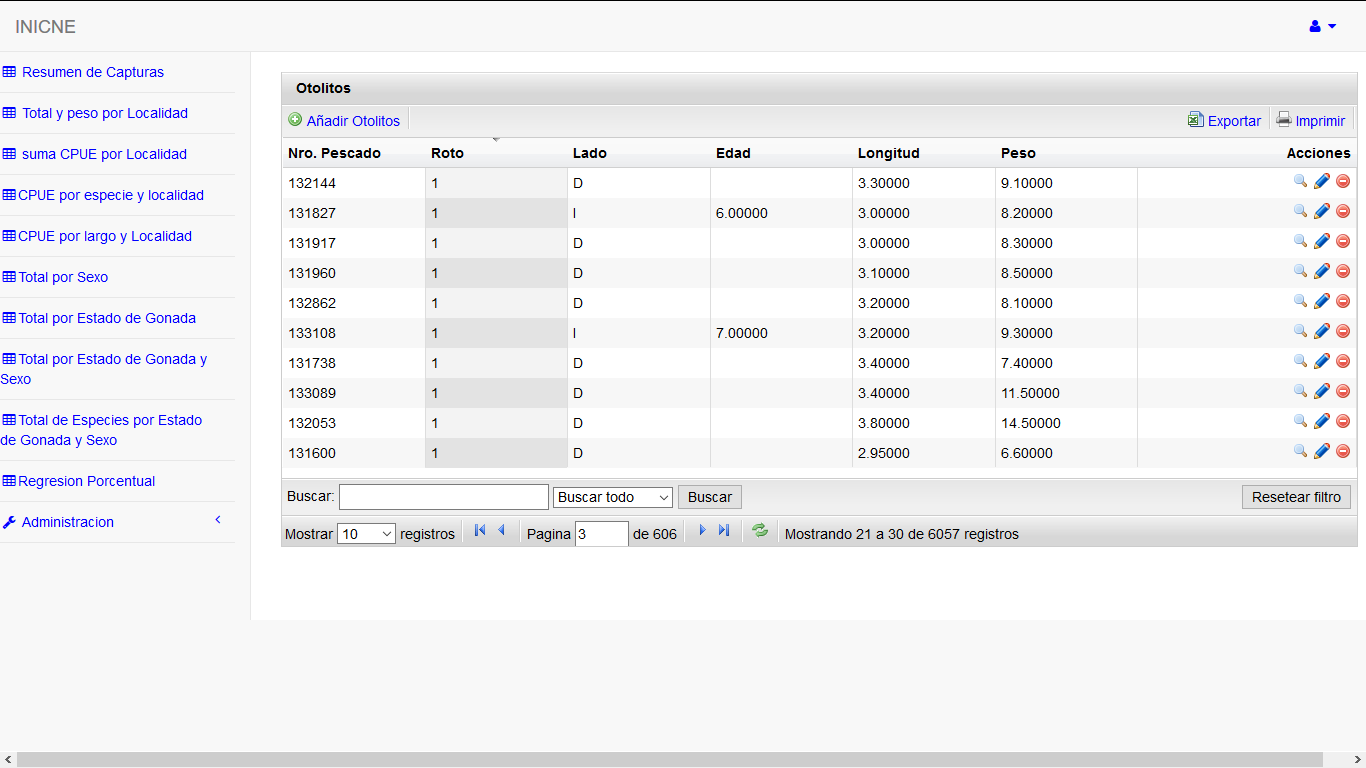


Fig. 27: Menú AMB

En algunos casos no todos los atributos se encuentran en la lista o resulta complicado observar varios al mismo tiempo en tablas grandes como la de pesca o pescado, resultando útil tener este tipo de vista también.

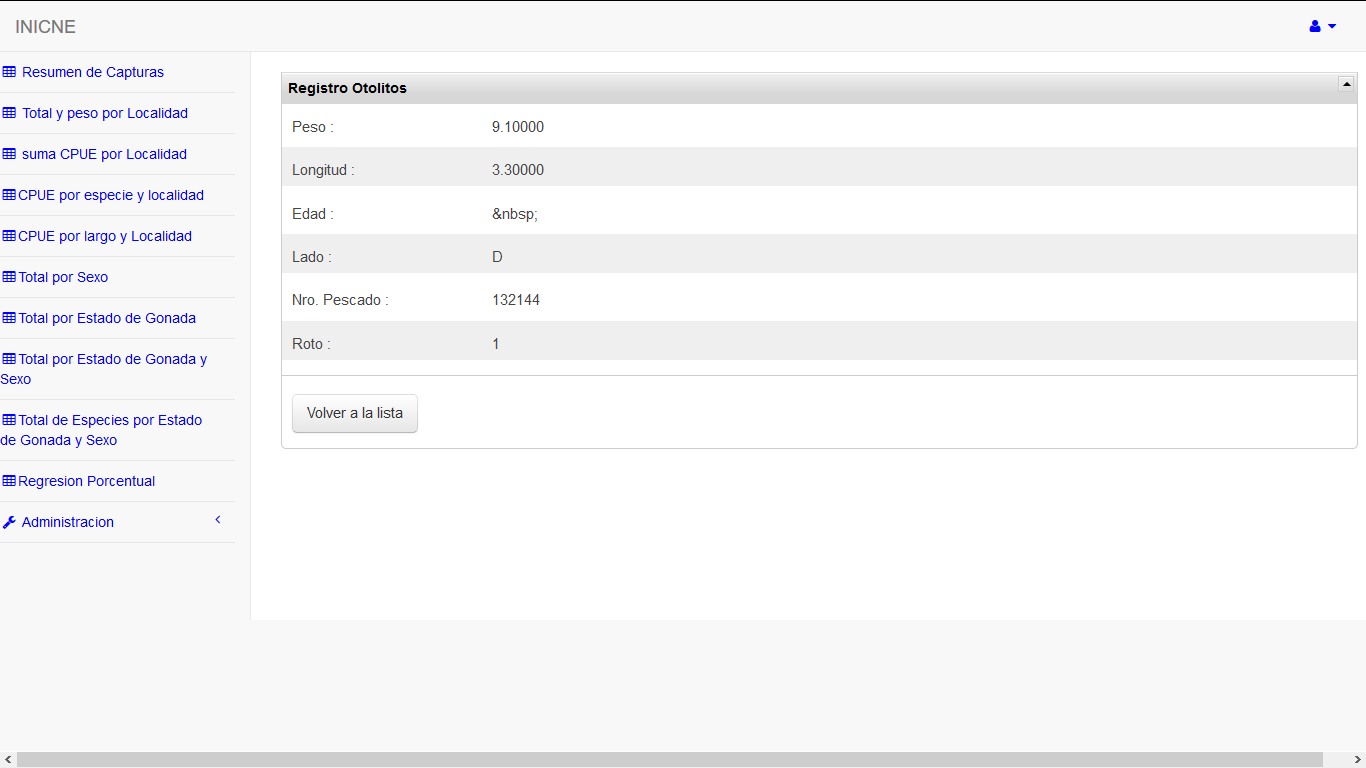


Fig. 28: Menú inspeccionar

Utilizando la opción de edición nos dirigimos una ventana similar a la anterior, con la particularidad de que algunos campos son editables, para asegurarse de que se mantenga la integridad referencial e integridad de identidad se quitaron las llaves primarias de la edición.

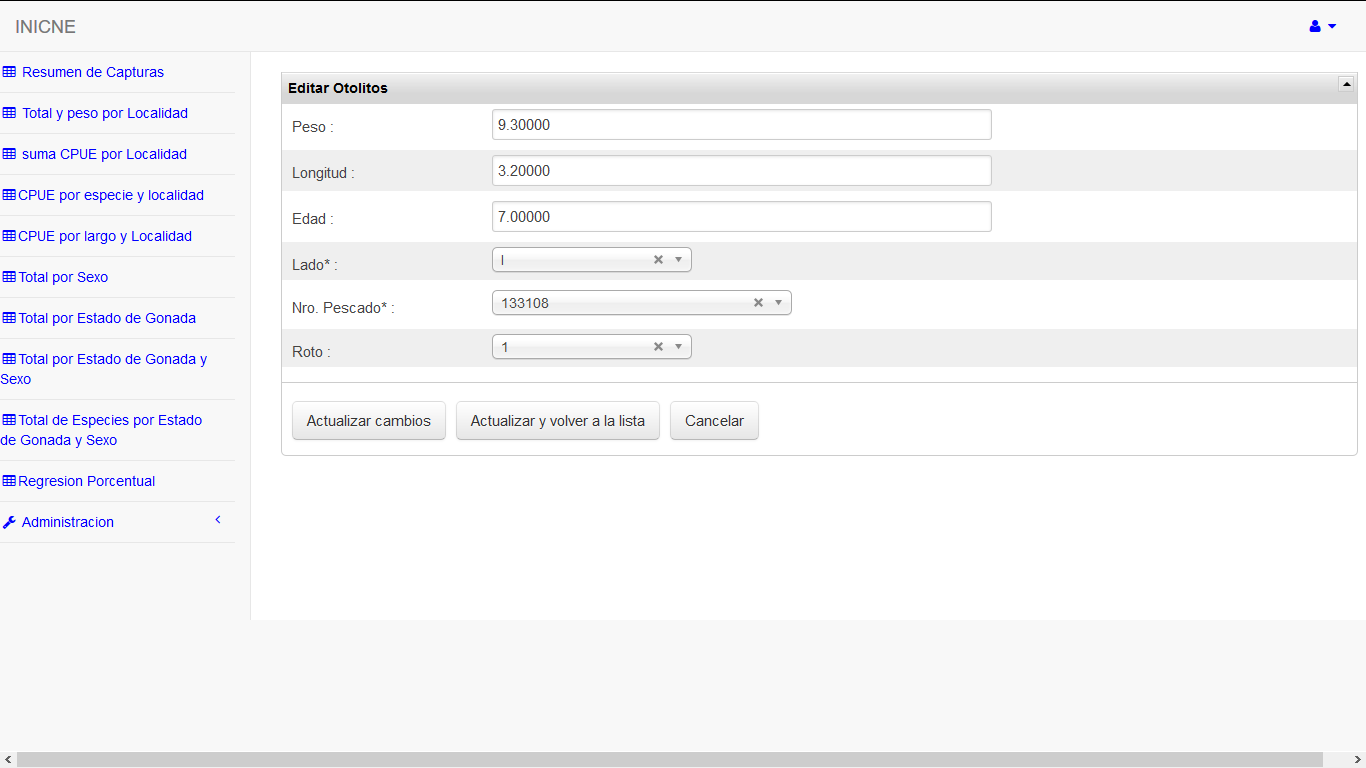


Fig. 29: Menú de modificación

El menú de altas y el de modificaciones son iguales, con la diferencia que el primero no inicia pre-cargado, para facilitar la carga en campos que representan una relación a otra tabla, se añadieron ComboBox, mostrando las opciones y evitando tener que buscar la misma antes de insertar.

