Universidad de las Ciencias Informáticas
Facultad 1

Módulo de JavaScript para la herramienta Auditoría de Código Fuente

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Autor:
Marlon Rodriguez Garcia

Tutores:
Ing. Michel Evaristo Febles Parker
Ing. Abel Alfonso Fírvida Donéstevez

La Habana Junio de 2015
"Año 57 de la Revolución"

La revolución no se lleva en los labios para vivir de ella, se lleva en el corazón para morir por ella.

Ernesto 'Che' Guevara de la Serna

Declaración de autoría

Declaro ser el autor del presente Trabajo de Diploma y se reconoce a la Universidad de las Ciencias Informáticas, los derechos patrimoniales del mismo con carácter exclusivo. Para que así conste firmo la presente declaración jurada de autoría en Ciudad de La Habana a los _____días del mes de _____ del año _____.

Marlon Rodriguez Garcia

_____

Firma del Autor

Michel Evaristo Febles Parker

_____

Firma del Tutor

Abel Alfonso Fírvida Donéstevez

_____

Firma del Tutor

Dedicatoria

A Fidel y Zoila(Titi) que se que me cuidan desde donde sea que se encuentren.

A Oscar y Judy que me han enseñado a ser mejor cada día, y son mi mas grande tesoro.

A Maye y Aldo que me dieron la vida e hicieron posible todos mis sueños.

A Yudita, Naivis y al pequeño Willian, que son mi motivación y espero ser su ejemplo.

A Glenda, por la espera, la dedicación, los buenos y malos ratos, por todo el amor que siempre me has hecho sentir.

Agradecimientos

A mis tutores por toda la ayuda que me prestaron a lo largo de la realización de este trabajo.

A mi tio Ernesto que fue mi primer profe en la universidad, más que mi tio mi amigo, mi hermano.

A la gente del mejor grupo de la UCI, el 1508, Ricard, Annareya, Daniel, Manuel, Yanet, Yisel, Neyvis, Ariagna, Yileni, Mayvis, Ivis, Yoandri, Dayrol, Roger, Anaily, Angel, Annia por compartir los mejores 5 años de toda mi vida

A los que ya no forman parte de mi grupo pero que siguen siendo mis bros,Yunior, Dairon, Diwel, Raul y Luis

Al piquete de frikis que mas que amigos son familia, Marbier, Alexander, Javier, Vice, Enmanuel, Ariel, Joseph, Pita, Dennis, Polo, Grabiel.

A mi gente de Guantánamo, Alvaro, Javier, Pavel, Yasser ,Cesar, Adolfo, Alejandro, Manuel, Papo, Gabriel, Walter.

A esos amigos que aunque no están todo el tiempo forman parte de mi, Yaiselis, Yordanka, Rosa, Jose, Arturo, Evelyn.

A los profes del proyecto siempre dispuestos a ayudar, Dayrelis, Hector, Alexis, Haniel, Mairin, Yusleidys, Luis Daniel.

A los profes que durante 5 años marcaron mi vida y que nunca olvidaré, Ilmaris, Gabriel, Daylianis, Odelkis, Mónica, Gendry, Zumeta, Saura y Yeni.

A mi familia, por su educación, su apoyo y confianza , gracias a uds hoy estoy aquí.

A mi novia, por soportar todas mis malcriadeces, y decirme las palabras necesarias para seguir aun cuando todo parecía perdido.

Resumen

La detección de vulnerabilidades en el código fuente de las aplicaciones es una de las tareas más complicadas de los programadores en la actualidad. Para dicha tarea se desarrollan aplicaciones que permiten identificar funciones o fragmentos de código que sean vulnerables. El Centro de Soluciones Libres de la Universidad de las Ciencias Informáticas, creador de la distribución cubana de GNU/Linux Nova ha desarrollado la herramienta Auditoría de Código Fuente (ACF) la cual permite detectar funciones vulnerables en códigos fuente C/C++, Perl, Python y Bash, pero no así en JavaScript. La presente investigación tiene como objetivo desarrollar un módulo que será integrado a la herramienta ACF para la detección de vulnerabilidades en códigos fuente JavaScript. Para el desarrollo del mismo se utilizó la metodología Open Up, la herramienta de modelado Visual Paradigm, el entorno de desarrollo Qtcreator y el lenguaje de programación C++.

Palabras clave: Auditoría de Código Fuente, JavaScript, vulnerabilidades.

Índice de contenido

Índice de tablas

Índice de ilustraciones

Introducción

JavaScript es un lenguaje de programación ubicado como el primero de más utilización en los proyectos que tienen que ver con la Web. Actualmente existen más de 16 mil millones de líneas de código de JavaScript en *GitHub*[1] y alrededor de 716,889 preguntas relacionadas con el lenguaje en *StackOverflow*[2] (1). Ha prevalecido como el número uno en el diseño de interfaces y funcionalidades en el lado del cliente e incluso ha ganado fuerza en el lado del servidor desde el nacimiento de *Node.js*[3].

En la Universidad de las Ciencias Informáticas (UCI) se han desarrollado al menos 163 proyectos o módulos utilizando JavaScript y se conoce que el lenguaje cuenta con gran popularidad en la industria local de software. El Centro de Ideoinformática (CIDI) es el encargado de realizar la mayoría de los portales web del país, entre los que se encuentran: Soy Cuba, Somos jóvenes, Casa Editora Abril, LabioFam, Cubatravel, Alma Mater, Pionero y Artemiseño en los cuales el lenguaje JavaScript es la base en la programación del lado del cliente. Otros centros que usan este lenguaje en el lado del cliente son el Centro de Gobierno Electrónico (CEGEL), encargado de los proyectos Tribunales y Fiscalía, la Empresa de Tecnologías de la Información para la Defensa (XETID) y el Centro de Telemática con la Plataforma de Seguridad en las Tecnologías de la Información.

El Centro de Soluciones Libres (CESOL), creador del Sistema Operativo **Nova,** distribución cubana de GNU/Linux, ha desarrollado la herramienta Auditoría de Código de Fuente (ACF) que

1 *Suite colaborativa donde se alojan proyectos utilizando el sistema de control de versiones Gi.t*

2 *Sitio web utilizado por una comunidad de desarrolladores informáticos pueden encontrar soluciones a problemas de programación en diferentes lenguajes.*

3 ***Node.js** es un entorno de programación en la capa del servidor basado en el lenguaje de programación JavaScript, con I/O de datos en una arquitectura orientada a eventos y basado en el motor JavaScript V8.*

se utiliza para encontrar fallas de seguridad en el código fuente de cualquier aplicación o librería que esté o se agregue a su repositorio[4]. Estas revisiones permiten detectar dichas fallas en lenguajes tales como: C, C++, Python, Bash y Perl, brindando mayor seguridad para las aplicaciones desarrolladas en estos lenguajes. Sin embargo, hasta ahora el proyecto ACF no soporta la revisión de código JavaScript, lo que podría conllevar a la inclusión de vulnerabilidades en el repositorio.

Partiendo de esta situación se eligió como **problema de investigación:** ¿Cómo detectar vulnerabilidades en códigos fuente JavaScript en el repositorio de Nova haciendo uso de la herramienta ACF?

Se definió como **objeto de estudio** el proceso de detección de las vulnerabilidades del código fuente del lenguaje JavaScript. Para darle solución al problema anteriormente planteado se define como **objetivo general** desarrollar un módulo utilizando las técnicas de recolección de vulnerabilidades de JavaScript que permita a la herramienta ACF la revisión de código en JavaScript del repositorio de Nova y se establece como **campo de acción** el código JavaScript del repositorio de Nova.

Para darle cumplimiento al objetivo general planteado, el mismo se dividió en los siguientes **objetivos específicos:**

- Analizar las herramientas de auditoría de código JavaScript existentes enfocándose en las más utilizadas actualmente.
- Diseñar un módulo para la herramienta ACF que permita la detección de vulnerabilidades

[4] *Colección de paquetes de programas que pueden ser descargados e instalados por los usuarios de la distribución correspondiente.*

en el código fuente JavaScript.

- Implementar las funcionalidades del módulo para la herramienta ACF.
- Probar las funcionalidades del módulo para la herramienta ACF.

Como **idea a defender** se plantea que la elaboración del módulo para la detección de vulnerabilidades del lenguaje JavaScript en la aplicación ACF puede aumentar la seguridad de las aplicaciones que se incluyan en el repositorio de Nova.

Para cumplir lo antes expuesto, se elaboraron las siguientes **tareas de investigación:**

- Revisión de bibliografía asociada a las herramientas de auditoría de código JavaScript.
- Definición de los requisitos funcionales y no funcionales para el módulo a desarrollar.
- Análisis de la arquitectura de ACF.
- Diseño del módulo acoplado a la arquitectura de ACF.
- Codificación de las funcionalidades definidas.
- Diseño y ejecución de casos de prueba a las funcionalidades implementadas.

Se espera como resultado que la herramienta ACF será capaz de analizar y detectar vulnerabilidades en código JavaScript.

Los **métodos científicos** que se emplearon para desarrollar el proceso de investigación fueron:

El método **histórico-lógico** plantea que se debe estudiar la trayectoria real de los fenómenos y acontecimientos en el transcurso de una etapa o período analizando las leyes generales del

funcionamiento y desarrollo del fenómeno, estudiando su esencia (2). Este método fue empleado para analizar la evolución de las aplicaciones que forman parte del objeto de estudio para poder comprender las posibles soluciones del problema planteado.

El método **inductivo-deductivo** permite llegar a proposiciones generales a partir de hechos aislados que confirman la teoría o a partir de estas teorías arribar a conclusiones sobre casos particulares que se verifican en la práctica (2). Este método se utilizó para inferir, a partir de las aplicaciones encargadas de revisar código fuente JavaScript, la manera en que se efectúan estas revisiones.

El método **analítico-sintético** consiste en la extracción de las partes de un todo, con el objeto de estudiarlas y examinarlas por separado para luego sintetizarla y tomar los elementos más importantes (2). Este método se usó para realizar el análisis de grandes volúmenes de información y tomar lo que realmente era necesario para la investigación.

El método **revisión bibliográfica** permite identificar las fuentes arbitradas[5], identificar y comprender las ideas principales y sintetizar el argumento central de cada sección a través de la supresión y generalización de ideas principales (3). Dicho método fue utilizado para determinar las fuentes y referencias bibliografías óptimas y actualizadas para la elaboración de la investigación.

El método **modelación** permite realizar una representación o modelo para investigar la realidad. Es un instrumento de la investigación de carácter material o teórico, creado para reproducir el objeto que se está estudiando (2). Este método fue utilizado para realizar el diseño de la

[5] *Aquella que cuenta con un comité editorial que revisa con un procedimiento de jueces la pertinencia, relevancia, originalidad y rigor metodológico de una publicación.*

estructura del sistema a través de los diagramas de clases, secuencia, componentes, paquetes y caso de uso.

El presente documento se encuentra dividido en tres capítulos. En el primero de ellos, **Fundamentación Teórica**, se exponen los principales conceptos del tema (seguridad, riesgo, amenaza, vulnerabilidad), se analizan las principales vulnerabilidades de las aplicaciones web como son fallas de inyección, *cross-site scripting*, *buffer-overflow* y además se explican las principales características de las herramientas de análisis de código estático para el lenguaje JavaScript, tales como JSLint y JSHint. Adicionalmente se exponen la metodología, los lenguajes de programación y las herramientas que se usarán para realizar la propuesta de solución.

