Detección de inyección de código malicioso en páginas web bancarias*[1]

Detection of malicious code injection in banking websites

Detecção de injeção de código malicioso em sites bancários

Johnathan Steven SALAMANCA LANCHEROS[2], Sandra RUEDA RODRÍGUEZ[3]

Recibo: 17.09.2018 – Aprobación: 23.03.2019

DOI: https://doi.org/10.30554/ventanainform.39.3310.2018

Resumen: *Las transacciones bancarias realizadas por medio de aplicaciones web son muy comunes hoy día. Sin embargo, diversos adversarios han desarrollado métodos para atacar el navegador y obtener información sensible del usuario. Un método de ataque es la inyección de código que consiste en alterar el contenido de una página web para pedir a un usuario su información bancaria y luego enviarla vía internet a un adversario. Este tipo de ataque es difícil de detectar porque algunas aplicaciones legítimas presentan el mismo comportamiento: generan contenido dinámico e inyectan código. Esta propuesta presenta un mecanismo para detectar inyección de código fraudulento, del lado del cliente, en páginas web bancarias con base en las direcciones URL presentes en los componentes de una página web. Dado que los servidores web*

* **Modelo para la citación de este artículo / Template for citation of this article / Modelo para a citação deste artigo:** SALAMANCA LANCHEROS, Johnathan Steven & RUEDA RODRÍGUEZ, Sandra (2018). Detección de inyección de código malicioso en páginas web bancarias. En: Ventana Informática No. 39 (jul-dic). Manizales (Colombia): Facultad de Ciencias e Ingeniería, Universidad de Manizales. p. 35-52. ISSN: 0123-9678. DOI: https://doi.org/10.30554/ventanainform.39.3310.2018

1 Artículo de investigación / Research article / Artigo de pesquisa
Proyecto / Project / Projeto: Detección de fraude bancario en navegadores por medio de análisis dinámico / Detection of banking fraud in Browsers through dynamic analysis / Detecção de fraude bancária em navegadores por meio de análise dinâmica.
Periodo / Period / Período: 01/2016 - 12/2016
Institución / Institution / Instituição: Universidad de los Andes (Bogotá, Colombia)

2 Ingeniero de Sistemas y Computación / Computer and systems engineer. / Engenheiro de Sistemas e Computação. Analista de Ciencias Cuantitativas / Analyst of Quantitative Sciences / Analista de Ciências Quantitativas, Banco AV Villas (Bogotá, Colombia). js.salamanca1967@uniandes.edu.co.

3 Ph.D. en Ciencias de la Computación e ingeniería / Ph.D. in Computer Science and Engineering / Ph.D. em Ciência da Computação e Engenharia. Profesora Asistente / Assistant profesor / Professor asistente, Universidad de los Andes (Bogotá, Colombia). sarueda@uniandes.edu.co. https://orcid.org/0000-0002-2111-9348

construyen las páginas web de los bancos a partir de archivos fuente almacenados en un conjunto preestablecido de fuentes, y solo envían datos a un conjunto preestablecido de destinos, es posible usar listas blancas para clasificar las URL. La propuesta es simple, fácil de mantener y la evaluación muestra que es efectiva para detectar extensiones maliciosas instaladas localmente.

Palabras clave: *Aplicaciones Web Bancarias, Ataques al Navegador, Análisis Dinámico.*

Abstract: *Online banking, via web applications, is very common nowadays. However, various adversaries have developed methods to attack browsers and leak sensitive data from users. Code injection is one of these methods, it modifies a web page, on the fly, to ask users for their data and leak it through internet. This kind of attack may be difficult to detect as legitimate applications have the same behavior: they dynamically generate content and inject code. This proposal presents a mechanism to detect malicious code injections on the client side, to banking websites, based on URLs the pages include. Since web servers build web pages of banks with source files from a predefined set of sources and only send data to a predefined set of targets, it is possible to use white lists to classify URLs. The proposal is simple, easy to manage and effective to detect malicious extensions locally installed.*

Keywords*: Online Banking Applications, Man in the Browser, Dynamic Analysis.*

Resumo: *As operações bancárias feitas através de aplicações web são muito comuns hoje em dia. No entanto, vários adversários desenvolveram métodos para atacar o navegador e obter informações confidenciais do usuário. Um método de ataque é a injeção de código que consiste em alterar o conteúdo de uma página Web para solicitar a um usuário suas informações bancárias e enviá-lo via internet a um adversário. Esse tipo de ataque é difícil de detectar porque alguns aplicativos legítimos têm o mesmo comportamento: eles geram conteúdo dinâmico e injetam código. Esta proposta apresenta um mecanismo para detectar a injeção fraudulenta de código, do lado do cliente, em páginas bancárias da Web com base nas URL presentes nos componentes de uma página da web. Como os servidores da Web criam as páginas Web dos bancos a partir de arquivos de origem armazenados em um conjunto predefinido de fontes e enviam somente dados para um conjunto predefinido de destinos, é possível usar listas de permissões para classificar os URL. A*

proposta é simples, fácil de manter e a avaliação mostra que é eficaz detectando extensões maliciosas instaladas localmente.