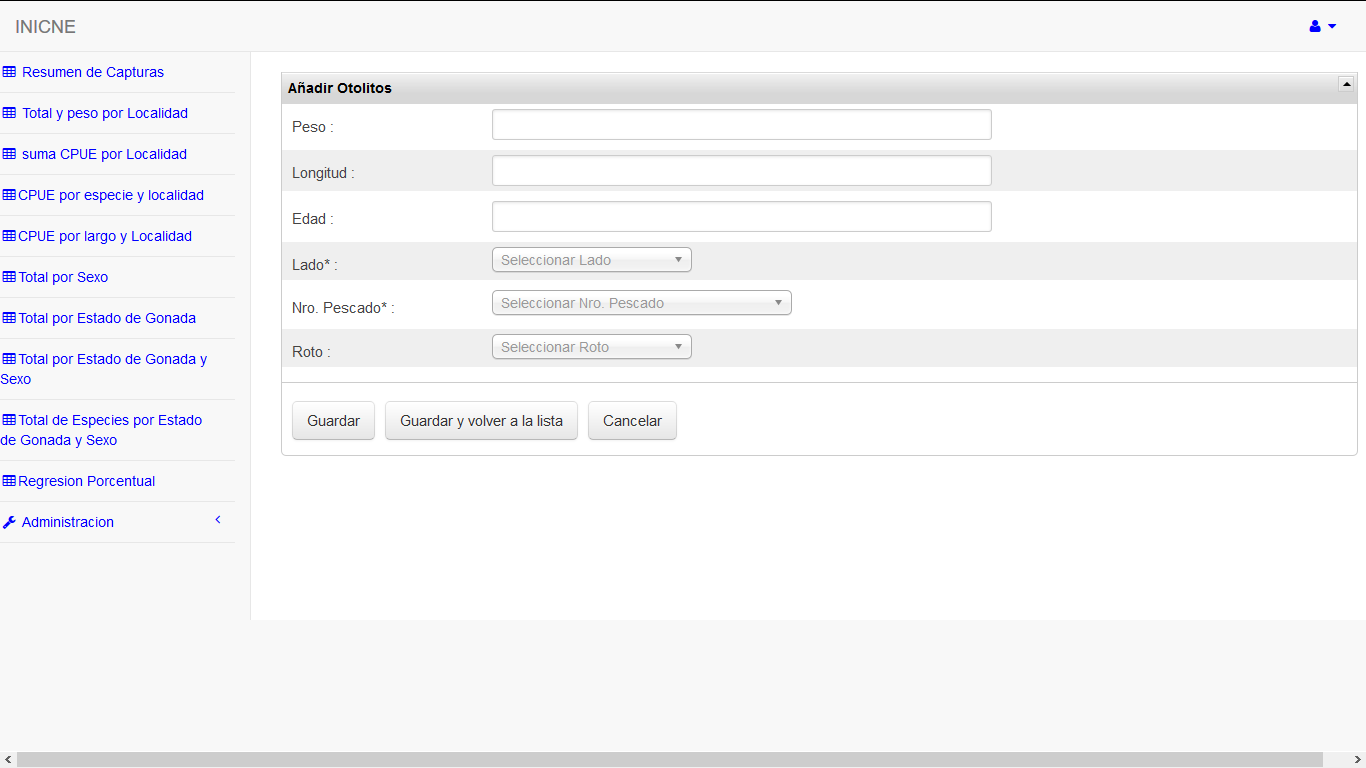


Fig. 30: Menú de altas

Al finalizar la carga o modificación se pueden observar 3 botones en la parte de abajo, el primero solo guarda la información, permitiendo realizar otra alta o modificación, el segundo realiza la acción y vuelve al menú de AMB, y el ultimo cancela la acción sin impactar en la base de datos.

4.3 Medidas preventivas

Si bien el desarrollo del software se realizó en función a las actividades asignadas a cada Sprint, es esta sección, para facilitar la comprensión y mantener una linealidad con el funcionamiento de la versión final de la aplicación se decidió por romper con ese esquema e ir explicando con las medidas adoptadas de forma definitiva a medida que se las encontraría mientras se interactúa con la aplicación.

4.3.1 Niveles de acceso

El uso de un esquema MVC a dos niveles no fue una decisión trivial, al inicio del análisis se consideró el funcionamiento de la aplicación como dos módulos independientes: el backend (PHP) solo se encargaría de establecer conexión con la base de datos y brindaría los datos, mientras que el frontend(JavaScript) se encargaría de mostrar los datos campar las acciones del usuario para dar vida a la aplicación pero esto presento una falla de seguridad al momento de agregar niveles de usuario, JavaScript es interpretado por el cliente y las acciones las realiza sobre elementos del DOM existentes, por lo que se puede decidir hacer visible o no ciertos elementos, pero basta con conocer el nombre del elemento y tener conocimientos básicos de JavaScript para hacerlo visible, en otras palabras, no existe control de acceso del lado del cliente. De ahí la necesidad de combinar los dos entornos.

Creados los niveles de acceso, sin credenciales validas, solo podemos acceder a uno de los controladores de PHP, el responsable del login o inicio de sesión. Como fue mencionado anteriormente, es el punto de la aplicación con mayor superficie de contacto con amenazas externas, el centro de los ataques para poder tener acceso a la aplicación. Si bien en este paso no se puede realizar un ataque XSS directamente, no protegerlo sería un pase libre para llegar a un punto donde se pueda.

4.3.2 Login

Se consideró que un ataque de fuerza bruta sería muy poco probable e ineficiente, es especial sabiendo que es necesario conocer un usuario para poder realizarlo, pero debido a que por uso de buenas prácticas lleva a encriptar contraseñas antes de almacenarlas en la base de datos, se utilizó la función PHP “password\_hash” con el parámetro bcrypt, el cual le indica utilizar el algoritmo crypt\_blowfish, que crea un hash de 60 caracteres en base a un Sting, o una contraseña en este caso. A la hora de iniciar sesión, se utiliza la función “password\_verify” que comprar la contraseña ingresada con el hash almacenado en la base de datos, lo importante de esta última función es que realiza las comparaciones siempre llevan el mismo tiempo, por lo que se mitiga el riesgo de ataques de fuerza bruta por timming attack, (en donde se evalúa el tiempo que un algoritmo tarda en comparar un String con otro nuevo igual al anterior pero añadiendo un carácter adicional, si tarda más tiempo que con este último se supone que el ultimo carácter agregado es correcto).

Una vez validada la identidad del usuario vuelve a entrar en juego el control de acceso, dependiendo del rol del usuario se enviaran o no ciertos campos, se tomó esta medida debido a que si solo deshabilitamos el botón que lleva a otra área, el mismo botón provee información de la ruta del área a la cual no tiene acceso, y mientras menos información tenga un atacante, más difícil resulta planear el ataque.

Como elemento adicional al control de acceso, se creó un controlador padre del cual heredan el resto de los controladores, este contiene tres clases, correspondientes a los 3 roles de usuarios, que se encarga de controlar los accesos a métodos de los controladores. Cada uno de los demás controladores se encargan de proporcional las API o vistas, dependiendo el nivel de acceso que estas requieran, el controlador hereda de una de las clases, por lo que de no tener el nivel de acceso requerido, no solo no se muestra en enlace a la funcionalidad, sino que el controlador encargado de la misma no será instanciado, por lo que para esa sesión de ese usuario el mismo no existe.

4.3.3 Sesiones

Por otra parte al momento de iniciar sesión en la aplicación se crea un sesión de CodeIgniter, cada sesión, valida por 6 horas, que tiene un id único formado por un String aleatorio, el nombre del usuario y la dirección IP del dispositivo utilizado. Esta información se almacena tanto en el servidor como en el cliente en forma de cookie, en caso de que un atacante robe la sesión, esta no le servirá de nada a menos de que compartan, o al menos hagan creer al servidor que comparten la misma dirección IP, y solo será útil mientras el usuario original no cierre su sesión, a que esta es destruida dejando a la cookie inutilizable, para agregar un nivel más de seguridad e impedir el robo de sesiones, se añadió un control de CSRF [12].

4.3.4 Control de entradas

Todos los informes disponibles están parametrizados, lo que implica tener campos donde el usuario puede ingresar información que eventualmente impactara en el servidor, al tener que existir en la base de datos los parámetros que los usuarios pueden ingresar se utilizó ComboBoxs, dejando a elegir uno entre varias opciones y en caso de querer modificarlo antes de que impacte en el servidor, los datos son enviados al servidor en forma de un POST, con la particularidad que son embebidos como un objeto compuesto de parejas nombre - valor del campo, todo esto como nombre de un único atributo enviado, y sin valor, lo que obliga a realizar un análisis previo a poder reemplazar. Al momento de llegar al servidor, este controla que sea del tipo de dato correcto antes de enviarlo al modelo, el cual los vuelve a subsanar y llama a un procedimiento almacenado en la base de datos.

4.3.5 Control de salidas

A su vez, debía realizarse un control de los elementos mostrados en los informes, si bien son tablas provenientes de resultados de operaciones de bases de datos propia, la modificación del nombre de una localidad o de una especie por un script forzaría a todos los usuarios que intentan generar ese informe a ejecutar el script, para ello por un lado se especifica que el contenido de las API son texto plano, no ejecutable, y se utiliza una librería para renderizar las tablas y además funciona como un filtro adicional, encargándose de que todo script que tenga que renderizar, solo muestre sus campos como parte de sí misma y no como código ejecutable.