En el capítulo 2 denominado **Análisis y Diseño de la Solución** se realiza la extracción de requisitos funcionales y no funcionales del módulo a desarrollar, se modelan las reglas del negocio y la estructura y funcionamiento de la solución a través de los diagramas de clase y de secuencia. Además se seleccionan los patrones de diseño que serán usados para resolver la problemática.

En el capítulo 3 denominado **Implementación y pruebas de la solución** se realiza el análisis de las funciones identificadas como vulnerables, se muestra el diagrama de componentes así como las pruebas realizadas al sistema, los resultados de las mismas y la validación de la solución.

Capítulo 1: Fundamentación teórica

En este capítulo se abordan los elementos fundamentales para explicar por qué es necesaria la elaboración de la solución propuesta. Además se analizan las herramientas que realizan funciones similares a la propuesta por el autor y se expone la metodología de desarrollo, el lenguaje de programación y las herramientas elegidas para elaborar la aplicación.

1.1 Conceptos y definiciones fundamentales.

1.1.1 Ataque informático

Un **ataque informático** es un método por el cual un individuo, mediante un sistema informático, intenta tomar el control, desestabilizar o dañar otro sistema informático (ordenador, red privada, etcétera) (4).

Hay diversos tipos de ataques informáticos. Algunos son:

- Ataque de denegación de servicio también llamado *ataque DoS (Denial of Service)*: es un ataque a un sistema de computadoras o red que causa que un servicio o recurso sea inaccesible a los usuarios legítimos, normalmente provocando la pérdida de la conectividad de la red por el consumo del ancho de banda de la red de la víctima o sobrecarga de los recursos computacionales del sistema de la víctima (5).
- *Man in the middle* a veces abreviado MitM: es una situación donde un atacante supervisa (generalmente mediante un rastreador de puertos) una comunicación entre dos partes y falsifica los intercambios para hacerse pasar por una de ellas (6).

- Ataque de día cero: ataque realizado contra un ordenador, a partir del cual se explotan ciertas vulnerabilidades, o agujeros de seguridad de algún programa o programas antes de que se conozcan las mismas, o que, una vez publicada la existencia de la vulnerabilidad, se realice el ataque antes de la publicación del parche que la solvente (7).

1.1.2 Amenaza

Cualquier circunstancia o hecho que pueda afectar adversamente a un sistema a través del acceso no autorizado, destrucción, divulgación o modificación de datos, o la denegación de servicio (8). Las amenazas pueden clasificarse en dos tipos:

- Intencionales: en caso de que deliberadamente se intente producir un daño.
- No intencionales: se producen acciones u omisiones de acciones que si bien no buscan explotar una vulnerabilidad, ponen en riesgo los activos de información y pueden producir un daño.

Se considera **amenaza** a los fallos cometidos por los usuarios al utilizar el sistema, los fallos internos tanto del hardware como del software, los ataques por parte de personas, al igual que los desastres naturales que puedan afectar a su computadora.

1.1.3 Riesgo

Es la posibilidad de que una amenaza se produzca, dando lugar a un ataque al equipo. Es la vulnerabilidad ante un potencial perjuicio o daño para las unidades, personas, organizaciones o entidades. Cuanto mayor es la vulnerabilidad mayor es el riesgo (9).

1.1.4 Vulnerabilidad

Una **vulnerabilidad** es un fallo o debilidad en el diseño, la implementación, el funcionamiento o la gestión de un sistema, que puede ser explotado con la finalidad de violar la política de seguridad del sistema (8).

Asociado al término vulnerabilidad se encuentran los términos ataque y amenaza, definidos anteriormente y que se relacionan como se muestra en la Ilustración 1.

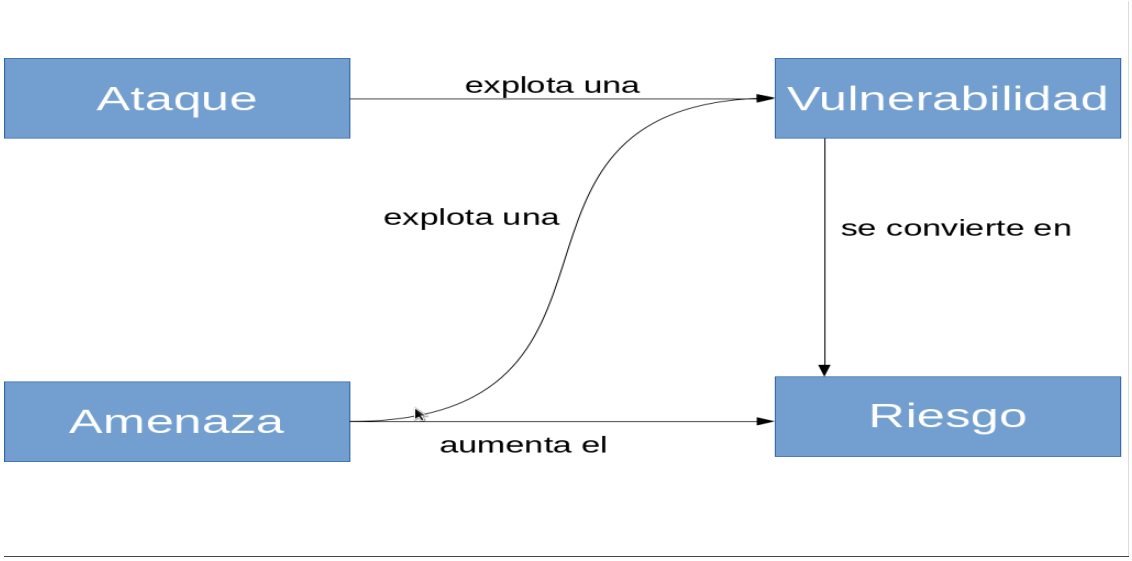


Ilustración 1: Relación entre Ataque, Amenaza, Vulnerabilidad y Riesgo.

1.2 Vulnerabilidades de las aplicaciones web

1.2.1 Fallas de Inyección

Las fallas de inyección ocurren cuando datos no confiables son enviados a un intérprete como parte de un comando o consulta. Los datos hostiles del atacante pueden engañar al intérprete

para ejecutar comandos no intencionados o acceder a datos no autorizados.

Las fallas de inyección son muy comunes, particularmente en código legado, el cual es frecuentemente encontrado en consultas SQL[6], LDAP[7], Xpath[8], comandos de Sistemas Operativos, argumentos de programa. Una falla de inyección puede resultar en pérdida o corrupción de datos, falta de integridad o negación de acceso. Una falla de inyección puede algunas veces llevar a la toma de posesión completa del servidor.

Ejemplos de escenarios de ataque:

La aplicación utiliza datos no confiables en la construcción de la siguiente consulta vulnerable SQL:

String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") +"' ' ";

El atacante modifica el parámetro ‘id’ en su navegador para enviar: ' or '1'='1.

Esto cambia el significado de la consulta devolviendo todos los registros de la tabla ACCOUNTS en lugar de solo el cliente solicitado.

http://example.com/app/accountView?id=' or '1'='1

En el peor caso, el atacante utiliza esta vulnerabilidad para invocar procedimientos almacenados especiales en la base de datos que permiten la toma de posesión de la base de datos y posiblemente también al servidor que aloja la misma.

---

6 *Es un lenguaje de programación utilizado para acceder a las bases de datos.*

7 *Protocolo Ligero/Simplificado de Acceso a Directorios es un protocolo a nivel de aplicación que permite el acceso a un servicio de directorio ordenado y distribuido para buscar diversa información en un entorno de red.*

8 *Es un lenguaje que permite construir expresiones que recorren y procesan un documento XML.*

1.2.2 *Cross-site scripting*(XSS)

XSS es la falla de seguridad más común en aplicaciones web. Las fallas XSS ocurren cuando una aplicación incluye datos suministrados por el usuario en una página enviada al navegador sin ser el contenido apropiadamente validado o escapado.

Existen tres tipos conocidos de fallas XSS:

1) Almacenados

2) Reflejados

3) XSS basado en DOM.

Los atacantes pueden ejecutar secuencias de comandos en el navegador de una víctima para secuestrar las sesiones de usuario, destruir sitios web, insertar código hostil, redirigir usuarios, instalar código malicioso en el navegador de la víctima.

Ejemplos de escenarios de ataque:

La aplicación utiliza datos no confiables en la construcción del siguiente código HTML sin validar o escapar los datos:

(String) page += "<input name='creditcard' type='TEXT‘ value='" request.getParameter("CC") + "'>";

El atacante modifica el parámetro ‘CC’ en el navegador:

'<script>document.location='http://www.attacker.com/cgi-bin/cookie.cgi?oo='+document.cookie</script>'.

Esto causa que el identificador de sesión de la víctima sea enviado al sitio web del atacante, permitiendo al atacante secuestrar la sesión actual del usuario. Notar que los atacantes pueden también utilizar XSS para anular cualquier defensa CSRF[9] que la aplicación pueda utilizar.

1.2.3 Pérdida de Autenticación y Gestión de Sesiones

Los desarrolladores a menudo crean esquemas propios de autenticación o gestión de las sesiones, pero conseguir que sean correctos es complicado. Por ello, a menudo estos esquemas propios contienen vulnerabilidades en las secciones de cierre de sesión, gestión de contraseñas, tiempo de desconexión, función de recordar contraseña, pregunta secreta, actualización de cuenta, etc. El atacante utiliza filtraciones o vulnerabilidades en las funciones de autenticación o gestión de las sesiones (por ejemplo cuentas expuestas, contraseñas, identificadores de sesión) para hacerse pasar por usuarios (10).

Estas vulnerabilidades podrían permitir que algunas o todas las cuentas sean atacadas. Una vez el ataque resulte satisfactorio, el atacante podría realizar cualquier acción que la víctima pudiese. Las cuentas privilegiadas son los objetivos prioritarios.

Ejemplos de escenarios de ataque:

Escenario #1: aplicación de reserva de vuelos que soporta re-escritura de direcciones URL poniendo los identificadores de sesión en la propia dirección:

[9] *Cross site request forgery: en este caso se refiere a defensas contra falsificación de petición en sitios cruzados*

http://example.com/sale/saleitems;jsessionid=2P0OC2JDPXM0OQSNDLPSKHCJUN2JV?dest=Hawaii

Un usuario autenticado en el sitio quiere mostrar la venta a sus amigos. Envía por correo electrónico el enlace anterior, sin ser consciente de que está proporcionando su identificador de sesión. Cuando sus amigos utilicen el anterior enlace utilizarán su sesión y su tarjeta de crédito.

Escenario #2: No se establecen correctamente los tiempos de desconexión en la aplicación. Un usuario utiliza un ordenador público para acceder al sitio. En lugar de utilizar la función de "Cerrar sesión", cierra la pestaña del navegador y se marcha. Un atacante utiliza el mismo navegador al cabo de una hora, y ese navegador todavía se encuentra autenticado.

1.2.4 Desbordamiento de buffer (*Buffer Overflow*)

Se produce cuando un programa no controla adecuadamente la cantidad de datos que se copian sobre un área de memoria reservada. Si dicha cantidad es superior a la capacidad pre-asignada, los bytes sobrantes se almacenan en zonas de memoria adyacentes, sobrescribiendo su contenido original, que probablemente pertenecían a datos o código almacenados en memoria (10).