Palavras-chave*: Aplicativos da Web para bancos, Ataques do navegador, Análise dinâmica.*

Introducción

Hoy, los bancos ofrecen a sus clientes plataformas web para facilitar la realización de sus transacciones bancarias. Sin embargo, diversos actores maliciosos han desarrollado ataques, como *Phishing* y *Spear Phishing,* según señalan Williams, Hinds & Joinson (2018, 1-2), para obtener las credenciales de los usuarios en estas plataformas y así realizar transacciones en su nombre.

Para proteger a sus clientes, los bancos invierten recursos en entrenamiento básico para los usuarios, aseguramiento de las plataformas que soportan sus servicios, y procedimientos y tecnologías para prevenir problemas de seguridad. Un aspecto difícil de abordar en este contexto es el aseguramiento de los navegadores que los clientes usan para conectarse a las aplicaciones web al estar bajo el control del usuario. Una de las técnicas de ataque, que involucra un navegador, es la inyección de código malicioso mientras se carga una página web.

Los sitios web vulnerables facilitan la inyección de código[4] que después afecta a los usuarios. Symantec (2016, 9 y 2017, 11) reportó que 77% de los sitios web escaneados en 2013 presentaban vulnerabilidades, este porcentaje cambió a 76 (2014), 78 (2015) y 76 (2016), mientras las vulnerabilidades críticas fueron de 16%, 20%, 15% y 9% en 2013, 2014, 2015 y 2016, respectivamente. Un usuario también puede ser engañado para instalar *malware*[5] que inyecta código *on the fly*.

Uno de los problemas para detectar inyecciones maliciosas es que algunas aplicaciones legítimas también usan la inyección para ayudar al usuario. Por ejemplo, según Chrome Web Store (2018), la extensión o *plugin* de *Web of Trust* (*WOT*) para navegadores, realiza cambios a

4 Aunque no todas las vulnerabilidades conducen necesariamente a inyección de código en el navegador, sí contribuyen a que ataques variados sean posibles. Incluso si un sitio web bancario no presenta vulnerabilidades, un usuario puede cargar en su navegador, de forma simultánea, un sitio web vulnerable y el sitio web de su entidad bancaria, creando un ambiente apropiado para un ataque.

5 Symantec (2016, 30), presenta un caso de 2014 que muestra su potencial dañino: los cibercriminales instalaron malware en la página web de miles de usuarios de *Boleto*, una forma de pago electrónico en Brasil, y así comprometieron transacciones por US$4 billones. Al pagar con un boleto, un cliente debe imprimirlo y consignar el dinero en un banco. El malware instalado interceptaba el boleto enviado al cliente y cambiaba la información de la cuenta bancaria para la consignación, desviando así el dinero sin que el usuario lo notara.

los resultados de una búsqueda para indicar la confiabilidad de cada sitio en la lista en forma de semáforos: verde para buena reputación, rojo para mala, y amarillo como advertencia de prudencia.

Este artículo propone la construcción de un mecanismo que identifique las inyecciones de código malicioso en páginas web bancarias, del lado del cliente, con base en la clasificación de las direcciones URL que contiene una página, ya que los sitios web bancarios, a diferencia de la mayoría de servicios, usan un conjunto preestablecido de servidores para construir sus páginas web y para enviar información. Se organiza así: primero se exponen los fundamentos teóricos usados en la solución planteada y en trabajos previos. Luego, se plantea diseño e implementación de la solución propuesta, así como la evaluación de la solución en diferentes escenarios. Finalmente, se presentan las conclusiones.

1. Fundamento teórico

1.1 Pedido y adulteración de páginas Web

Las aplicaciones web, incluidas las aplicaciones web bancarias, implementan un esquema de comunicación cliente-servidor que usa el protocolo HTTP (*HyperText Transfer Protocol*). De acuerdo con Kurose & Ross (2012, 99), en este esquema hay dos procesos: un cliente (el navegador que el usuario ejecuta) y un servidor (el sitio web que responde a la solicitud del cliente), ejecutándose en nodos diferentes y comunicándose por medio de mensajes http que la red transmite. Cuando un usuario escribe una dirección web en su navegador, éste envía una petición HTTP[6] al servidor web para pedir la página que debe desplegar. El procedimiento se repite cada vez que el usuario pide más páginas u objetos web.