4.3.6 Menú de altas, bajas y modificaciones

Al ingresar al área de ABM la situación fue más complicada, si bien se utilizó una librería que facilito la creación de las vistas y las consultas a la base de datos esta recibió modificaciones antes de ser implementada.

Al ingresar a alguna de las funciones de ABM la librería mostraba una lista de los elementos en la base de datos, correspondientes a esa tabla, separado en columnas por atributos, incrustando en el HTML el contenido de la base de datos mediante bucles de PHP, por lo que todo código ejecutable almacenado en la base de datos, era ejecutado por el navegador inmediatamente al cargar dicha vista, para solucionar esto se analizó el código de la librería, y al momento de recuperar la información de la base de datos y antes de incrustarlo en el HTML se toma la misma y se utiliza una de las funciones recomendadas por OWASP htmlspecialchars que convierte caracteres especiales en entidades HTML, lo que significa que cualquier carácter especial en la codificación utilizada en dicho sitio es reemplazado por su representación HTML y pierden su carácter de especial. Para que esta función actúe correctamente se debió verificar que todos los encabezados HTML indiquen la misma codificación, debido al uso del latín para la nomenclatura de las especies se optó por UTF-8, pero de no aclararlo se podía ingresar caracteres especiales UTF-7 y serian ejecutados.

Avanzando, la aplicación permite ver la información completa de un registro seleccionado, en la cual se aplicaron los mismos cambios que en la anterior, y por último y la sección más crítica, las altas y modificaciones. De no ser precavidos un usuario podría ingresar un script creando un XSS almacenado o inyectar una sentencia SQL que borre una tabla, la base de datos completa.

Para cada uno de los campos se creó un filtro manual, indicando el tipo de dato aceptado, lo que simplifico el caso de fechas, valores provenientes de otras tablas números, en el caso de campos alfanuméricos como los nombres de usuario, campañas o especies esto continuaba siendo un problema, por lo que se agregó un filtro más que controla que solo sean caracteres alfanuméricos, y quita cualquier etiqueta HTML que tenga evitando posibles scripts o inyecciones. Por parte de la base de datos también se limitó la longitud de los campos de tipo String, por lo que la mayoría de los ataques XSS ofuscados o codificados en ASCII, hexadecimal, octal u otros no tienen suficiente longitud para almacenarse de forma completa en la base de datos. En el caso de que un atacante gane acceso a la base de datos e inserte directamente un Script, lógicamente no va a pasar por el filtro XSS que elimina las etiquetas, pero siguen activos los filtros que controlan las salidas de la aplicación, que evitan que estos se ejecuten, en vez de eso solo se los muestra como cualquier otro valor, permitiendo detectar una intrusión al sistema.

4.4 Respuesta del software ante ataques

Aquí se comprobara lo anteriormente dicho, iniciando por el login. Al iniciarse la conexión con la base de datos al inicio de la aplicación, existe la posibilidad de inyectar código SQL en el mismo sin la necesidad de credenciales validas, como se puede ver en la Fig. 31, se utiliza una inyección SQL, ingresando un String sin significado, cerrando las comillas que se utilizarían al formular la consulta del login, e introduciendo código SQL adicional, seguido de un signo “#” que sirve para comentar el resto de la consulta original. Como se puede observar en la respuesta del sistema, lo toma como un simple error de credenciales.

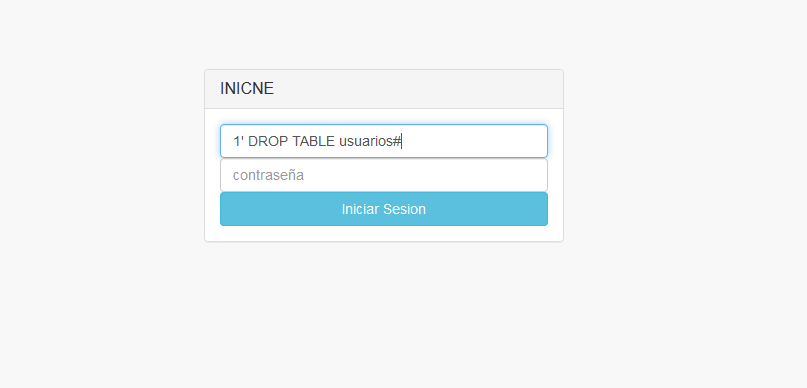


Fig. 31: validación de login

Como se mencionó en la sección anterior, almacenar las contraseñas en texto plano no trae consigo ninguna ventaja, por lo que en la base de datos solamente se almacenan hashes de las mismas. La diferencia entre encriptar y hashear es que la encriptación es un proceso reversible, en cambio al usar hash no hay forma de saber que String lo origino.

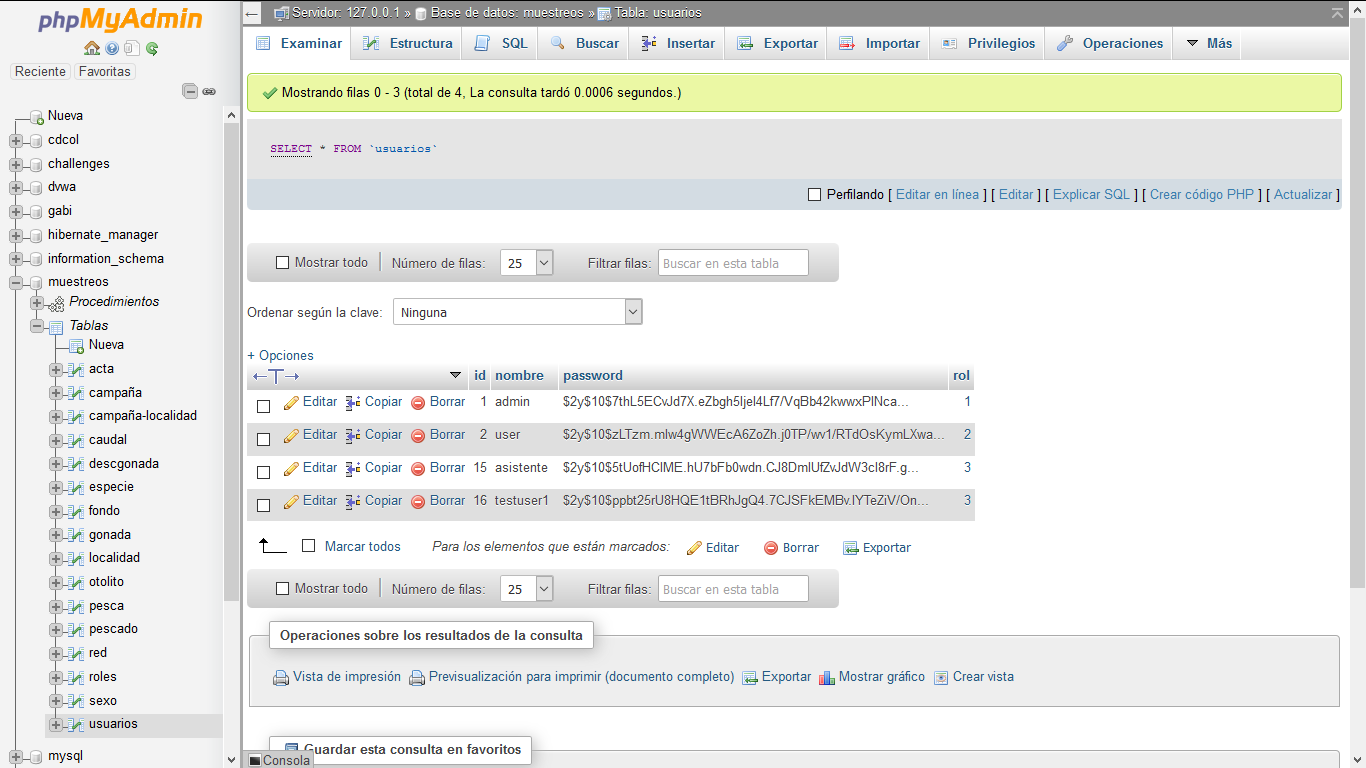


Fig. 32: Almacenamiento de claves

Continuando con el control de acceso, como ya se mencionó antes, un administrador puede acceder a la totalidad de las funciones de la aplicación, mientras que un usuario no.