Ejemplo de escenarios de ataque:

Un programa tiene definidos dos elementos de datos continuos en memoria: un *buffer* de 8 bytes tipo *string*, A, y otro de dos bytes tipo entero, B. Al comienzo, A contiene bytes nulos y B contiene el número 3 (cada carácter se representa mediante un byte).

0	0	0	0	0	0	0	0	0	3
Buffer A								Buffer B	

A continuación, el programa intenta almacenar la cadena de caracteres "demasiado" en el buffer A, seguido de bytes nulos para marcar el fin de string. Al no validarse la longitud de la cadena, se sobrescribe el valor de B:

'd'	'e'	'm'	'a'	's'	'i'	'a'	'd'	'o'	0
Buffer A								Buffer B	

A pesar de que el programador no quería cambiar el contenido del buffer B, el valor de éste ha sido reemplazado por un número equivalente a parte de la cadena de caracteres.

1.3 Vulnerabilidades del Lenguaje JavaScript

1.3.1 *DOM-based cross-site scripting* (XSS basado en DOM)

XSS basado en DOM[10] es un tipo de XSS que ocurre exclusivamente en el lado del cliente. Es más grave que los ataques XSS ordinarios, ya que tales secuencias de comandos no se pueden filtrar a través de un *firewall* de aplicaciones web.

Esta vulnerabilidad es el resultado del uso indebido del DOM con JavaScript, lo cual permite abrir otra página web con código malicioso JavaScript incrustado, afectando el código de la primera página en el sistema local. Cuando el XSS es local, ningún código malicioso es enviado al

[10] *Document Object Model: es la estructura de objetos que genera el navegador cuando se carga un documento y se puede alterar mediante JavaScript para cambiar dinámicamente los contenidos y aspecto de la página.*

servidor. El funcionamiento toma lugar completamente en la máquina del cliente, pero modifica la página proporcionada por el sitio web antes de que sea interpretada por el navegador para que se comporte como si se realizara la carga maliciosa en el cliente desde el servidor. Esto significa que la protección del lado del servidor que filtra el código malicioso no funciona en este tipo de vulnerabilidad. (10)

Ejemplo de DOM-based XSS:

Una página web *http://www.example.com/test.html* que contiene el código:

```
<script>
document.write("<b>Current URL<b> : " + document.baseURI);
</script>
```

Si envía una petición HTTP como este ***http://www.example.com/test.html# <script> de alerta (1) </ script>***, bastante simple, su código JavaScript será ejecutado, porque la página está escribiendo lo que está escrito en la dirección URL de la página con la función *document.write*. Si se observa en la fuente de la página, no se verá <***script> de alerta (1) </ script>***, ya que todo está sucediendo en el DOM y es hecho por el código JavaScript ejecutado.

Después de que el código malicioso es ejecutado por la página, sólo tiene que explotar esta vulnerabilidad de *DOM-based XSS* para robar las *cookies*[11] del usuario o cambiar el comportamiento de la página como desee.

En la siguiente tabla se realiza una comparación entre los ataques de tipo XSS clásicos y los ataques de XSS basado en DOM.

[11] *Pequeña información enviada por un sitio web y almacenada en el navegador del usuario, de manera que el sitio web puede consultar la actividad previa del usuario.*

Características del ataque	**XSS clásico**	**XSS basado en DOM**
Causa principal	Código fuente	Código fuente
Premisas	Incrustación inapropiada de los datos del cliente en las páginas HTML de salida (por el servidor)	Referenciación y el uso inadecuado en el código del lado del cliente, de los objetos DOM que no están totalmente controlados y verificados por las páginas HTML generadas por el servidor
ipo de página	Dinámica	Estática y dinámica
Detección	Sistemas de detección de intrusos, registros.	No puede ser detectado por el servidor si el atacante usa las técnicas apropiadas de evasión
Detección	Sistemas de detección de intrusos, registros.	No puede ser detectado por el servidor si el atacante usa las técnicas apropiadas de evasión
Detección de vulnerabilidades	Simulación de ataque; La revisión de código del lado del servidor; Herramientas de detección de vulnerabilidad que realizan	Simulación de ataque; La revisión de código del lado del cliente; Herramientas de detección de vulnerabilidad que realizan

	pruebas de penetración auto-mática	pruebas de penetración auto-mática
nsa	Sanitación del lado de servidor, los sistemas de prevención de intrusos.	Sanitación del lado de servidor, los sistemas de prevención de intrusos, en menor medida.

Tabla 1: Comparación entre XSS y XSS basado en DOM

1.4 Analizadores de seguridad de JavaScript

1.4.1 Herramienta JSLint

JSLint es un analizador *online* de código JavaScript creado por Douglas Crockford que nos permite mostrar puntos en lo que el código no cumpla unas determinadas reglas establecidas de "código limpio". Es un programa JavaScript que busca problemas en los programas JavaScript, es una herramienta de calidad del código. Toma una fuente JavaScript y lo analiza, Si encuentra un problema, devuelve un mensaje que lo describe y una ubicación aproximada del mismo dentro de la fuente.

JSLint mira algunas convenciones de estilo, así como problemas estructurales. Esto no prueba que un programa es correcto, simplemente proporciona otro par de ojos para ayudar a detectar problemas. No es una herramienta óptima, ya que es bastante exhaustiva y da muchos falsos positivos. Además tiene muchos detractores que alegan que los criterios evaluados son bastante subjetivos según el punto de vista de su creador (11).

1.4.2 Herramienta JavaScript Lint

Se basa en el motor JavaScript del navegador Firefox. Proporciona un marco sólido que no sólo puede comprobar la sintaxis de JavaScript, sino también examinar las técnicas de codificación utilizados en el *script* y advertir contra prácticas cuestionables. Puede comprobar todo el código fuente JavaScript para errores comunes sin tener que ejecutar el *script* o la apertura de la página web. Algunos errores que son detectados:

- Falta punto y coma al final de una línea.
- Las llaves sin un *if, for, while*, etc.
- El código que nunca se ejecuta debido a un *return, throw, continue, o break*.
- Declaraciones de casos en un *switch* que no tienen una sentencia *break*.
- Un cero al inicio que convierte un número en octal (base 8).
- Comentarios en los comentarios.
- La ambigüedad de si dos líneas adyacentes son parte de la misma declaración.
- Las declaraciones que no hacen nada.

Otros errores no tan comunes que también son detectados son:

- Las expresiones regulares que no están precedidos por un paréntesis, declaraciones o coma.
- Las declaraciones que están separados por comas en lugar de punto y coma.

- El uso de incremento (++) y decremento (--) a excepción de las declaraciones simples como "i ++;" o "--i;".
- La utilización del tipo *void*.
- Cadenas de signos sucesiva (por ejemplo x +++ y) o menos (por ejemplo x---y).

1.4.3 Herramienta JSHint

JSHint es una herramienta impulsada por la comunidad para detectar errores y problemas potenciales en el código JavaScript y hacer cumplir las convenciones de codificación de su equipo. Es muy flexible para que pueda ser fácilmente cambiada a sus directrices de codificación particulares y al ambiente en el cual se ejecutará el código. JSHint es de código abierto, JSHint fue creado y es mantenido por Anton Kovalyov, un programador de San Francisco.

Algunas de las opciones disponibles son:

Opción	**Descripción**
bitwise	Esta opción prohíbe el uso de operadores de bit a bit como ^ (XOR), | (OR), & (AND), etc.
curly	Esta opción requiere que siempre se utilicen llaves {} alrededor de los bloques en ciclos y condicionales.
eqeqeq	Esta opción prohíbe el uso de igualdad (==) y no igualdad (!=) para utilizar identidad === y no identidad (!==).
es3	Esta opción indica que el código se necesita adherir a la especificación de EC-

	MAScript 3. Útil si el código se va a ejecutar en navegadores antiguos – en especial Oldie – o en otro contexto de legado de JavaScript.
freeze	Esta opción prohíbe que se sobrescriban prototipos de objetos nativos como Array, Date, entre otros.
immed	Esta opción prohíbe el uso de IIFE sin envolverlas en paréntesis. El envolver la función en paréntesis ayuda a los lectores de código a entender que la expresión es el resultado de la función y no la función como tal.
latedef	Esta opción prohíbe el uso de una variable antes de que fuera definida.
quotmark	Esta opción refuerza la consistencia en las comillas utilizadas en el código. Acepta tres valores: *true* si no se quiere forzar un estilo en particular, *"single"* si únicamente se permiten comillas simples y *"double"* si únicamente se permiten comillas dobles.
undef	Esta opción prohíbe el uso explícito de variables no declaradas.

Tabla 2: Funcionalidades de JSHint

1.4.4 Herramienta Closure Compiler

Closure Compiler es un compilador de optimización de JavaScript. Analiza código fuente JavaScript, elimina el código muerto y reescribe y minimiza lo que queda. También comprueba la sintaxis, las referencias a variables y los tipos. Se utiliza en muchas de las aplicaciones de JavaScript de Google, incluyendo Gmail, Google Web Search, Google Maps y Google Docs. Es un verdadero compilador de JavaScript. En lugar de compilar desde un idioma fuente a código máquina, compila desde JavaScript para mejor JavaScript.

Entre los errores que detecta están:

- *JSC_BITWISE_OPERAND_OUT_OF_RANGE: operando fuera de rango. Este error significa que el valor del operando de la izquierda de una operación en modo bit no se ajusta a 32 bits.*
- *JSC_CONSTANT_REASSIGNED_VALUE_ERROR: Valor de una variable asignado dos veces. Este error significa que se está tratando una constante como una variable normal mediante la asignación de un valor a la misma más de una vez.*
- *JSC_DIVIDE_BY_0_ERROR: División por cero. Este error significa que hay una expresión división aritmética con un denominador de 0.*
- *JSC_DUPLICATE_EXTERN_INPUT:* Entrada externa duplicada. Este error significa que usted ha suministrado el mismo archivo externo más de una vez.

1.4.5 Resumen de análisis de las herramientas

	Modo de ejecución	Errores que detecta	XSS
JSLint	online	Errores de sintaxis y de estándares predefinidos de código	no
JSHint	consola	Errores de sintaxis y de estándares predefinidos de código	no
JavaScript Lint	Complemento en el navegador Firefox	Errores de sintaxis	no
Closure Compiler	online	Errores de sintaxis	no

Tabla 3: Resumen análisis de herramientas

Las herramientas analizadas están diseñadas para encontrar errores de sintaxis en códigos JavaScript, y no así para detectar vulnerabilidades que puedan conllevar a un XSS basado en DOM por lo que no son factibles para su utilización en la solución del problema.

1.5 Tecnologías a utilizar en el desarrollo de la solución

1.5.1 Metodologías de desarrollo de software

Una metodología de desarrollo de software es un conjunto de pasos y procedimientos que deben seguirse para desarrollar software. Una metodología está compuesta por varias etapas, las tareas que se realizan en las etapas, las restricciones que deben aplicarse, las técnicas y herramientas que se deben emplear y la forma de controlar y gestionar el proyecto (12). El uso de una metodología define qué hacer, cómo y cuándo durante el planeamiento, desarrollo y mantenimiento de un proyecto.

Open UP

Es un proceso modelo y extensible, dirigido a la gestión y desarrollo de proyectos de software basados en desarrollo iterativo, ágil e incremental apropiado para proyectos pequeños y de bajos recursos; y es aplicable a un conjunto amplio de plataformas y aplicaciones de desarrollo (13).