Las inyecciones de código web son mecanismos utilizados para alterar el contenido de una página web y obtener información confidencial del usuario. Esta manipulación es posible por medio de aplicaciones, instaladas subrepticiamente en el navegador, que mediante llamados a las interfaces de programación (API: application programming interface)

6 Un mensaje HTTP para pedir objetos tiene tres partes: - la petición (indica el tipo de pedido y la versión del protocolo), el encabezado (indica los parámetros para la transacción) y el cuerpo. Por ejemplo, si un mensaje http contiene un objeto de tipo video se usa el encabezado *Content-Type: video*, el cuerpo tiene el contenido del mensaje, por ejemplo, datos de un formulario, señalan Kurose & Ross (2012, 105). Como el protocolo HTTP no garantiza confidencialidad, ni integridad de los mensajes, se le adicionó *Secure Sockets Layer*, SSL, dando como resultado el protocolo http seguro o https, que ofrece confidencialidad, integridad y autenticación de origen y destino.

disponibles interceptan el contenido enviado por un servidor y lo alteran antes de ser desplegado. Estos ataques se conocen como *man-in-the-browser* (MIB), como informan Boutin (2014, 25) y Continella et al. (2017, 117). Los *plugins* o extensiones de los navegadores son mecanismos usados por los atacantes para instalar aplicaciones maliciosas, ya sea engañando al usuario para que instale la extensión o aprovechando una vulnerabilidad en un navegador para generar la instalación automática cuando un usuario carga una página web comprometida.

De acuerdo con Boutin (2014, 26), una inyección de código usualmente depende de un archivo de configuración, copiado en el computador del usuario, que contiene la página objetivo de la inyección (*set_url*), el código a insertar (*data_inject*) y las líneas de código entre las que se insertará (*data_before y data_after*). El servidor encargado de gestionar el archivo es denominado *servidor de Comando y Control* (C&C). El uso de un archivo de configuración da flexibilidad al ataque, dado que puede ser modificado fácilmente desde el C&C, pues el código que lo compone es relativamente simple y facilita las inyecciones.

Como los archivos de configuración tienen información útil para los investigadores en seguridad, sus creadores procuran que el contenido sea difícil de obtener, usando diferentes técnicas de cifrado, como lo señalan Cabrera & Larco (2018, 15). Otra técnica para dificultar la detección de una inyección de código, indica Boutin (2014, 29), es la ofuscación de código, donde se cambia código *Javascript* simple, como una asignación de variables, por un conjunto de operaciones matemáticas y lógicas con el mismo efecto del código original.

1.2 Métodos genéricos para detección de Malware

Los expertos en seguridad, según Castellanos et al. (2016, 15), usan técnicas que pueden ser clasificadas en dos categorías principales: técnicas de análisis estático y técnicas de análisis dinámico, para entender las aplicaciones maliciosas y cómo trabajan. Para detectar código malicioso, aseguran Shim et al. (2018, 2), se tienen métodos estáticos y dinámicos, que evalúan un programa con base en patrones o firmas: mientras en el análisis estático un patrón se define con base en instrucciones sospechosas y flujos de datos o de control, sin ejecutar el programa, en el análisis dinámico[7] un patrón se define con base en comportamientos sospechosos a tiempo de ejecución.

7 Una limitación de este método es que solo tiene en cuenta el camino que efectivamente ejecuta, no considera todos los posibles caminos que un programa puede seguir en diferentes ejecuciones.

Damodaran et al. (2017, 3) realizan una comparación entre los dos métodos: - El análisis estático busca analizar un programa sin ejecutarlo. El análisis examina el código de un programa y construye un modelo que representa los caminos que puede seguir durante su ejecución, por ejemplo, grafos de flujos de control, de flujos de datos y de llamados a funciones, y - El análisis dinámico examina las instrucciones que efectivamente se ejecutan y los resultados de las mismas. Entre la información que puede ser considerada en este método se incluyen llamadas a diferentes API, llamadas al sistema, cambios en archivos y cambios en variables de configuración.

Amro (2018, 6), indica que un programa será clasificado como *malware* si alguno de sus patrones es consistente con un patrón malicioso reconocido. Sin embargo, los dos tipos de métodos de análisis, estáticos y dinámicos, pueden generar falsos positivos y falsos negativos, al fallar en el reconocimiento de malware que no ha sido previamente detectado; como no se conocen las instrucciones o comportamientos de estos programas, es difícil marcar sus instrucciones y comportamientos como sospechosos.