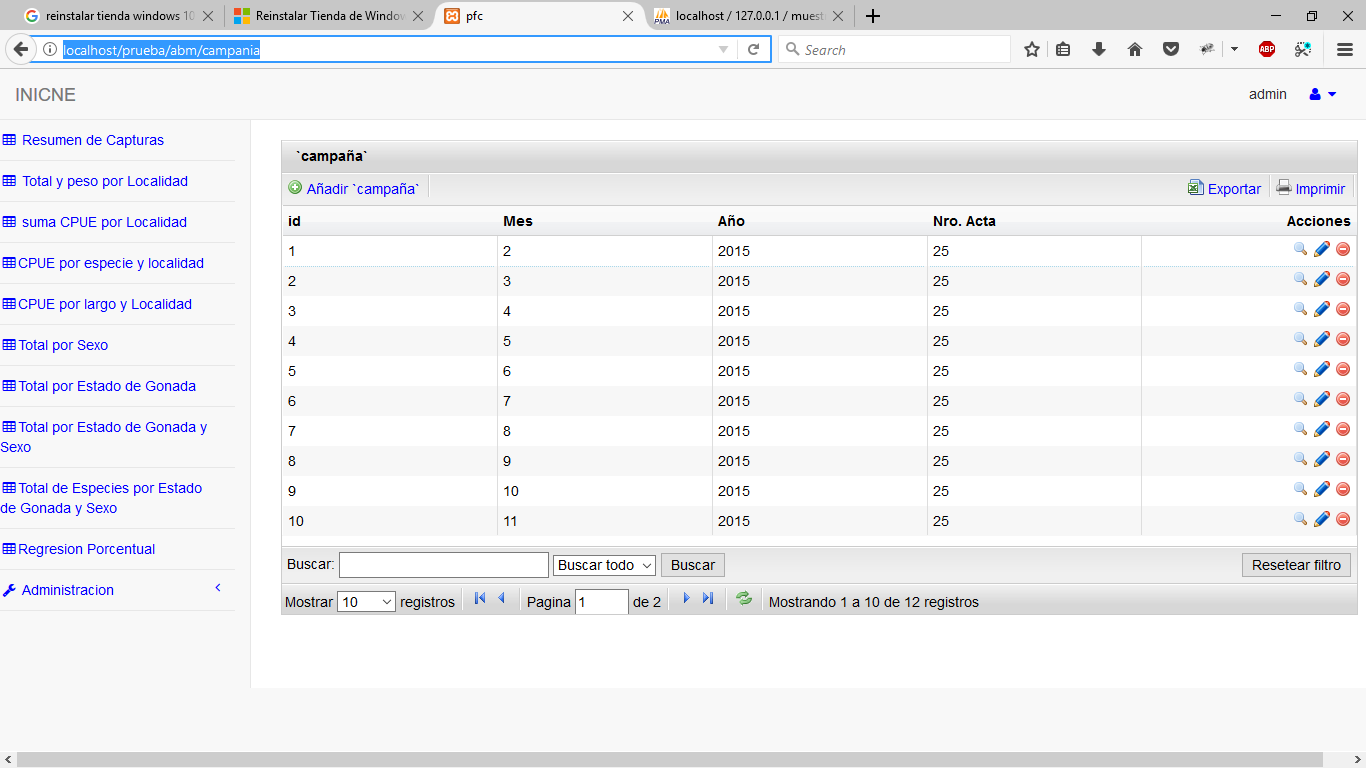


Fig. 33: Control de acceso, menú admin

Para verificar esto se inició de la forma más básica, se ingresó como administrador, se copió la URL de un controlador que provee acceso a una función restringida para los usuarios, se ingresó en otro navegador con el perfil de usuario y se intentó acceder a esta URL. La aplicación respondió como era esperado, re direccionando al usuario a la pantalla principal.

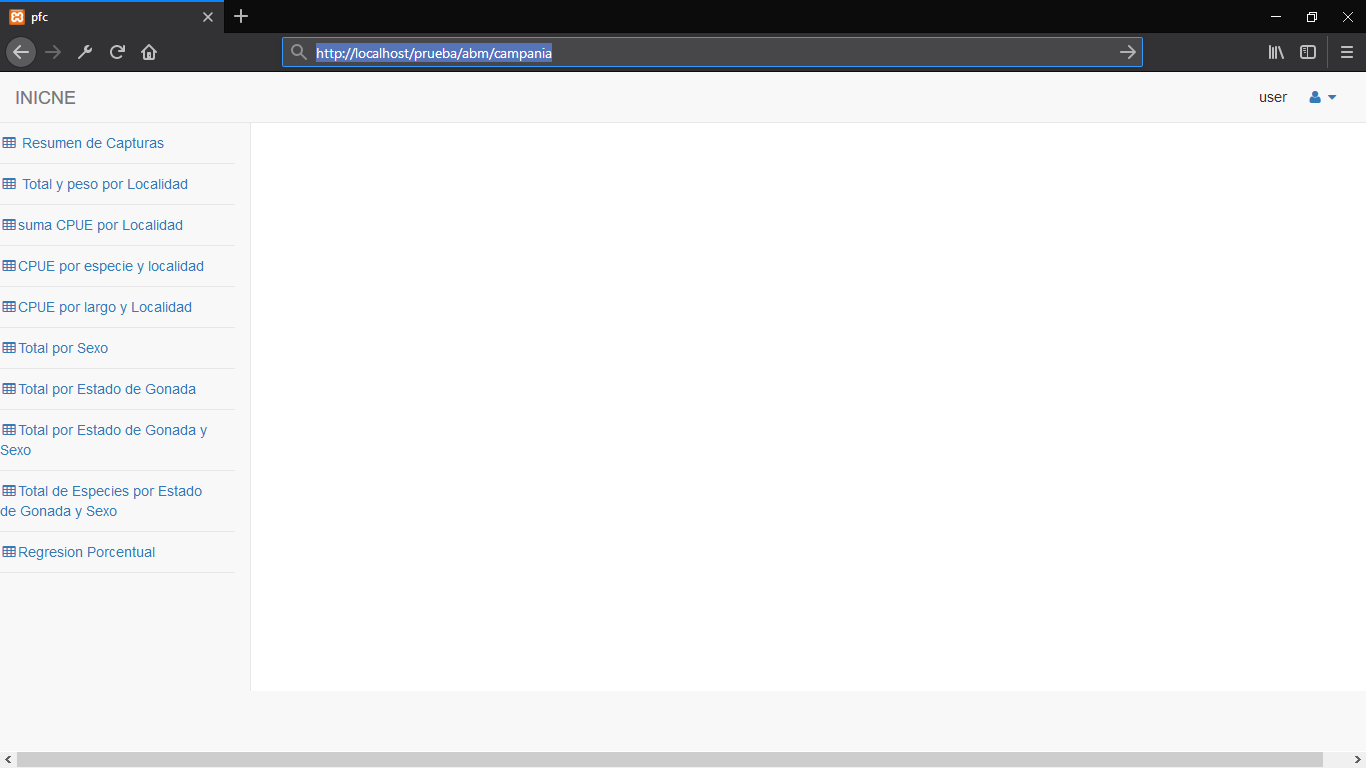


Fig. 34: Control de acceso, menú de usuario

Una vez comprobado eso, se prosiguió con la validación de entradas en distintos formularios, dentro de la carga de datos relativos a la aplicación no hay ninguna razón para recibir caracteres que no sean alfanuméricos, por lo que en dicho área, el ingreso de scripts no es exitoso.

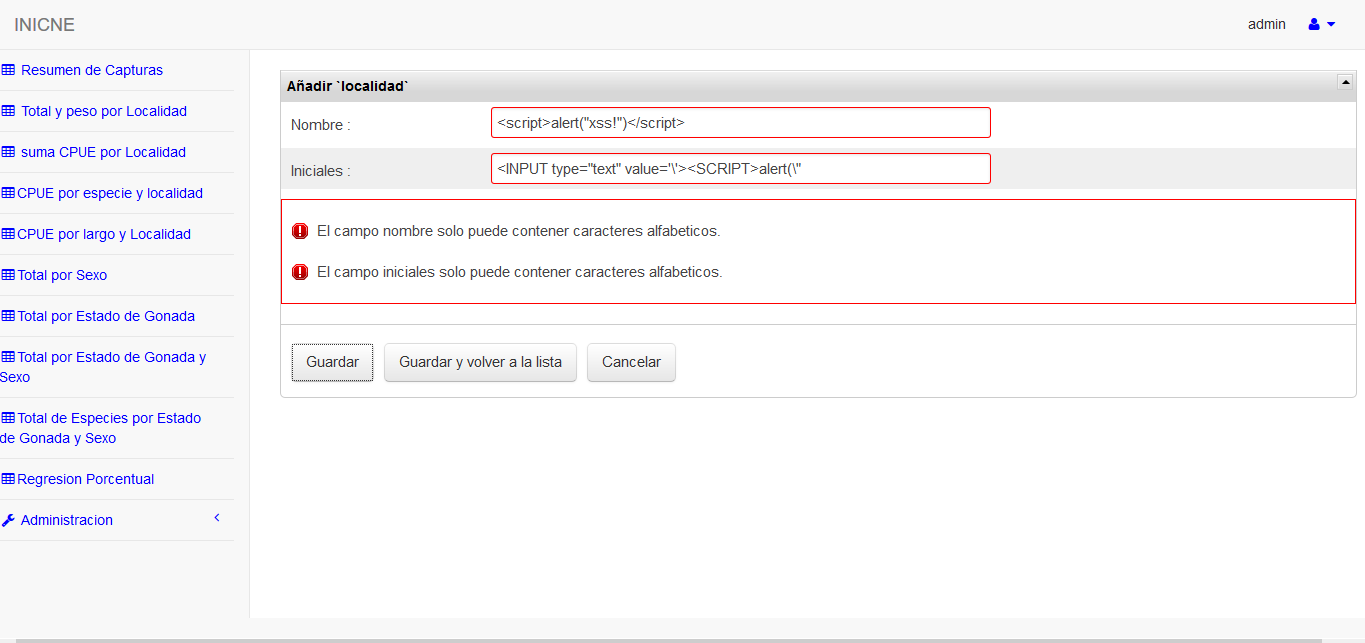


Fig. 35: validación de entradas: AMB

Por otra parte, para los usuarios se permitió el uso de caracteres especiales, principalmente para permitir el uso del guion y guion bajo, por lo que el ingresar un Script, este es tomado por la aplicación, luego, previa al impacto en la base de datos, se ejecuta un filtro encargado de verificar que el nombre ingresado no posea ningún tipo de etiqueta, y en caso contrario, las remueve.