Fases de OpenUP

1. **Concepción**: Se basa en el entendimiento del propósito y objetivos y en obtener suficiente información para confirmar lo que el proyecto debe hacer. El objetivo de ésta fase es capturar las necesidades de los *stakeholder*[12] en los objetivos del ciclo de vida para el proyecto.

2. **Elaboración**: Se tratan los riesgos significativos para la arquitectura. El propósito de esta fase es establecer la base de la elaboración de la arquitectura del sistema.

3.**Construcción**: Esta fase está enfocada al diseño, implementación y prueba de las funcionalidades para desarrollar un sistema completo. El propósito de esta fase es completar el desarrollo del sistema basado en la arquitectura definida.

4. **Transición**: Asegurar que el sistema es entregado a los usuarios, y evalúa la funcionalidad y rendimiento del último entregable de la fase de construcción (14).

Se elige la metodología OpenUp porque la misma permite un desarrollo evolutivo para obtener retroalimentación y mejoramiento continuo, permite centrarse en la arquitectura de forma temprana para minimizar el riesgo y organizar el desarrollo. Además el proyecto ACF fue elaborado usando esta metodología por lo cual el autor considera que lo más correcto es continuar con el mismo estilo de desarrollo.

[12] *Son las personas que pueden afectar o son afectados por las actividades de una empresa*

1.5.2 Lenguaje de programación

Un lenguaje de programación es aquella estructura que, con una cierta base sintáctica y semántica, imparte distintas instrucciones a un programa de computadora (15). Los lenguajes de programación representan en forma simbólica y en manera de un texto los códigos que podrán ser leídos por una persona. Son independientes de las computadoras a utilizar. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana.

C++: Es un lenguaje imperativo orientado a objetos derivado de **C**. En realidad un súper conjunto de **C**, que nació para añadirle cualidades y características de las que carecía. El resultado es que como su ancestro, sigue muy ligado al hardware subyacente, manteniendo una considerable potencia para programación a bajo nivel, pero se la han añadido elementos que le permiten también un estilo de programación con alto nivel de abstracción. Dado que el mismo permite realizar programación estructurada y programación orientada a objetos es considerado un lenguaje de programación multiparadigma.

Se define el uso de este lenguaje para maximizar la compatibilidad entre el módulo que se desea realizar y el programa ACF, dado que el mismo está realizado en dicho lenguaje.

1.5.3 Herramientas de modelado

Las herramientas de modelado de sistemas informáticos, son herramientas que se emplean para la creación de modelos de sistemas que ya existen o que se desarrollarán. Permiten crear un "simulacro" del sistema, a bajo costo y riesgo mínimo (16).

Dentro de las herramientas de modelado podemos encontrar el grupo de herramientas

denominadas CASE (***C****omputer* ***A****ided* ***S****oftware* ***E****ngineering*, ingeniería asistida por computadora). Se pueden definir como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un Software (17).

Algunas de estas herramientas son: Microsoft Project, Rational Rose, JDeveloper, Magic Draw, Microsoft Visio, BoUML y Visual Paradigm.

Visual Paradigm

Visual Paradigm es una herramienta de modelado que ha sido concebida para soportar el ciclo de vida completo del proceso de desarrollo del software a través de la representación de todo tipo de diagramas. Se caracteriza por el uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación, capacidades de ingeniería directa e inversa, el modelo y código que permanece sincronizado en todo el ciclo de desarrollo, posee licencia gratuita, soporte de UML versión 2.0 y permite generar diagramas de Casos de Uso, Paquetes, Clases, etc. Soporta aplicaciones web, varios idiomas, permite la generación de código para Java y exportación como HTML, fácil de instalar y actualizar, compatibilidad entre ediciones y es posible integrarlo con las herramientas:

- Eclipse/IBM WebSphere.
- Jbuilder.
- NetBeans IDE.

Se utilizará la versión 8.0, y el lenguaje de modelado será UML 2.0.

1.5.4 Entorno de desarrollo Integrado (IDE)

Un IDE es una aplicación de software, que provee características comprensivas para facilitar al programador el desarrollo de software. Consta de un editor de código fuente, herramientas de construcción automática y *debugger*[13]. Algunos IDEs contienen un compilador, intérprete (computacional), o ambos, tales como NetBeans y Eclipse.

QtCreator

QtCreator es un IDE multiplataforma. Posee un editor de código con soporte para C++, QML[14] y *ECMAscript*[15], herramientas para la rápida navegación del código, resaltado de sintaxis y auto-completado, control estático de código y estilo a medida que se escribe. Permite la creación y diseño de *forms*[16] para proyectos C++ que son totalmente funcionales y pueden ser previsualizados inmediatamente para asegurarse de que se verá exactamente como se pensó. Se utilizará la versión 2.4.1.

1.5.5 Gestor de Documentación.

Un **generador de documentación** es una herramienta de programación que genera documentación destinada a los programadores (documentación de API) o a usuarios finales, o a ambos, a partir de un conjunto de código fuente especialmente documentado, y en algunos

13 *Es un programa usado para probar y eliminar los errores de otros programas (el programa "objetivo")*

14 *Qt Modeling Language: es un lenguaje declarativo basado en JavaScript para diseñar aplicaciones de interfaz de usuario.*

15 *Es el lenguaje de scripting estandarizado por Ecma Internacional en el ECMA-262.*

16 *Son los formularios de proyectos, las interfaces de usuarios*

casos, archivos binarios.

Doxygen

Doxygen es la herramienta estándar para la generación de documentación de fuentes en C++, pero también es compatible con otros lenguajes de programación populares como C, Objective-C, C#, PHP, Java, Python y otros.

Puede generar documentación en línea (en HTML) y/o un manual de referencia (LaTeX) a partir de un conjunto de archivos fuente documentados. También hay soporte para la generación de salida en formato RTF (MS-Word), PostScript, PDF con hipervínculos, HTML comprimido, y las páginas de manual Unix. La documentación se extrae directamente de las fuentes, lo que hace mucho más fácil mantener la documentación en conformidad con el código fuente.

1.6 Conclusiones del Capítulo 1

El análisis de las principales vulnerabilidades del lenguaje JavaScript es la base de la presente investigación. Para darle solución a las mismas se definieron un conjunto de elementos necesarios como son la metodología OpenUP, los lenguajes de programación C y C++, y las herramientas QtCreator y Visual Paradigm. Del análisis de las herramientas que analizan código fuente JavaScript se concluyó que las mismas no resuelven el problema planteado por lo que es necesario realizar un módulo el cual se capaz de, una vez integrado a la herramienta ACF, detectar las vulnerabilidades identificadas.

Capítulo 2: Análisis y diseño de la solución.

En el presente capítulo se exponen los requisitos del sistema, se realiza la modelación de los mismos en diagramas de casos de uso y su correspondiente descripción. Se elabora el diseño de la arquitectura del sistema y de su funcionamiento a través de los diagramas de clases, componentes y secuencia.

2.1 Propuesta de solución

Se implementará un módulo para la herramienta Auditoría de Código Fuente el cual será capaz de solucionar las vulnerabilidades planteadas en el capítulo anterior. Dicho módulo tendrá el nombre "*JavaScript*" y se utilizará para la realización del mismo, las herramientas y metodologías escogidas anteriormente.

2.2 Requisitos funcionales

Los requisitos funcionales son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. En algunos casos, los requisitos funcionales de los sistemas también pueden declarar explícitamente lo que el sistema no debe hacer (18).

Listado de requisitos:

RF1. Detectar vulnerabilidades en códigos fuente JavaScript.

RF1.1 Detectar vulnerabilidades XSS basada en DOM

RF1.2 Generar reporte de vulnerabilidades encontradas.

2.3 Requisitos no funcionales

Los requisitos no funcionales son restricciones de los servicios o funciones ofrecidos por el sistema. Incluyen restricciones de tiempo, sobre el proceso de desarrollo y estándares. Los requisitos no funcionales a menudo se aplican al sistema en su totalidad. Normalmente solo se aplican a características o servicios individuales del sistema (19).

Restricciones de diseño e implementación:

RNF1 Utilizar como lenguaje de programación C/C++.

RNF2 El estilo de programación debe ser el especificado en la Guía de estilo para lenguaje C/C++.

RNF3 Se utilizará Qtcreator como IDE, Visual Paradigm como herramienta de modelado, UML 2.0 como lenguaje de modelado y Doxigen como gestor de documentación.

Interfaz de usuario

RNF4 El módulo no contará con una interfaz de usuario dado que utilizara la de la propia herramienta ACF.

Restricciones de software

RNF5 El sistema operativo será GNU/Linux Nova 2013

2.4 Definición de actores y casos de uso

Un caso de uso es una secuencia de interacciones entre un sistema y alguien o algo que usa alguno de sus servicios, es una descripción de los pasos o las actividades que deberán realizarse para llevar a cabo algún proceso (20).

A continuación de describen los actores del sistema y las actividades que realizan los mismos.

Actores	Justificación
Usuario	Es la persona que revisa el código fuente de determinada aplicación en busca de vulnerabilidades.

2.4.1 Diagrama de casos de uso

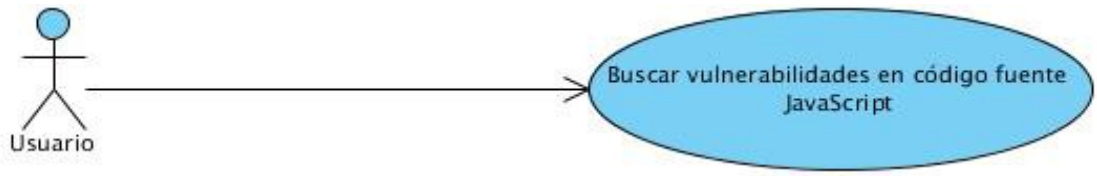


Ilustración 2: Diagrama de Casos de Uso

Caso de Uso # 1	**Buscar vulnerabilidades en el código fuente JavaScript.**
Actores	Usuario
Resumen	El caso de uso inicia cuando el usuario decide analizar un determinado paquete de aplicaciones, ficheros o archivos que contienen código JavaScript, y finaliza cuando el sistema muestra el reporte con las vulnerabilidades encontradas o si no posee ninguna
Precondiciones	El usuario desea analizar un paquete, fichero o directorio con archivos en JavaScript.
Referencias	R1
Complejidad	Alta
Prioridad	Alta
Poscondiciones	

Flujo Normal de Eventos		
#	**Acción del Actor**	**Respuesta del Sistema**
1	El usuario solicita detectar las vulnerabilidades de un fichero .js	2 Obtiene los ficheros que serán analizados.

		3. Busca vulnerabilidades que conlleven a un XSS basado en DOM, si no existen vulnerabilidades ir al flujo alterno 1
		4. Genera reporte de vulnerabilidades encontradas.

Flujo Alterno 1		
	Acción de los Actores	**Respuesta del Sistema**
		4. No se genera el reporte de vulnerabilidades

Tabla 4: Descripción de Caso de uso

2.5 Arquitectura de Software

La arquitectura de software se refiere a la estructuración del sistema. Representa un diseño de alto nivel del mismo que tiene dos propósitos primarios: satisfacer los atributos de calidad (desempeño, seguridad, modificabilidad) y servir como guía en el desarrollo (21). La herramienta ACF está desarrollada con una arquitectura n-capas. Dicha arquitectura permite organizar el trabajo por niveles y facilita el trabajo cuando se desea introducir algún cambio en el sistema porque solo se afecta el nivel que se desea modificar. La herramienta ACF se encuentra dividida en 3 capas o niveles:

Vista

La capa Vista es donde se crean las interfaces que serán visualizadas por el usuario, en la herramienta ACF se encuentra compuesto por el paquete ***Facade_Subsystem.*** Esta capa está conectada únicamente con la capa Lógica de Negocio.