1.3 Antecedentes

Provos et al. (2007) afirman que el uso de plataformas web para soportar transacciones puede exponer la información confidencial y comprometer los dispositivos, siendo usualmente el eslabón más débil el nodo donde el usuario ejecuta sus programas[8]. Además, señalan que para comprometer un sitio web, se insertan pequeños fragmentos de HTML en sus páginas web causando la instalación automática de *malware* que le puede dar al atacante control remoto sobre el sistema infectado para obtener información confidencial del usuario. Se denomina *drive-by-download* al procedimiento por el cual se aprovechan las vulnerabilidades de un sitio web para insertar código que permita a un navegador descargar e instalar automáticamente software malicioso. Hay cuatro formas para comprometer una página web con inyecciones de código:

- Servidor web. La seguridad del contenido que entrega un servidor web depende de la seguridad de la plataforma subyacente. Si algún componente de la plataforma presenta fallas de seguridad, señalan

8 A diferencia de los servidores web que manejan la mayoría de servicios comerciales, los computadores y dispositivos personales corren gran número de aplicaciones que no son administradas ni actualizadas, dado que los usuarios no tienen entrenamiento como administradores de sistemas, facilitando la acción atacante de encontrar y aprovechar una vulnerabilidad en un navegador web desde el acceso de un usuario.

Provos el al. (2008, 2), un adversario podría insertar código malicioso en los archivos fuente usados para generar los documentos HTML.

- Contenido aportado por usuarios. Algunos sitios web, como blogs y manejadores de perfiles y foros, permiten a los visitantes aportar su propio contenido. Estos sitios pueden presentar problemas cuando no filtran de forma apropiada el contenido que los visitantes adicionan porque ellos pueden insertar código HTML que es posteriormente interpretado, por los navegadores de otros usuarios, como contenido de la página.
- Publicidad. Algunos sitios generan ingresos por medio de anuncios publicitarios, para ello insertan en sus propias páginas web pequeñas piezas de JavaScript proporcionadas por las compañías anunciantes. Estas compañías por lo general son de fiar, dado que dependen de la confianza de sus clientes para mantenerse en el negocio. Sin embargo, algunas obtienen sus contenidos de otros proveedores lo cual puede resultar en inyección de código malicioso.
- *Widgets* de terceros. Un *widget* es un enlace, que un administrador de un sitio web inserta en el código fuente de sus propias páginas, para adicionar alguna funcionalidad, como un contador de visitas. Si el proveedor del widget está comprometido, este mecanismo puede habilitar inserciones de código malicioso.

Entre los problemas de contenido aportado por usuarios, *Cross Site Scripting* (XSS[9]: ejecución de comandos de sitios cruzados), aprovecha la confianza de un usuario en un sitio web para pasar información confidencial a otro sitio, por medio de código *Javascript*[10] o HTML inyectado en la página. A pesar de ser un ataque ampliamente conocido se presenta de forma frecuente, al punto que OWASP (2017, 13) lo sitúa la segunda vulnerabilidad en ocurrencia. Para OWASP (2018), este ataque puede presentarse de forma almacenada o reflejada: - en la primera, el código inyectado es almacenado en el servicio objetivo, como un foro de mensajes, y un usuario carga el código malicioso, sin saberlo, cuando solicita la información almacenada, mientras – en la segunda, los ataques llegan al usuario por medio de un enlace, cuando el usuario selecciona el enlace,

9 «*Tiene que ver mucho con inyección basada en HTML 5, que normalmente se da en navegadores comunes centrándose especialmente en el cliente y no en el servidor. La mayoría de los últimos teléfonos inteligentes funcionan con versiones de su sistema operativo basado en HTML 5, por desgracia las aplicaciones basadas en HTML 5 también son susceptibles de ataques de cross-site scripting como la mayoría de las aplicaciones web*» (Herrera, 2017, 34).

10 Zhang et al. (2018, 1193) encontraron que la inyección de código, vía JavaScripts, es común en el despliegue de páginas web en contextos móviles, en particular, contenido y direcciones web son los recursos más manipulados.

el código inyectado viaja al sitio web vulnerable, el cual refleja el ataque en el navegador del usuario que termina ejecutando el código creyendo que viene de un sitio confiable.