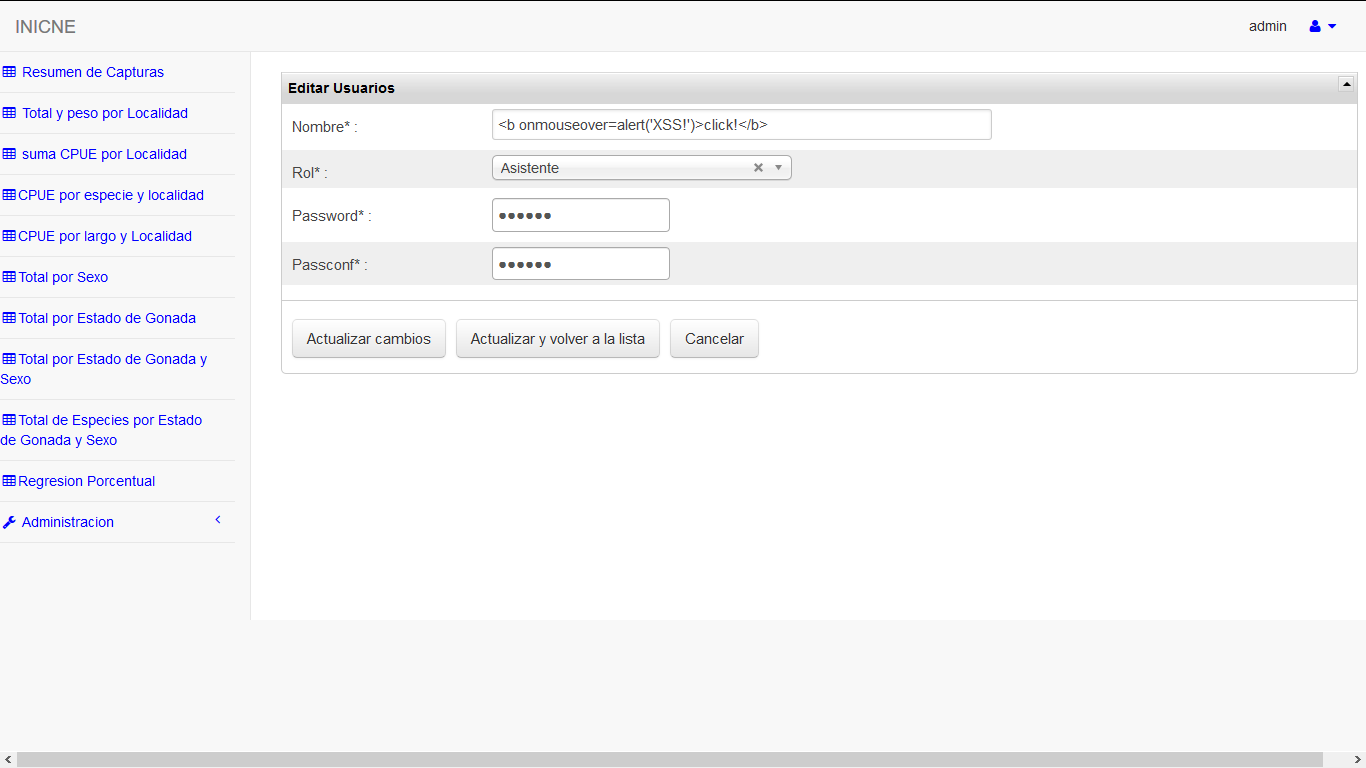


Fig. 36: validación de entradas: usuarios

Como se puede observar en la Fig. 37, el impacto en la base de datos fue únicamente del texto incluido entre las etiquetas, el resto fue descartado.

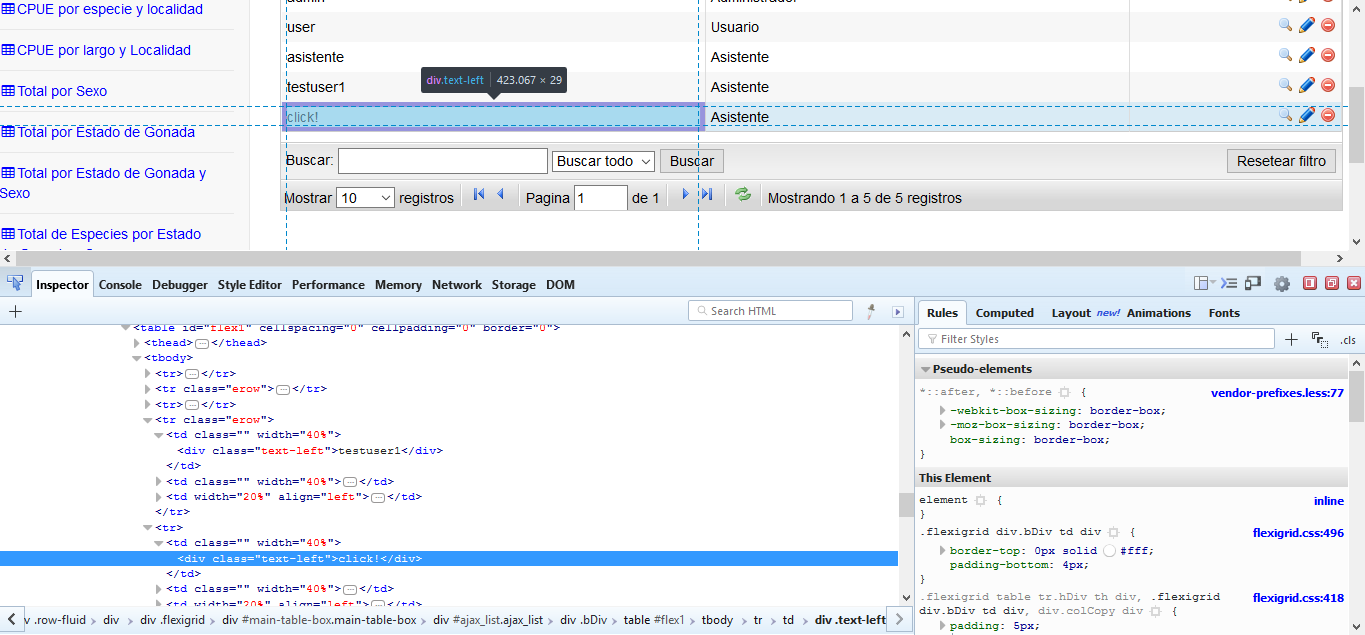


Fig. 37: validación de entradas: respuesta del sistema

Esto no solo sucede con etiquetas HTML, sino también con otras como Script, XML y CSS lo que bloquea un amplio rango de ataques de inyección de código.