Lógica de Negocio

La capa Lógica de Negocio es donde se crean las clases principales de la herramienta, son las clases que poseen los métodos para realizar las funciones de búsqueda y acceso a datos. Está compuesta por los paquetes ***Heart_Subsystem*** *y* ***Plugins_Subsystem***. Se relaciona con las capas Vista y Acceso a datos y funciona como intermediaria entre la interacción del usuario y el acceso a la Base de Datos.

Acceso a datos.

La capa Acceso a Datos es donde se almacenan los registros de búsqueda y los datos que son generados por las clases de la capa superior. Está compuesta por el paquete ***DBConexion_Subsystem.*** Solo se relaciona con la capa Lógica de Negocio**.**

2.5.1 Arquitectura basada en componentes

Para la elaboración de la solución se utiliza una arquitectura basada en componentes que esta presente en la capa Lógica de Negocio en el paquete ***Plugins_Subsystem***, más específicamente el subpaquete ***LanguajeSource***. Este paquete contiene los componentes de cada uno de los lenguajes de programación que la herramienta es capaz de analizar, por tanto el nuevo módulo será añadido en él.

La arquitectura basada en componentes es la combinación de partes que se integran como un todo para formar una solución. Como su nombre lo indica está conformada por componentes que son unas piezas de código preelaborado que encapsula alguna funcionalidad expuesta a través de interfaces estándar. Los componentes son los "ingredientes de las aplicaciones", que se juntan y combinan para llevar a cabo una tarea (22).

2.5.2 Patrones de Diseño GRASP

2.5.2 Patrones de Diseño GRASP

GRASP es un acrónimo que significa ***G****eneral* ***R****esponsibility* ***A****ssignment* ***S****oftware* ***P****atterns* (*patrones generales de software para asignación de responsabilidades*). Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en formas de patrones. Son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software (23).

Creador

Este patrón es el responsable de asignarle a una clase la responsabilidad de crear una instancia de otra. La nueva instancia deberá ser creada por la clase que:

- Tiene la información necesaria para realizar la creación del objeto.
- Usa directamente las instancias creadas del objeto.
- Almacena o maneja varias instancias de la clase.
- Contiene o agrega la clase.

En la solución planteada la clase ***JavaScript.cpp*** es la encargada de crear objetos de tipo ***Token*** y ***LexicalAnalyser*** las cuales son las encargadas de analizar el código en busca de vulnerabilidades.

Alta Cohesión

La cohesión es la medida en la que un componente se dedica a realizar solo la tarea para la cual fue creado, delegando las tareas complementarias a otros componentes. (Una clase debe de

hacer lo que respecta a su entidad, y no hacer acciones que involucren a otra clase o entidad) (24).

Es un patrón evaluativo que el desarrollador aplica al valorar sus decisiones de diseño. Una clase de alta cohesión posee un número relativamente pequeño, con una importante funcionalidad relacionada y poco trabajo que hacer. Colabora con otros objetos para compartir el esfuerzo si la tarea es grande (25).

Este patrón está presente en la clase ***JavaScript.cpp*** ya que la misma es la que contiene la declaración de las funciones vulnerables, pero quien realiza la búsqueda es la clase ***JSLexicalAnalyser.cpp*** la cual es la que contiene el autómata que revisa el código

En el siguiente fragmento de código extraído de la clase ***JavaScript.cpp*** podemos visualizar como se invocan los métodos de la clase ***JSLexicalAnalyser.cpp*** para realizar la obtención de tokens[17] y luego compararlos con las funciones definidas como vulnerables.

```
Token* token;
fileManager->append(detailsdir,
        "*----------------| %s | ----------------*\n", param.c_str());

while ((token = ((JSLexicalAnalyser*)tokenizer)->nextToken()) != NULL) {

    if (vulnerable_funct.count(token->getTK()) != 0) {

        if (token->getTK() == "document.URL") {
            vulnerabilities["document.URL"]++;
```

Ilustración 3: Fragmento del código ilustrando el patrón Alta Cohesión

[17] *Son las expresiones obtenidas luego de realizar el análisis sintáctico del código fuente analizado.*

Bajo Acoplamiento

El acoplamiento indica el grado de interdependencia entre las unidades de software (módulos, funciones, subrutinas, bibliotecas, etc.) de un sistema informático, es decir, el grado en que una unidad puede funcionar sin recurrir a otras; dos funciones son absolutamente independientes entre sí cuando una puede hacer su trabajo completamente sin recurrir a la otra (26).

Este patrón estimula asignar una responsabilidad de modo que su colocación no incremente el acoplamiento tanto que produzca los resultados negativos propios de un alto acoplamiento. Soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que acrecienten la oportunidad de una mayor productividad (25).

En este fragmento del diagrama de clases podemos apreciar que la clase ***JavaScript*** solo tiene relación directa con las clases ***Plugin*** y con la clase ***acf-javascript***, lo que evidencia que el nivel de acoplamiento es bajo.

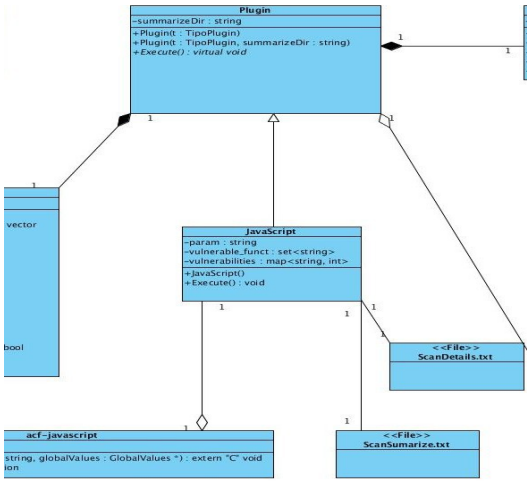


Ilustración 4: Fragmento del Diagrama de Clases ilustrando el patrón bajo acoplamiento

2.5.3 Diagrama de Paquetes

Un **diagrama de paquetes** muestra cómo un sistema está dividido en agrupaciones lógicas mostrando las dependencias entre esas agrupaciones. En el siguiente diagrama se muestran las relaciones entre las capas y los paquetes que contienen las mismas.

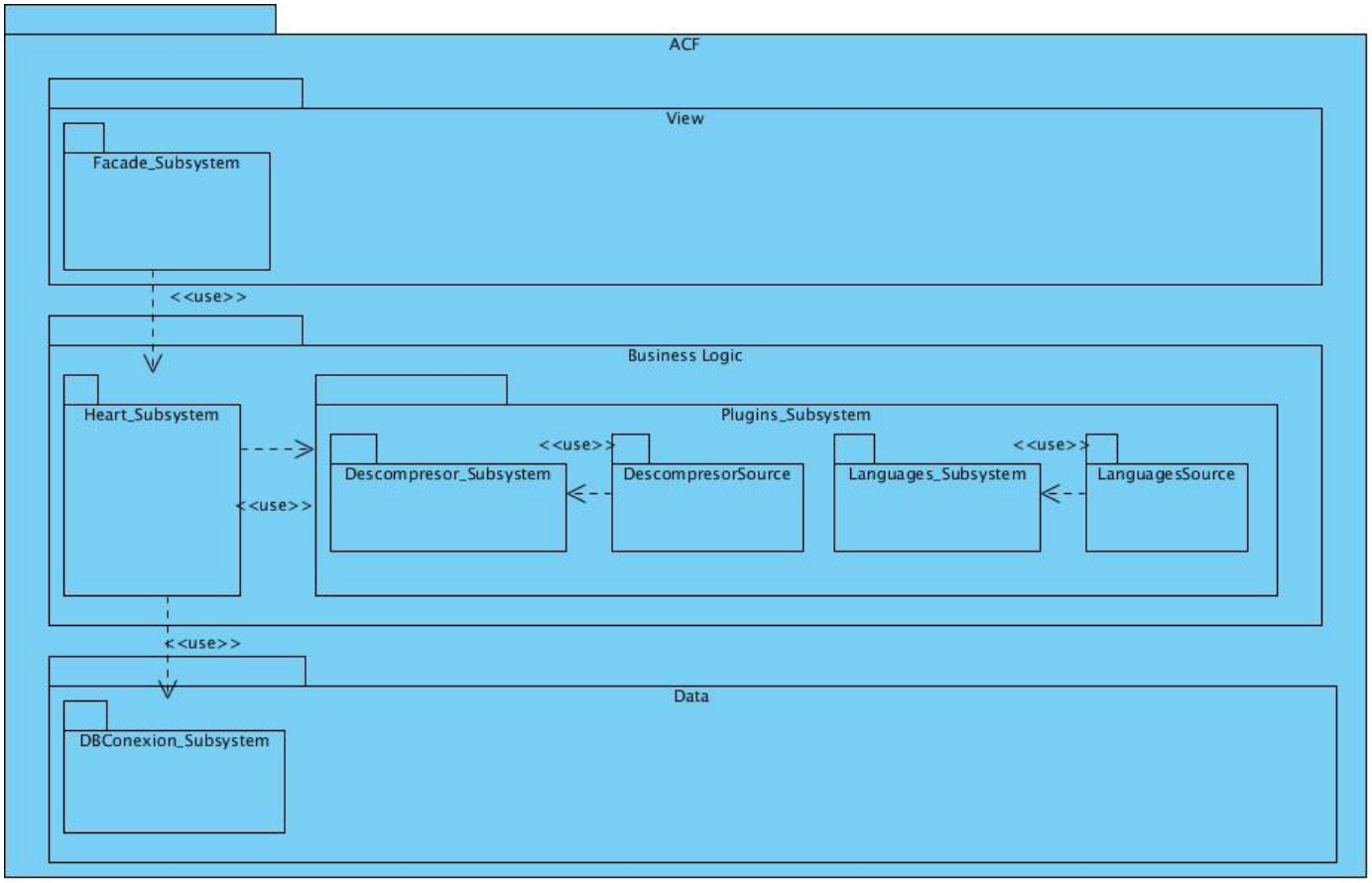


Ilustración 5: Diagrama de Paquetes

2.6 Diagrama de Clases

Los diagramas de clases muestran las diferentes clases que componen un sistema y cómo se relacionan unas con otras. Son diagramas estáticos porque muestran las clases, junto con sus métodos y atributos, así como las relaciones estáticas entre ellas: qué clases «conocen» a qué otras clases o qué clases «son parte» de otras clases, pero no muestran los métodos mediante los que se invocan entre ellas (27).

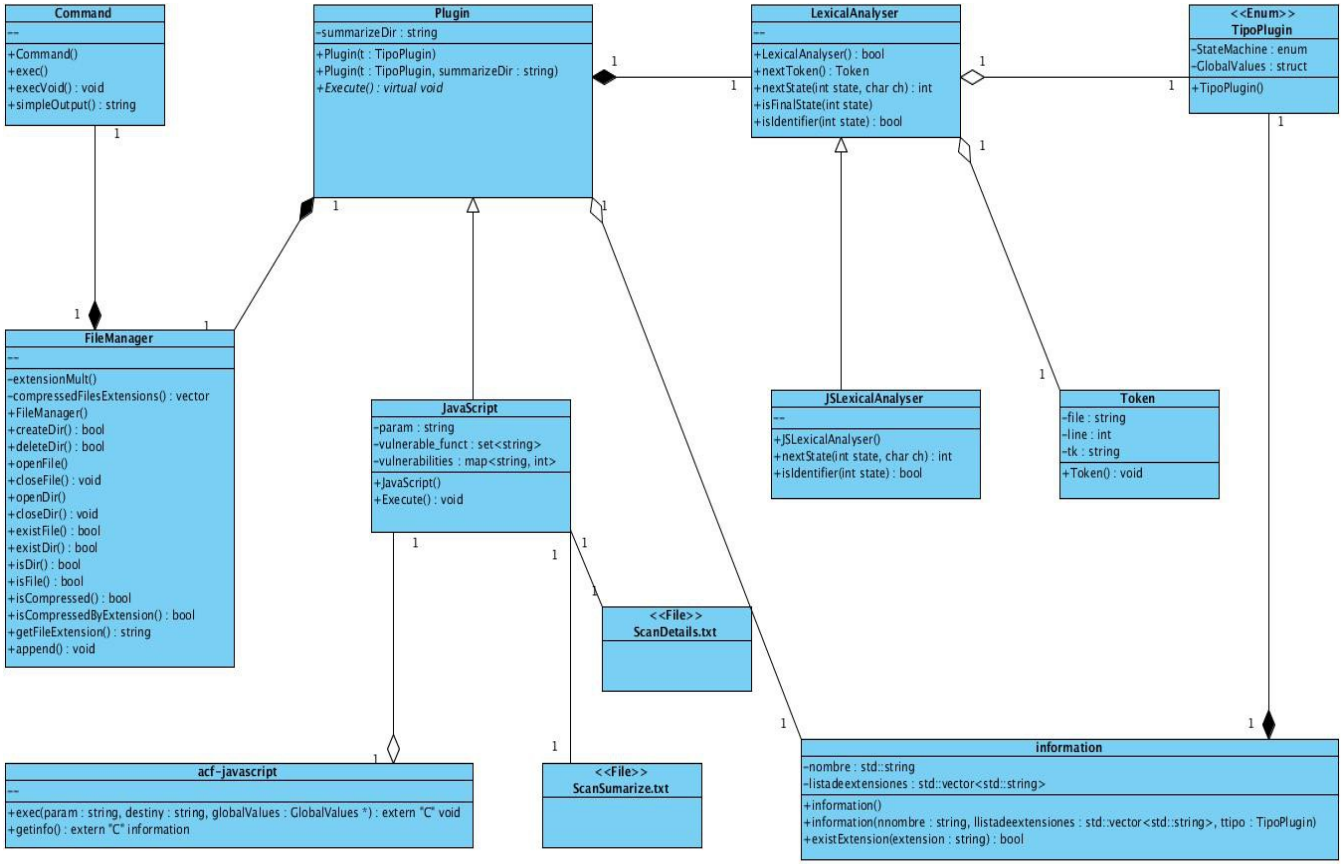


Ilustración 6: Diagrama de Clases

2.7 Diagrama de Secuencia

Un diagrama de secuencia muestra una interacción, que representa la secuencia de mensajes entre las instancias de clases, componentes, subsistemas o actores. Muestra instancias y eventos de ejemplo, en lugar de clases y métodos; pueden aparecer más de una instancia del mismo tipo y más de una aparición del mismo mensaje (28). En la siguiente imagen se muestra la interacción entre las clases del sistema y la funcionalidades añadidas.(Ver Diagrama de Secuencia)

2.8 Conclusiones del capítulo 2

Como parte del proceso de análisis y diseño de la solución propuesta se realizó la extracción de los requisitos funcionales y no funcionales del módulo a desarrollar. Se modeló, como parte de la metodología, el caso de uso correspondiente a la solución, los diagramas de clase y secuencia que permiten representar parte de la lógica del negocio y el modelo de dominio que permite representar las entidades relacionadas en la solución. Se hizo uso de los patrones de diseño GRASP, en específico los patrones Creador, Alta Cohesión y Bajo Acoplamiento.

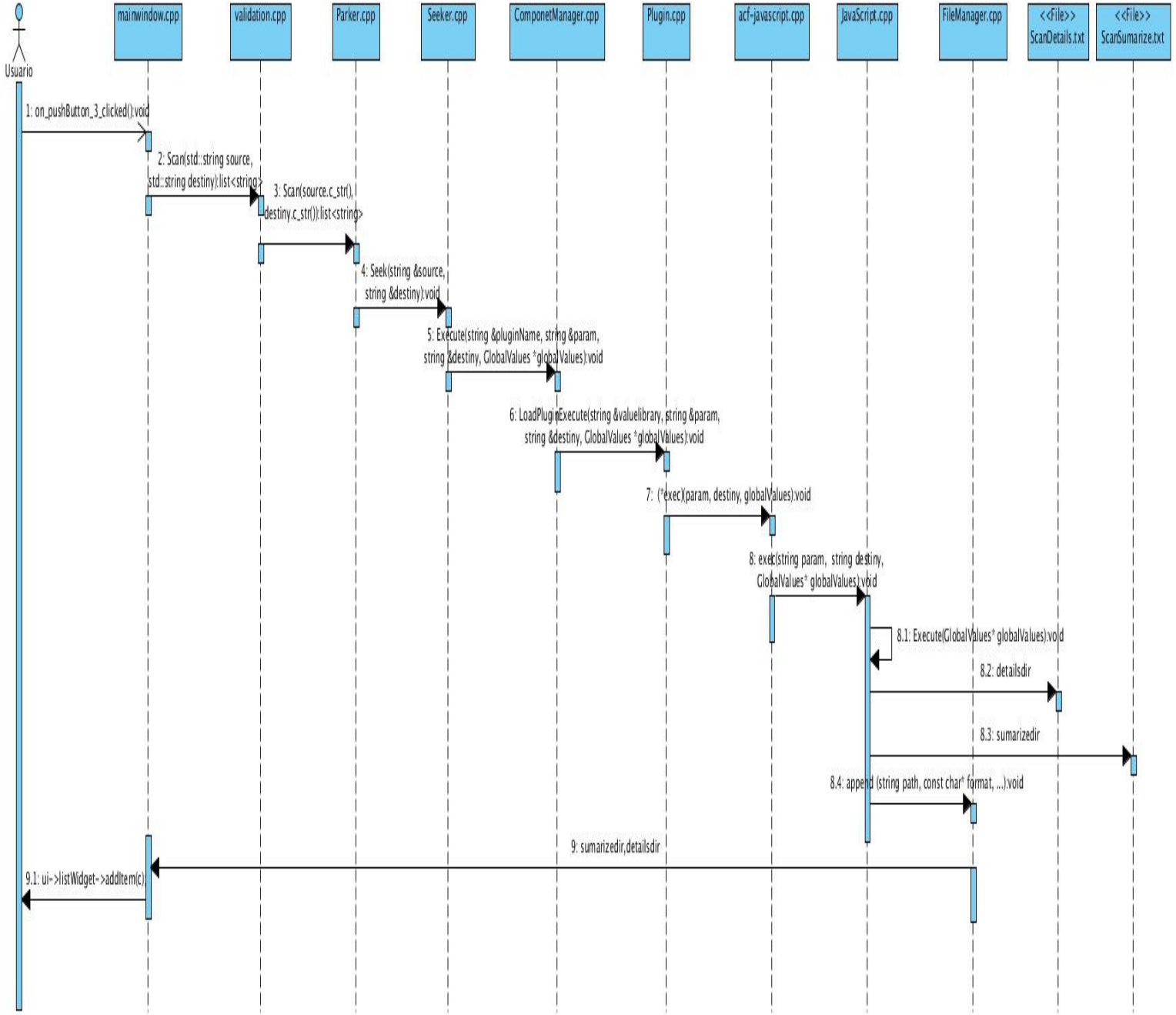


Ilustración 7: Diagrama de Secuencia

Capítulo 3: Implementación y pruebas de la solución.

En este capítulo se expone la estructura del módulo a través del diagrama de componentes, se realiza el análisis de las principales funciones que serán detectadas y ejecutan las pruebas y la validación de la solución

3.1 Diagrama de Componentes

Un componente es aquello que forma parte de la composición de un todo. Se trata de elementos que, a través de algún tipo de asociación o contigüidad, dan lugar a un conjunto uniforme (29).

Un diagrama de componentes representa cómo un sistema de software es dividido en componentes y muestra las dependencias entre estos. En el siguiente diagrama se muestran las relaciones entre las clases y cabeceras del sistema.

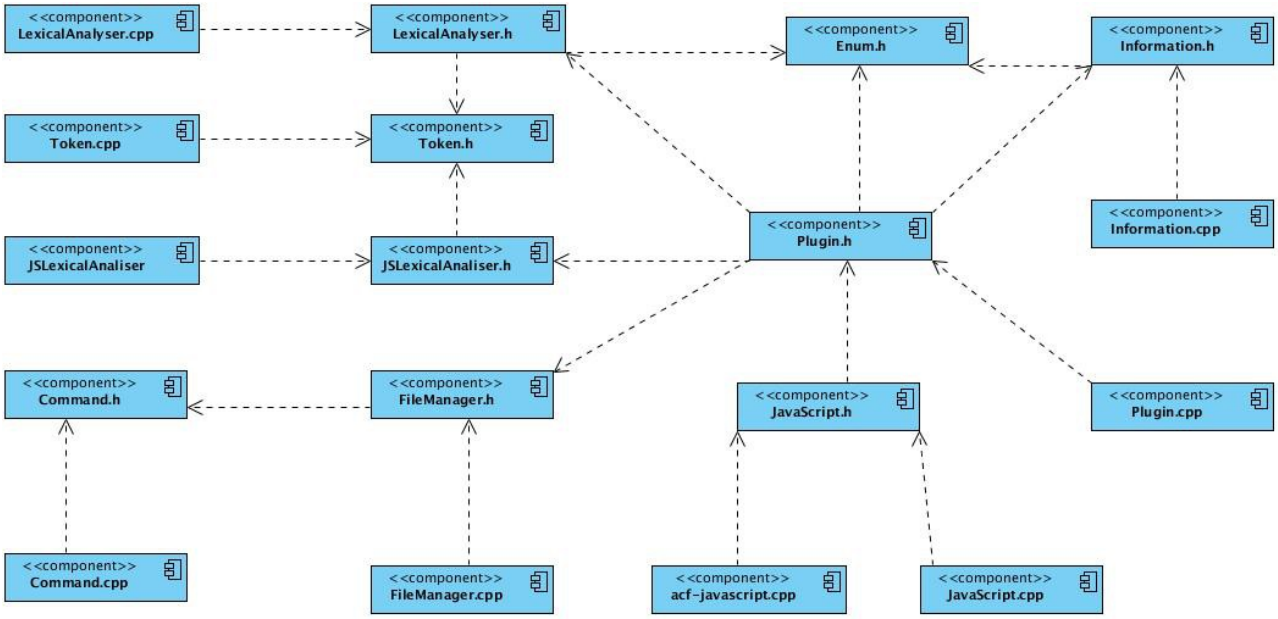


Ilustración 8: Diagrama de Componentes

3.2 Implementación de la solución

Funciones vulnerables detectadas

Funciones	Descripción	Tipo
document.URL	Esta función devuelve la URL actual del documento HTML.	Información
document.documentURI	Esta función devuelve la dirección del fichero como un string, en DOM3 es un atributo de lectura escritura y en DOM4 es de solo lectura	Información
location	Es una propiedad de sólo lectura que devuelve un objeto Localización, que contiene información acerca de la URL del documento y proporciona métodos para cambiarla y cargar otra URL.	Advertencia
location.hash	Establece o devuelve la parte de anclaje (#) de una URL	Advertencia
Location.href	Establece o devuelve toda la URL del documento.	Advertencia
location.search	Establece o devuelve la cadena de consulta de una URL.	Advertencia
document.referrer	Devuelve la URL del documento que ha cargado el documento actual.	Advertencia

document.write	El método write() escribe expresiones HTML o código JavaScript a un documento. Si se utiliza después de que un documento HTML se ha cargado completamente, se eliminará todo el HTML existente.	Alerta
document.writeIn	Este método es ídem a document.write(), con la inclusión de escribir un carácter de nueva línea después de cada declaración.	Alerta
setTimeout	Este método ejecuta una función o evalúa una expresión después de un tiempo determinado.	Alerta
eval	Esta función evalúa o ejecuta un parámetro	Alerta
setInterval	Esta función puede ejecutar o evaluar una expresión por intervalos de tiempo o continuar ejecutando una función hasta que se active la propiedad clearInterval() o se cierre la ventana.	Alerta

Tabla 5: Funciones vulnerables detectadas

Para definir el nivel de la vulnerabilidad se tuvieron en cuenta 2 criterios, nivel de uso y criticidad de la función. Ver Anexo 1.