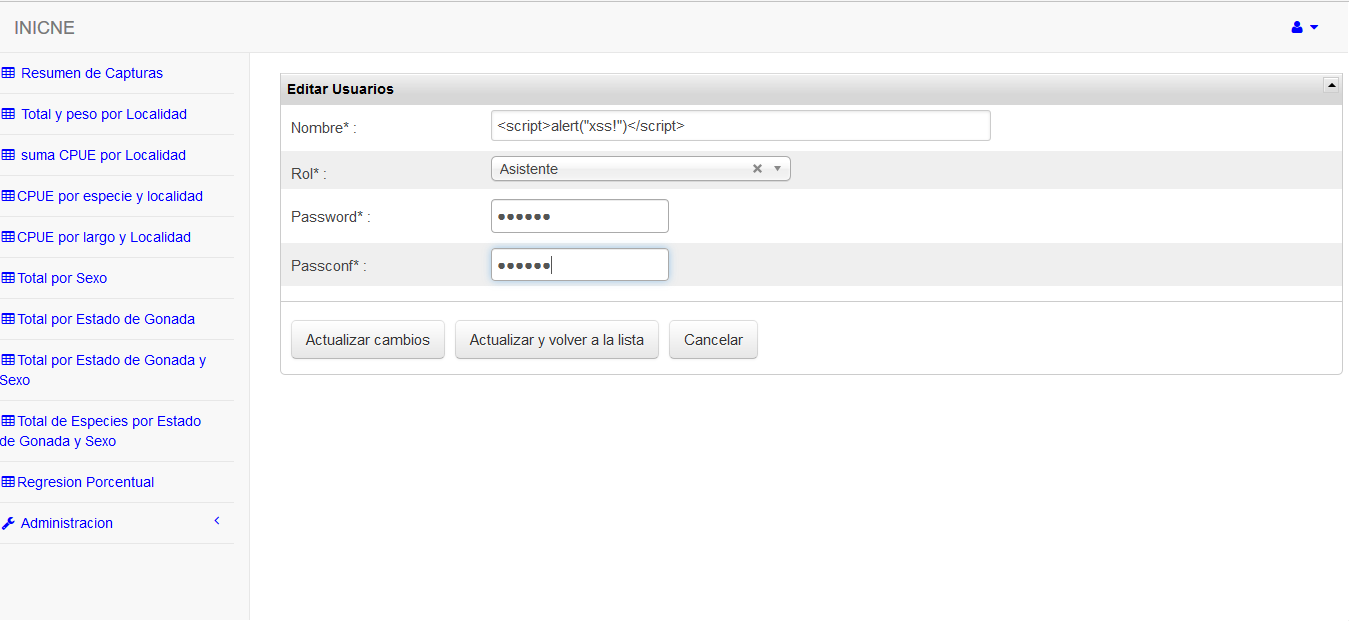


Fig. 38: validación de entradas con marcas de Script

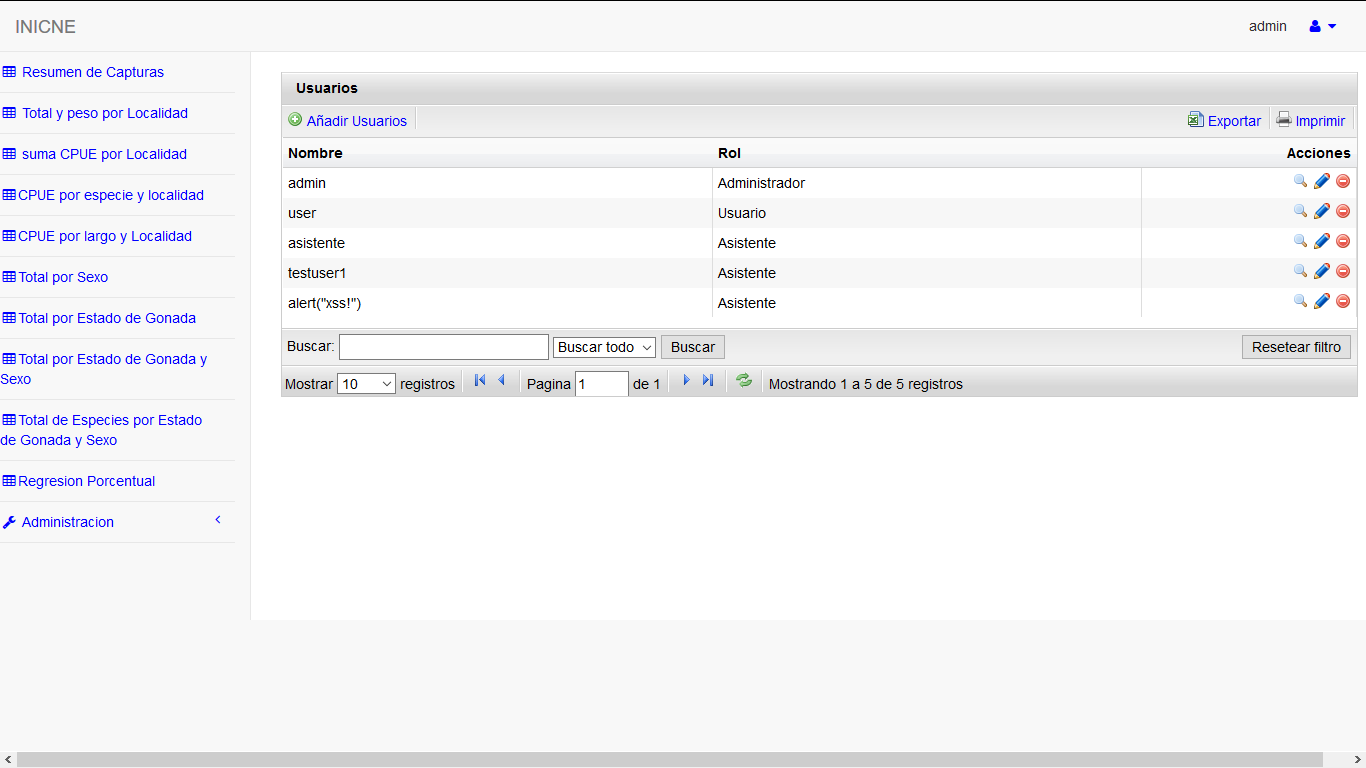


Fig. 39: resultado de validación de entradas con marcas de Script

Otro aspecto a tener en cuenta, según algunas bibliografías más importante que el anterior, es la validación de salidas, es decir, todo elemento que la aplicación muestre a los usuarios, y sea creado por los mismos usuarios. Idealmente si los filtros de entrada son buenos no habría porque preocuparse por las salidas, pero existe la posibilidad de que un atacante haya explotado una vulnerabilidad antes de ser parcheada, o de inyección directa en la base de datos, como también sirve como un doble control ante fallas del filtro de entrada.

Para probar los mismos se insertaron scripts directamente en la base de datos y luego se ejecutó la aplicación, pasando por el menú de AMB, y los informes que mostrarían ese dato. En una primera instancia el código se ejecutaba en el menú AMB, luego de modificaciones en la librería, se volvieron a ejecutar las pruebas y solo se mostraba como información estática, al igual que en los informes.

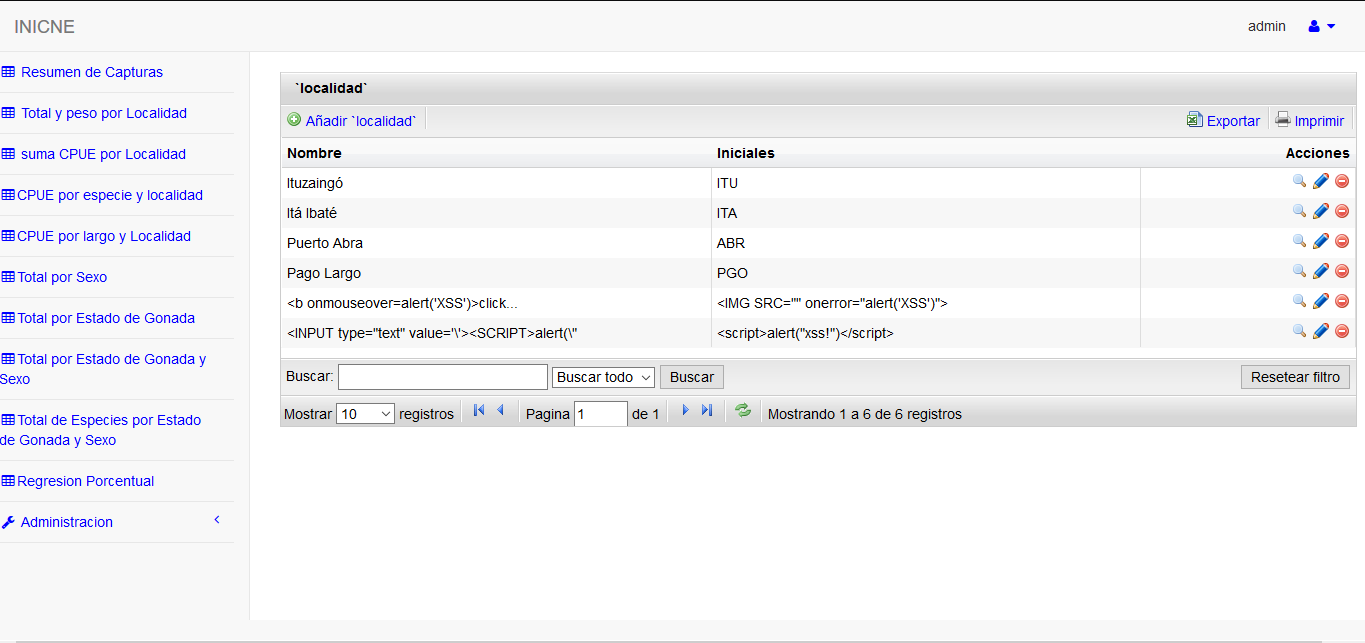


Fig. 40: Respuesta ante Scripts insertados

Continuando con la lista de ataques a probar se encuentra el robo de sesiones, lo cual resulta increíblemente fácil de realizar una vez inyectado un XSS. Debido que los mismos ya no se ejecutaban en la aplicación, se utilizó TamperData para extraer la cookie de sesión de usuario administrador, activa en una ventana de Mozilla Firefox

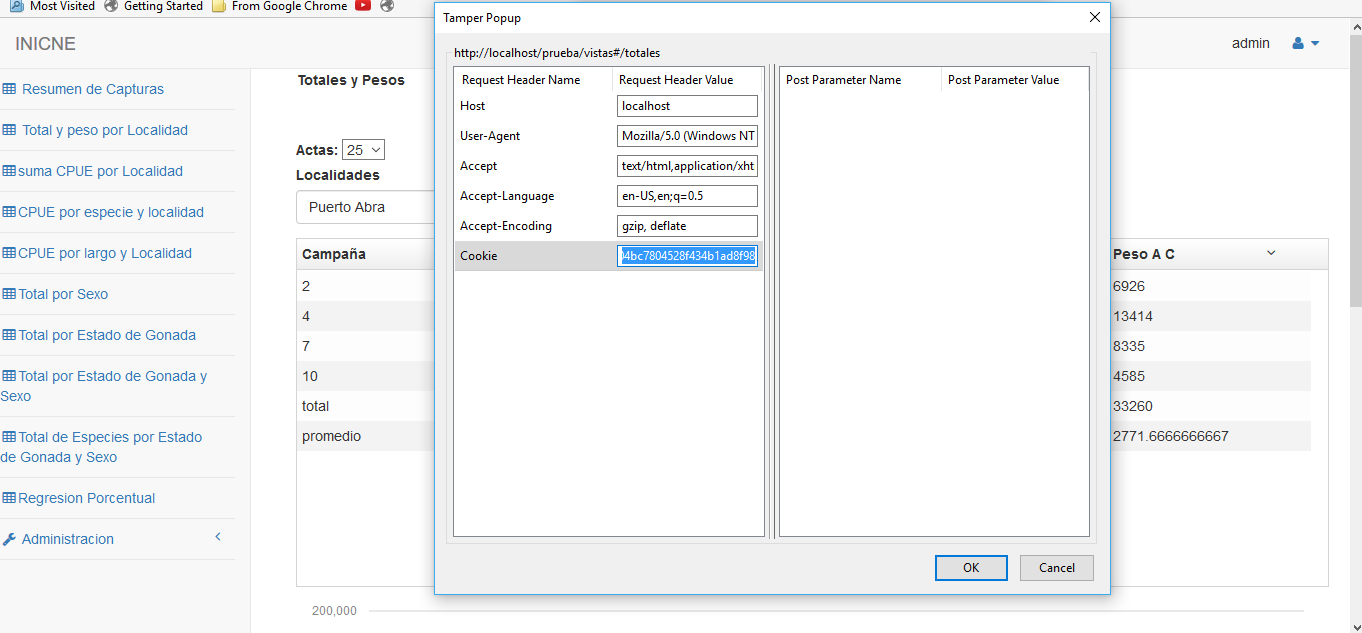


Fig. 41: Robo de sesiones: victima

Por otra parte, se inició una sesión de usuario en Google Chrome, se realizó una petición http, se utilizó TamperData para modificar la sesión de usuario reemplazarla por la de administrador. En la primera prueba el robo de sesión fue un éxito, por lo que se implementó el uso de token CSRF, que genera un código aleatorio único para cada petición.