Para su implementación estas funciones se definieron en un archivo xml el cual se muestra en el Anexo 2.

Ejemplos del uso de estas funciones que provocan XSS basado en DOM.

1- El siguiente segmento de código JavaScript lee un ID de empleado, **eid**, a partir de una solicitud HTTP y se lo muestra en pantalla al usuario.

```
<script>
var pos=document.URL.indexOf("eid=")+4;
document.write(document.URL.substring(pos,document.URL.length));
 </script>
```

El código de este ejemplo funciona correctamente si **"*eid*"** contiene solo texto alfanumérico estándar. Si **"*eid*"** tiene un valor que incluye caracteres meta o código fuente, el explorador web ejecutará el código al tiempo que muestra la respuesta HTTP. El peligro real es que un usuario creará la dirección URL malintencionada y después, utilizará trucos de correo electrónico o de ingeniería social para atraer a las víctimas para que visiten un vínculo a la dirección URL. Cuando las víctimas, hagan clic en el vínculo, sin darse cuenta reflejarán el contenido malintencionado a través de la aplicación web vulnerable en sus propios equipos

2- El siguiente ejemplo contiene este código en el lado del cliente:

```
<script>
document.write("Site is at: " + document.location.href + ".");
</script>
```

Un atacante puede anexar #<script> alert ('XSS') </script> a la URL de la página afectada que, al ejecutarse, mostraría el cuadro de alerta. En este caso, no sería enviado el código añadido al servidor, todo lo que venga después del carácter # no será tratado como un consulta del

navegador sino como un fragmento de la misma. En este ejemplo, el código se ejecuta de inmediato y una alerta de "XSS" es mostrada por la página.

3 - Uso incorrecto de la funcion eval().

```
var txtField = "field1";
var txtUserInput = "'test@csnc.ch';alert(1);";
eval("document.forms[0]." + txtField + ".value =" + txtUserInput);
```

La última comilla doble provoca que los datos introducidos por el usuario sean tratados como código JavaScript. Esto da como resultado el siguiente código JavaScript sea ejecutado por eval() provocando una brecha en el sistema:

```
document.forms[0].field1.value = 'test@csnc.ch';alert(1);
```

La forma correcta seria:

```
var txtField = "field1";
var txtUserInput = "'test@csnc.ch';alert(1);";
eval("document.forms[0]." + txtField + ".value = txtUserInput");
```

y lo que se ejecutaría por la función eval seria **document.forms[0].field1.value = txtUserInput.**

3.3 Pruebas

Las pruebas de software son las investigaciones empíricas y técnicas cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto a la parte interesada o *stakeholder*. Son un conjunto de actividades dentro del desarrollo de software.

3.3.1 Prueba de aceptación

Las pruebas de aceptación se elaboran a lo largo de la iteración, en paralelo con el desarrollo del sistema, y adaptándose a los cambios que el sistema sufra (30). El método de prueba seleccionado es el de caja negra el cual se centra en los requisitos funcionales del software. Dentro de este tipo de prueba se utilizó la técnica de partición de equivalencia.

El método de prueba de Caja Negra permite obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. En ellas se ignora la estructura de control, concentrándose en los requisitos funcionales del sistema y ejercitándolos. Esta estrategia de prueba se centra en la verificación de las funcionalidades de la aplicación: datos que entran, resultados que se obtienen, interacción con los actores, funcionamiento de la interfaz de usuario y en general todo aquello que suponga estudiar el correcto comportamiento que se espera del sistema.

Las pruebas de caja negra permiten encontrar:(30)

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a las bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y terminación.

Para confeccionar los casos de prueba de caja negra existen distintos criterios; algunos de ellos son: (30)

- Técnica de la Partición de Equivalencia: Divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.
- Técnica del Análisis de Valores Límites: Prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.
- Técnica de Grafos de Causa-Efecto: Permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

Para probar el sistema se utiliza la técnica de partición equivalente ya que la misma divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Un caso de prueba ideal descubre de forma inmediata una clase de errores, que de otro modo requerirían la ejecución de muchos casos antes de detectar el error genérico.

3.3.2 Caso de Prueba

Caso de uso buscar vulnerabilidades en el código fuente JavaScript.

Descripción general

El caso de uso inicia cuando el usuario decide analizar un determinado paquete de aplicaciones, ficheros o archivos que contienen código JavaScript, introduce las direcciones de los ficheros a analizar y del reporte.

Condición de Ejecución

Debe existir al menos un fichero que contenga código fuente JavaScript.

Escenario	Descripción	V1 Dirección	V2 Dirección de reporte	Respuesta del sistema	Flujo central
EC 1.1 Detecta vulnerabilidades en códigos fuente JavaScript	El usuario llena los campos de dirección y presiona el botón auditar	/ home/nova/ Documents/ extJs	/home/	El sistema muestra el reporte resumido con la cantidad de vulnerabilidades encontradas(Anexo 4) y genera los ficheros ScanSumarize.txt(Anexo 3) y ScanDetails.txt en la dirección indicada por el usuario.	1-El usuario introduce las direcciones de los ficheros a analizar y del reporte. 2-El usuario presiona el botón auditar. 3-El sistema muestra el reporte de vulnerabilidades.
EC 1.1 No existen vulnerabilidades en códigos fuente JavaScript	El usuario llena los campos de dirección y presiona el botón auditar	/ home/nova/ Documents/ extJs	/home/	El sistema muestra un mensaje con el texto "No se detectaron vulnerabilidades" y genera el fichero ScanDetails en la dirección indicada	1-El usuario introduce las direcciones de los ficheros a analizar y del reporte. 2-El usuario presiona el botón auditar.

				por el usuario	3-El sistema muestra un mensaje con el texto "No se detectaron vulnerabilidades"

Tabla 6: Caso de prueba

Descripción de las variables

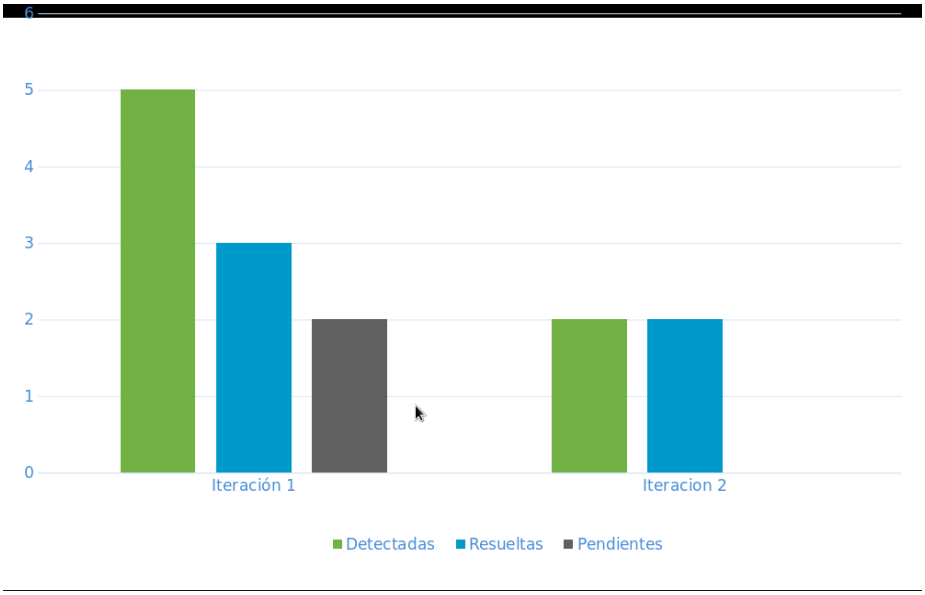
No	Nombre del Campo	Clasificación	Valor nulo	Descripción
1	Dirección	Campo de texto	no	Este campo admite una dirección dentro del sistema.
2	Dirección de reporte	Campo de texto	no	Este campo admite una dirección dentro del sistema.

Tabla 7: Descripción de las variables

3.3.3 Resultados de las pruebas

Durante el transcurso del período de prueba se detectaron cinco no conformidades, de estas tres fueron errores ortográficos que aparecían en el reporte generado por la herramienta y dos fueron

errores de implementación, se realizaron dos iteraciones de prueba y fueron resueltas todas las no conformidades.



3.3.4 Validación de la solución

Para realizar la validación de la aplicación se realizó el análisis de varios *frameworks* que cuentan con gran volumen de ficheros en JavaScript. Estos son *jquery, extjs y dojo***.** En la siguiente tabla se muestran los datos sobre la cantidad de vulnerabilidades esperadas y las vulnerabilidades identificadas por la aplicación.

	Vulnerabilidades esperadas	Vulnerabilidades detectadas	Por ciento(%)
ExtJs	33	33	100%
DoJo	30	30	100
Jquery-plugin	23	23	100%

Tabla 8: Validación de la solución

Estos datos demuestran la eficiencia y fortaleza de la herramienta para detectar las funciones definidas como vulnerables.

Conclusiones del capítulo 3

En este capítulo se elaboró el diagrama de componente del sistema, se analizaron las funciones vulnerables que son detectadas por la solución y su implementación en el código fuente de la misma. Además se realizaron las pruebas de aceptación utilizando el método de caja negra y la técnica partición equivalente. Para realizar dicha prueba se utilizó un caso de prueba basado en el caso de uso del capítulo anterior. Todas las no conformidades detectadas fueron solucionadas.

Conclusiones Generales

Al finalizar la presente investigación se concluye que las herramientas estudiadas para el análisis de código fuente JavaScript no permiten detectar las funciones vulnerables de dicho lenguaje, por lo que fue necesario utilizar la estructura de los componentes de los demás lenguajes dentro de la herramienta Auditoría de Código Fuente para lograr un componente que respondiera a la necesidad que es hoy en día incrementar la seguridad en todos los sistemas informáticos del país.

El módulo desarrollado permite la recolección de errores en el código JavaScript que se incluye en el repositorio de Nova, dando así cumplimiento al objetivo de la investigación y aun cuando este necesita ser perfeccionado y extendido, este trabajo comienza un camino que facilitará a Nova una herramienta que en los próximos años podría definir la integración de los principales desarrollos endógenos (Fiscalía, Elecciones, Cedrux) de Cuba con la distribución cubana de GNU/Linux puesto que esta tendría los elementos para potenciar la seguridad de estos sistemas.

Recomendaciones

Se recomienda:

- Realizar los módulos de los lenguajes restantes, tales como C#, Ruby, etc.
- Realizar una funcionalidad que permita a la herramienta ser capaz de revisar la integridad de los ficheros que analiza.

Bibliografía

1. CONWAY, Drew. dataists » Ranking the popularity of programming languages. [online]. [Accessed 19 March 2015]. Available from: http://www.dataists.com/2010/12/ranking-the-popularity-of-programming-langauges/

2. PERÉZ, Gaston, GARCÍA, G and N, Irma. *Metodología de la Investigación Educacional.* La Habana : Editorial Pueblo y Educación, 1996.

3. Centro de Recursos para la Escritura Académica del Tecnológico de Monterrey. [online]. [Accessed 23 April 2015]. Available from: http://sitios.ruv.itesm.mx/portales/crea/buscar/que/6_lospasos.htm

4. ¿Cuál es la definicion de Ataque informático? [online]. [Accessed 23 April 2015]. Available from: http://www.alegsa.com.ar/Dic/ataque%20informatico.php

5. ROUSE, Margaret. What is distributed denial-of-service attack (DDoS)? - Definition from WhatIs.com. [online]. [Accessed 11 February 2015]. Available from: http://searchsecurity.techtarget.com/definition/distributed-denial-of-service-attack

6. Ataque MitM. [online]. [Accessed 11 February 2015]. Available from: http://es.kioskea.net/contents/30-ataque-mitm

7. RODRÍGUEZ, Rafael Enrique. Integración de Plugin a un navegador web para análisis de vulnerabilidades en JavaScript. [online]. [Accessed 11 February 2015]. Available from: http://repository.ucatolica.edu.co/xmlui/handle/10983/1377

8. RFC 4949 - Internet Security Glossary, Version 2. [online]. [Accessed 3 March 2015]. Available from: https://tools.ietf.org/html/rfc4949

9. Seguridad Informática: ¿Qué es una vulnerabilidad, una amenaza y un riesgo? - Aprende a Programar - Codejobs. [online]. 9 July 2012. [Accessed 3 March 2015]. Available from: http://www.codejobs.biz/es/blog/2012/09/07/seguridad-informatica-que-es-una-vulnerabilidad-una-amenaza-%20y-un-riesgo#sthash.gpzyi0Rr.N4JB7Ggx.dpbs

10. OWASP_Top_10_-_2010_FINAL_(spanish).pdf. [online]. Available from: https://www.owasp.org/images/2/2d/OWASP_Top_10_-_2010_FINAL_%28spanish%29.pdf

11. JSLint y JSHint, analizadores de código javaScript online. [online]. [Accessed 15 April 2015]. Available from: http://www.genbetadev.com/herramientas/jslint-y-jshint-analizadores-de-codigo-javascript-online

12. Choosing An Appropriate System Development Methodology - SelectingDevelopmentApproach.pdf. [online]. [Accessed 19 February 2015]. Available from: https://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SelectingDevelopmentApproach.pdf

13. YANG, Michael. OpenUP. [online]. [Accessed 19 February 2015]. Available from: http://epf.eclipse.org/wikis/openup/

14. Gestión de Proyectos: OpenUP como alternativa metodológica para proyectos pequeños de software. [online]. [Accessed 19 February 2015]. Available from: http://kasyles.blogspot.com/2008/09/openup-como-alternativa-metodolgica.html

15. Definición de lenguaje de programación - Qué es, Significado y Concepto. [online]. [Accessed 19 February 2015]. Available from: http://definicion.de/lenguaje-de-programacion/

16. ALEGSA, Leandro. ¿Cuál es la definicion de Herramienta de modelado? [online]. [Accessed 19 February 2015]. Available from: http://www.alegsa.com.ar/Dic/herramienta%20de%20modelado.php

17. Herramientas CASE para el proceso de desarrollo de Software - Monografias.com. [online]. [Accessed 19 February 2015]. Available from: http://www.monografias.com/trabajos73/herramientas-case-proceso-desarrollo-software/herramientas-case-proceso-desarrollo-software.shtml

18. Requerimientos funcionales y no funcionales. [online]. [Accessed 16 April 2015]. Available from: http://www.scribd.com/doc/37187866/Requerimientos-funcionales-y-no-funcionales#scribd

19. Requerimientos Funcionales y No Funcionales (RF/RNF). [online]. [Accessed 16 April 2015]. Available from: http://ingenieriadesoftware.bligoo.com.mx/requerimientos-funcionales-y-no-funcionales-rf-rnf

20. CERIA, Santiago. *Ingeniería de Software I,Casos de Uso Un Método Práctico para Explorar Requerimientos* [online]. Buenos Aires, Argentina : Universidad de Buenos Aires, [no date]. Available from: http://www-2.dc.uba.ar/materias/isoft1/2001_2/apuntes/CasosDeUso.pdf

21. Arquitectura de Software | SG. [online]. [Accessed 30 April 2015]. Available from: http://sg.com.mx/revista/27/arquitectura-software

22. Desarrollo de Software basado en Componentes. [online]. [Accessed 30 April 2015]. Available from: https://msdn.microsoft.com/es-es/library/bb972268.aspx

23. Patrones GRASP | Prácticas de Software. [online]. [Accessed 18 April 2015]. Available from: http://www.practicasdesoftware.com.ar/2011/03/patrones-grasp/

24. Alta cohesión y bajo Acoplamiento – Diseño de Software | El Blog de Julio Pari. [online]. [Accessed 18 April 2015]. Available from: http://blog.juliopari.com/alta-cohesion-y-bajo-acoplamiento-diseno-de-software/

25. Patrones Grasp | El Mundo Informático. [online]. [Accessed 18 April 2015]. Available from: https://jorgesaavedra.wordpress.com/category/patrones-grasp/

26. ALEGSA, Leandro. ¿Cuál es la definición de Acoplamiento (informática)? [online]. [Accessed 18 April 2015]. Available from: http://www.alegsa.com.ar/Dic/acoplamiento.php

27. Elementos de UML. [online]. [Accessed 18 April 2015]. Available from: https://docs.kde.org/stable/es/kdesdk/umbrello/uml-elements.html

28. Diagramas de secuencia UML: Referencia. [online]. [Accessed 18 April 2015]. Available from: https://msdn.microsoft.com/es-es/library/dd409377.aspx

29. Definición de componentes - Qué es, Significado y Concepto. [online]. [Accessed 11 May 2015]. Available from: http://definicion.de/componentes/

30. PRESSMAN, ROGER S. *Ingenieria de Software Un Enfoque Práctico*. [no date].

Anexo 1

Definición del nivel de las vulnerabilidades detectadas

Criterios:

Nivel de uso: El nivel de uso está determinado por la frecuencia en que es usada una función en un código. Para determinar un por ciento se realizó el análisis de diferentes conjuntos de ficheros JavaScript con el objetivo de establecer un promedio. Se definieron 3 categorías: Alto, Medio y Bajo

Criticidad: La criticidad está determinada por el efecto que podría provocar el uso incorrecto de una determinada función para el sistema. Se definieron 3 categorías: Alta, Media y Baja

Matriz para evaluar el nivel de las vulnerabilidades

Nivel de uso/C**riticidad**	Baja	Media	Alta
Bajo	Baja	Baja	Media
Medio	Baja	Media	Alta
Alto	Media	Alta	Alta

Nivel de Vulnerabilidades

Baja = Información

Media = Advertencia

Alta = Alerta

Funciones	Nivel de uso	Criticidad
document.URL	Medio	Baja
document.documentURI	Medio	Baja
location	Medio	Media
location.hash	Medio	Media
Location.href	Alto	Media
location.search	Medio	Media
document.referrer	Bajo	Alta
document.write	Alto	Alta
document.writeIn	Medio	Alta
setTimeout	Alto	Alta
eval	Alto	Alta
setInterval	Alto	Alta

Anexo 2


```
- <contenedor>
  - <vulnerabilidad>
      <nombre>document.URL</nombre>
      <tipo>Information</tipo>
    - <mensaje>
        Función vulnerable 'document.URL' Esta propiedad permite obtener la dirección URL del documento , verifique que se está usando correctamente
      </mensaje>
    </vulnerabilidad>
  - <vulnerabilidad>
      <nombre>document.documentURI</nombre>
      <tipo>Information</tipo>
    - <mensaje>
        Función vulnerable 'document.documentURI' Esta propiedad permite obtener la dirección URL del documento , verifique que se está usando correctamente
      </mensaje>
    </vulnerabilidad>
  - <vulnerabilidad>
      <nombre>document.location</nombre>
      <tipo>Warning</tipo>
    - <mensaje>
        Función vulnerable 'location' Esta propiedad almacena informacion sobre la URL verifique que se está usando correctamente
      </mensaje>
    </vulnerabilidad>
  - <vulnerabilidad>
      <nombre>location.href</nombre>
```

Anexo 3


ScanDetails.txt
204 *----------------| analizado | ----------------*
205
206 *----------------| /home/nova/Descargas/ext-4.2.0.663/src/data/Field.js | ----------------*
207 -> ADVERTENCIAS = 0
208 -> INFORMACIONES = 0
209 *----------------| analizado | ----------------*
210
211 *----------------| /home/nova/Descargas/ext-4.2.0.663/src/data/Connection.js | ----------------*
212 ADVERTENCIA [line: 341] Función vulnerable 'setTimeout' Este método puede ser usado para ejecutar funciones por periodos de tiempo, verifique que se está ejecutando correctamente
213 ADVERTENCIA [line: 544] Función vulnerable 'setTimeout' Este método puede ser usado para ejecutar funciones por periodos de tiempo, verifique que se está ejecutando correctamente
214 -> ADVERTENCIAS = 2
215 -> INFORMACIONES = 0
216 *----------------| analizado | ----------------*
217
218 *----------------| /home/nova/Descargas/ext-4.2.0.663/src/data/Connection.js | ----------------*
219 -> ADVERTENCIAS = 0
220 -> INFORMACIONES = 0
221 *----------------| analizado | ----------------*
222

Anexo 4

Resultado de la auditoría

```
**********Archivos que deben ser analizados**********

/home/nova/Descargas/ext-4.2.0.663/src/data/Connection.js
->  ADVERTENCIAS = 2
-> INFORMACIONES = 0

/home/nova/Descargas/ext-4.2.0.663/src/data/JsonP.js
->  ADVERTENCIAS = 1
-> INFORMACIONES = 0

/home/nova/Descargas/ext-4.2.0.663/src/Template.js
->  ADVERTENCIAS = 1
-> INFORMACIONES = 0

/home/nova/Descargas/ext-4.2.0.663/src/Ext.js
->  ADVERTENCIAS = 1
-> INFORMACIONES = 0

/home/nova/Descargas/ext-4.2.0.663/src/XTemplateCompiler.js
->  ADVERTENCIAS = 1
-> INFORMACIONES = 0
```