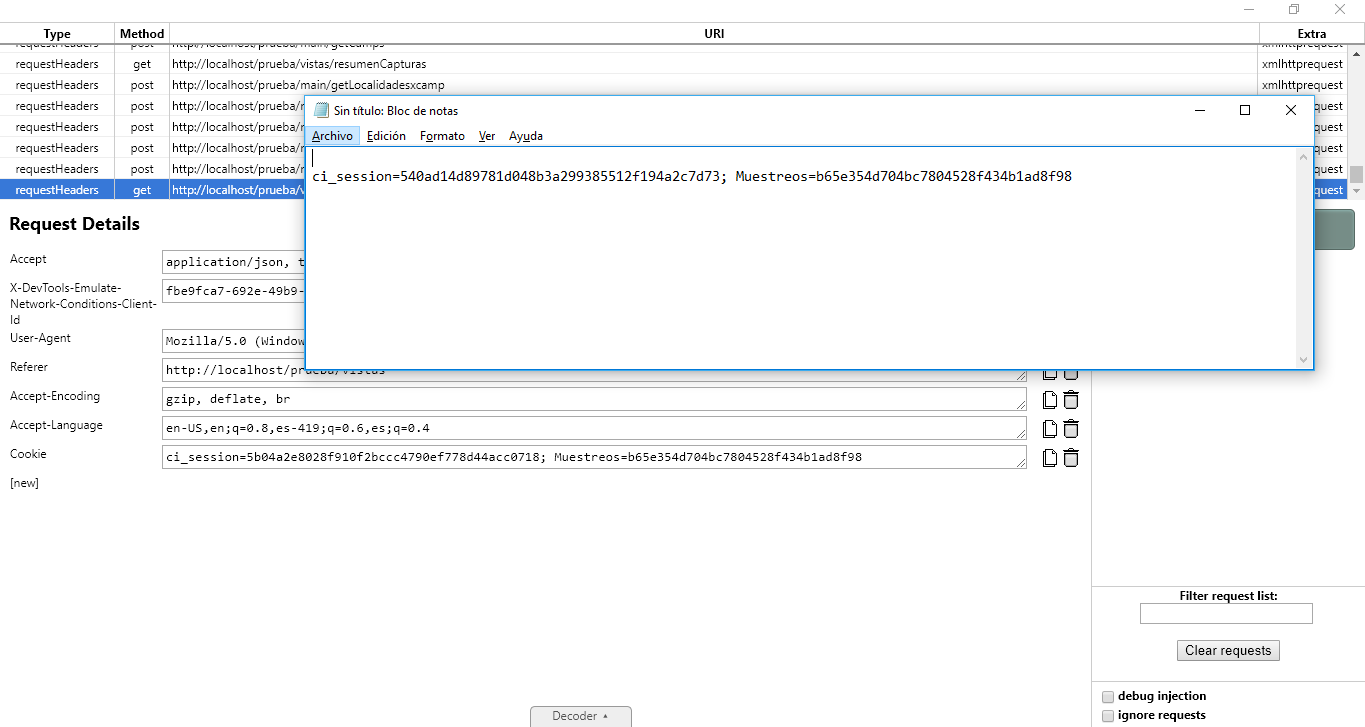


Fig. 42: Robo de sesiones: atacante

Se repitieron las pruebas, procediendo de igual forma, salvo que esta vez el resultado no fue exitoso, el usuario de Chrome fue redirigido al formulario que había solicitado, pero con sus credenciales iniciales

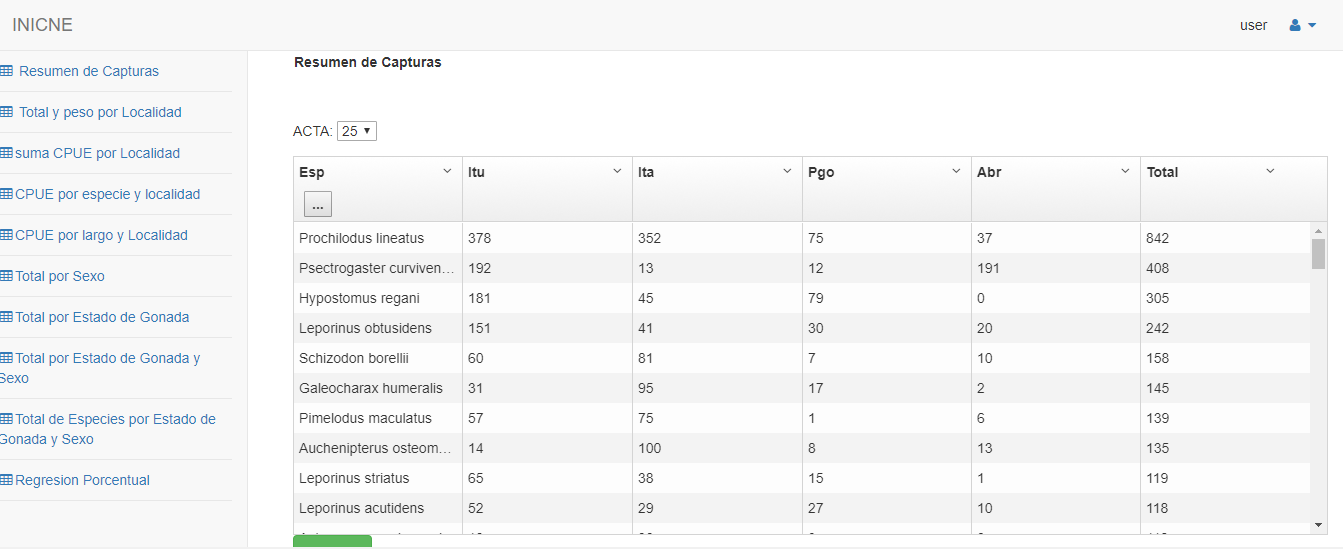


Fig. 43: Robo de sesiones: resultado

Para terminar con la etapa de pruebas, se intentó modificar la información ingresada a la aplicación, antes de que impacte en el servidor. Esto permite realizar un Bypass de los filtros de entrada implementados en el cliente.

La forma de proceder fue similar a la anterior, en un formulario de altas o de modificación se presionó el botón de guardar, y se utilizó TamperData para interceptar el mensaje

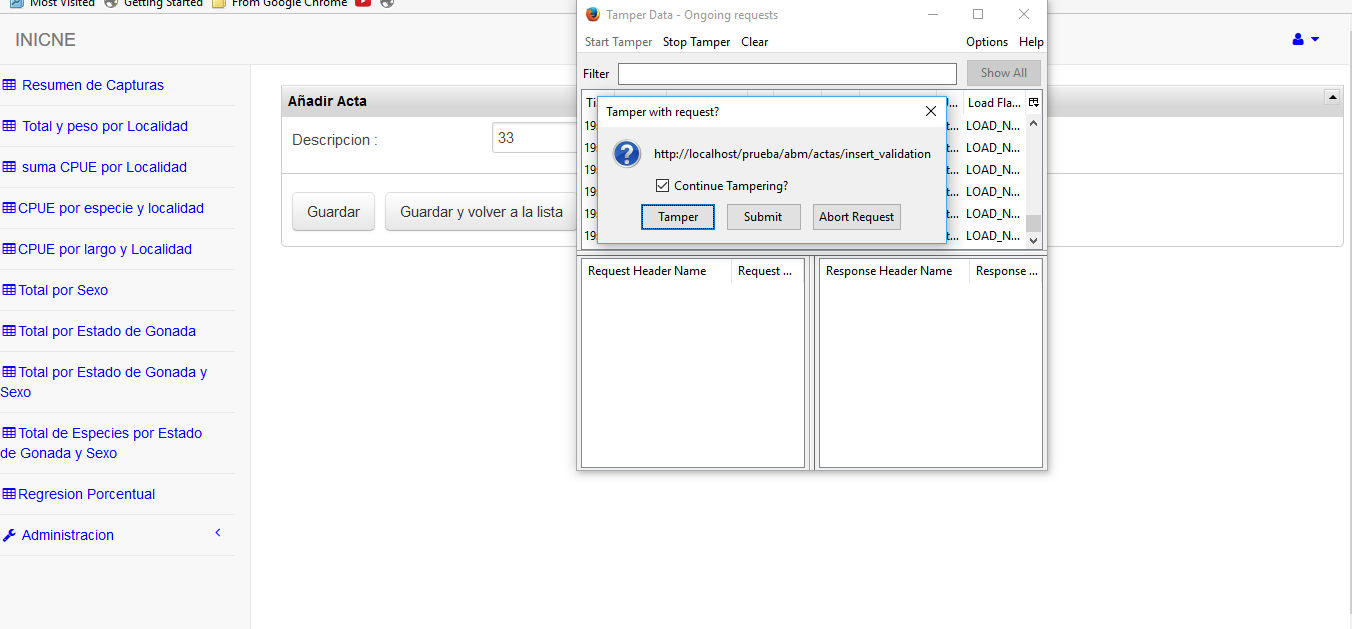


Fig. 44: Modificación de información: interceptar el mensaje

El mensaje fue reemplazado por un Script simple, que emite una alerta y se envió.

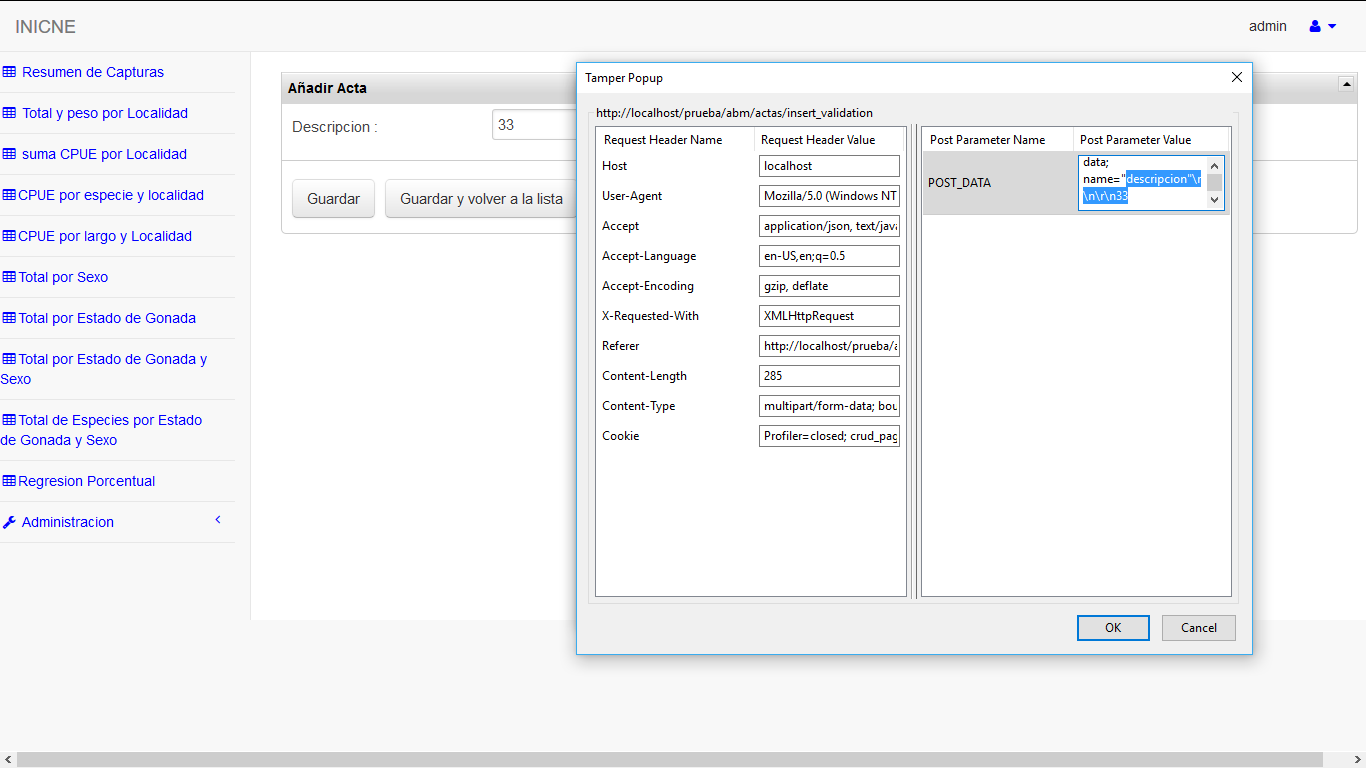


Fig. 45: Modificación de información: modificación del mensaje

El resultado fue insatisfactorio, el valor también se envía encriptado junto a los demás datos, por lo que la modificación no es efectiva y ,por más que lo fuese, el filtro esta aplicado a nivel servidor, por lo que en realidad solo se evita la validación de campos a nivel cliente.

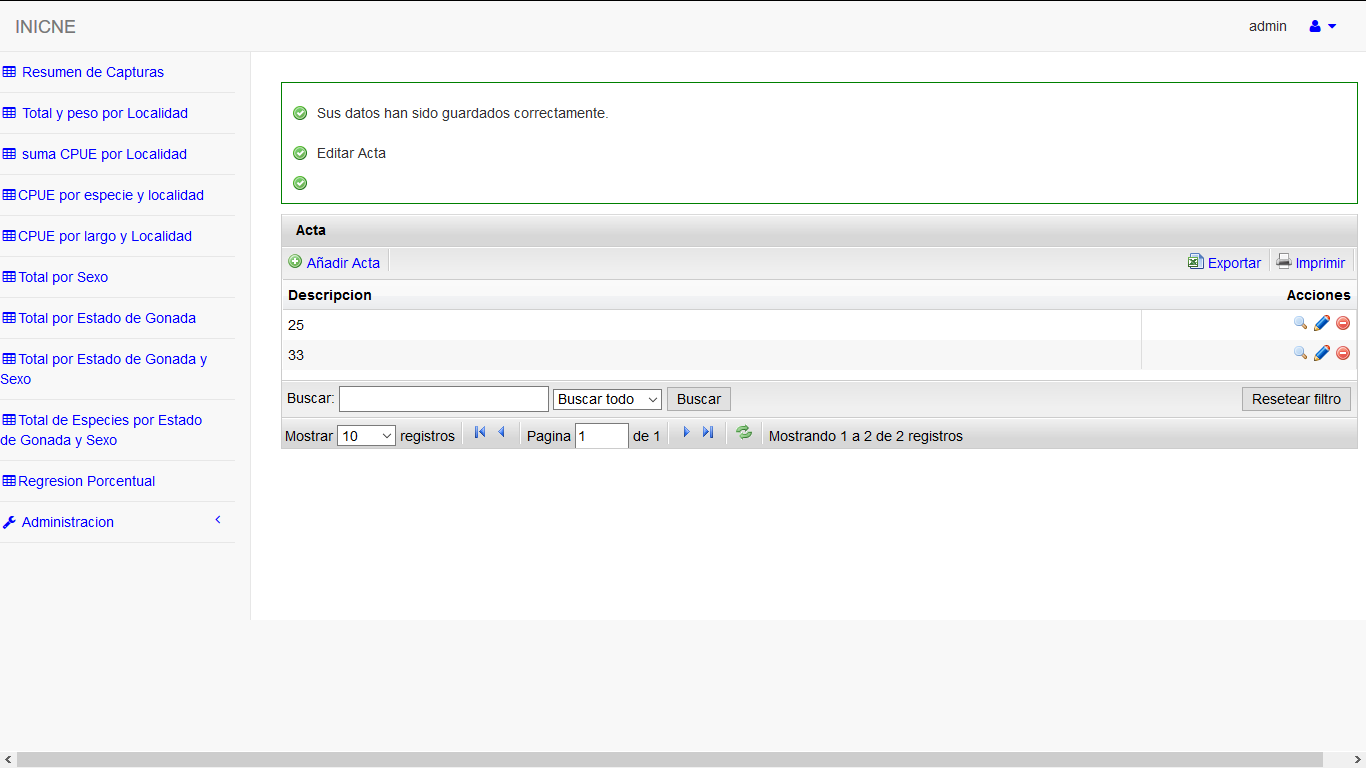


Fig. 46: Modificación de información: resultado

Capítulo 5. Conclusiones y futuros trabajos

Se pudo concluir de manera exitosa el desarrollo de la aplicación siguiendo un patrón MVC, realizando una división clara de las tareas entre el cliente y el servidor, y utilizando metodologías agiles (XP) lo que implicó un cambio en la forma de trabajar, aumentando la comunicación con el cliente y añadiendo incrementos a medida que se avanzaba con las historias de usuario. Esto resulto más notorio en el diseño de la base de datos donde, si bien, se podía diseñar en su totalidad en una primera instancia, se fueron anexando tablas y atributos cuando era necesario.

También se demostró la diferencia entre utilizar procedimientos almacenados y formular consultas mediante la librería propuesta por el framework utilizado para realizar consultas complejas, teniendo el primero ventajas de velocidad de respuesta y simplicidad.

A su vez se mostró un ejemplo de cómo desarrollar software seguro y como se pueden utilizar métodos específicos de seguridad y adaptarlos a la metodología usada. A pesar de estar orientado a evitar ataques XSS, a medida que se adquirían más conocimientos sobre estos se podía ver que distintos ataques a aplicaciones web estaban mucho más relacionados de lo que se había asumido antes de iniciar el proyecto. Demostrando que la seguridad se aplica al sistema de forma global y teniendo en cuenta muchos aspectos. De nada sirve proteger a la aplicación solamente en un aspecto porque como se mostró anteriormente, un ataque XSS puede llevar a un ataque CSRF, a un robo de sesión, puede redirigir a otro sitio, como así también si solo nos enfocamos a defender contra XSS, podemos ser vulnerables a los mismos a través de una inyección SQL u otro tipo de ataque.

Así como se aplicaron medidas de seguridad, tanto preventivas (filtros de entrada, uso de sesiones) como correctivas (filtros de salida), la aplicación está lejos de ser totalmente segura, siempre hay lugar para mejorar. Ciertos aspectos de seguridad quedaron fuera del alcance de este trabajo como ser un control de direcciones IP a la hora del login para evitar ataques de fuerza bruta y ataques de denegación de servicio. Las sesiones funcionan almacenándose en el servidor, podrían ser almacenadas en una tabla de la base de datos para tener un mejor control de los usuarios, como también se podrían agregar tablas de auditoria, que sirvan para saber que usuario realizo las modificaciones y también para poder volver a ataras en caso de la dicho modificación fuese errónea.

# 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] | Attack and Defense Labs, «Ravan,» [En línea]. Available: http://www.andlabs.org/tools/ravan.html. [Último acceso: 25 10 2016]. |
| [17] | M. Heiderich, E. A. Vela Nava, G. Heyes y D. Lindsa, Web Application Obfuscation ‘-/WAFs..Evasion..Filters//alert(/Obfuscation/)-’, Burlington, MA: Elsevier, 2011. |
| [18] | P. Brady, «Survive the Deep End: PHP Security,» 26 Octubre 2015. [En línea]. Available: http://phpsecurity.readthedocs.org/en/latest/. |
| [19] | 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]. |
| [20] | 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. |
| [21] | 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. |
| [22] | PHP Group, «PHP,» [En línea]. Available: http://php.net/manual/es/intro-whatis.php. [Último acceso: 30 10 2017]. |
| [23] | EllisLab , «CodeIgniter,» [En línea]. Available: https://codeigniter.com/. [Último acceso: 30 10 2017]. |
| [24] | C. Wells, «A Weak Foundation,» de *Securing Ajax Applications*, O’Reilly, 2007, pp. 152-154. |
| [25] | A. t. Holdener, Ajax: the Definite Guide, O’Reilly Media, Inc.,, 2008. |
| [26] | Google, «AngularJS,» [En línea]. Available: https://angularjs.org/. [Último acceso: 30 10 2017]. |
| [27] | Oracle Corporation, «MySQL,» [En línea]. Available: https://www.mysql.com/. [Último acceso: 30 10 2017]. |
| [28] | Oracle Corporation , «MySQL,» [En línea]. Available: https://www.mysql.com/products/workbench/. [Último acceso: 30 10 2017]. |
| [29] | Hitachi Vantara, «Pentaho,» [En línea]. Available: http://www.pentaho.com/. [Último acceso: 30 10 2017]. |
| [30] | Software Freedom Conservancy, «Git,» [En línea]. Available: https://git-scm.com/. [Último acceso: 30 10 2017]. |
| [31] | npm, Inc, «NPM,» [En línea]. Available: https://www.npmjs.com. [Último acceso: 30 10 2017]. |
| [32] | Apache Friends, «XAMPP,» [En línea]. Available: https://www.apachefriends.org/es/index.html. [Último acceso: 30 10 2017]. |
| [33] | EasBacklog, «EasBacklog,» [En línea]. Available: https://easybacklog.com/. [Último acceso: 30 10 2017]. |
| [34] | mozilla.org, «Mozilla,» [En línea]. Available: https://www.mozilla.org/es-AR/. [Último acceso: 31 10 2017]. |
| [35] | Mozilla.org, «Fireug,» [En línea]. Available: https://getfirebug.com/. [Último acceso: 31 10 2017]. |
| [36] | A. Judson, «Mozdev,» [En línea]. Available: http://tamperdata.mozdev.org/. [Último acceso: 31 10 2017]. |
| [37] | B. G. Santhosh , K. M. Vineet , J. Ebenezer, S. V. Muni , T. K. Asoke y R. . P. Alwyn , «Suraksha: A Security Designers’ Workbench,» Hack.in, Kanpur, India, 2009. |
| [38] | 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. |