Otra vulnerabilidad de los navegadores es *Cross-Origin Javascript capability leaks,* la cual permite cambiar los permisos de un *script* que se está ejecutando (como transacciones bancarias), con la identidad del usuario. Ella, informan Barth, Weinberger & Song (2009), se basa en una falla en la arquitectura de los navegadores Web: el DOM (*Document Object Model*) y el motor de *Javascript* evalúan de forma diferente que sus objetos pertenezcan al mismo origen, pudiendo afectar todos los sitios web, así estén blindados contra XSS.

Ante la incidencia de vulnerabilidades, Liu et al. (2016, 2) expresan que se han propuesto salvaguardas como segunda línea de defensa (ej: SOMA, DCS y CSP), para mitigarlas. Sin embargo, a continuación se presentan algunos mecanismos:

- Política del mismo origen (SOP – *Same Origin Policy*) que, según Van Acker, Hausknecht & Sabelfeld (2018), proporciona una política de referencia para un origen completo, prohibiendo, de manera predeterminada, el acceso desde un elemento de un origen sobre elementos que provienen de otros orígenes.
- Aprobación mutua del mismo origen (SOMA – *Same Origin Mutual Approbation*) es una mejora a SOP, al definir una política de seguridad para garantizar la aprobación del contenido incluido en páginas web, con un conjunto de reglas que controlan el flujo de información y previenen la inclusión de código no autorizado (Inyección de código, *drive-by-download*, fuga de información, etc.). Sobre ella, Liu et al. (2016, 12) señalan que cuando el navegador solicita el recurso del servidor en la lista blanca (navegadores habilitados), encontrando un sí o un no, con lo cual decide permitir la solicitud.
- Continella et al. (2017) combinan dos técnicas para detectar inyecciones web de código: - análisis diferencial, que analiza el despliegue de la página web en diferentes máquinas buscando diferencias, pues debería verse igual, a menos que ocurran algunas inyecciones de código (algunas legítimas pueden variar dependiendo del contexto, lo que lleva a la generación de falsos positivos), por lo que se complementa con filtros como listas blancas, y - análisis de memoria, buscando en memoria los archivos de configuración de las inyecciones.

2. Metodología

Este trabajo aborda el problema de controlar las inyecciones de código malicioso en el contexto de un usuario que carga en su navegador una aplicación web bancaria. Para manejarlo se plantea el diseño y construcción de una extensión al navegador que verifique la validez de la información utilizada para construir una página web y del destino de la información entregada por un usuario, e informe al usuario si encuentra contenido sospechoso.

Para determinar si existe una potencial fuga de información, la extensión evaluará el código HTML de una página, después de la ejecución de todos los *scripts* de creación, dado que el código malicioso puede ser ofuscado, y determinará el origen de los diferentes componentes que conforman la página web y el destino de los enlaces y formularios de la misma.

Los sitios web bancarios, a diferencia de otros, incluyen un número reducido de fuentes y destinos de información, que permite establecer con precisión listas cortas con los nombres de los dominios considerados como fuentes y destinos confiables, considerando que las validaciones se hacen con base en la técnica de lista blanca (solo los dominios explícitamente mencionados son confiables).

A diferencia de SOP y SOMA, esta propuesta no realiza el análisis antes del despliegue de una página web, sino que espera a que el navegador ejecute las modificaciones correspondientes a la inyección de código. Así, no es necesario evaluar el código fuente de los *scripts* para determinar si son potencialmente maliciosos, sino directamente el código HTML. Además, no solo evalúa la fuente de la información, sino también revisa el destino de la información.

2.1 Diseño

La extensión propuesta tiene dos componentes que correrán en una arquitectura cliente-servidor: - el cliente es el componente integrado al navegador web, se encarga del análisis del código HTML, mediante la extracción de los componentes de interés, para obtener origen (*src*) de los componentes que forman parte de la página y el destino al que irá la información (*href* en los enlaces y *action* en los formularios) de cada etiqueta HTML en la página web, y - el servidor es externo y se encarga de examinar la información extraída por el cliente y determinar la posibilidad de fuga de información. Se realiza un análisis de lista blanca para validación de origen y destino, donde solo serán válidos

aquellos destinos explícitamente indicados en la lista, marcando como contenido sospechoso si una fuente o destino no esté en la lista.

2.2 Implementación

2.2.1 Herramientas. El cliente se implementó como una extensión de *Google Chrome* que recolecta la información de interés de una página web. El servidor se implementó como una aplicación web en *Play! Framework* y un prototipo del servidor se encuentra desplegado en *Heroku* (https://tesisis.herokuapp.com/). El servidor implementa el protocolo HTTPS, para garantizar un canal de comunicación protegido entre la el cliente y el servidor. La Figura 1 muestra el diagrama de despliegue de los nodos de la extensión y sus relaciones.

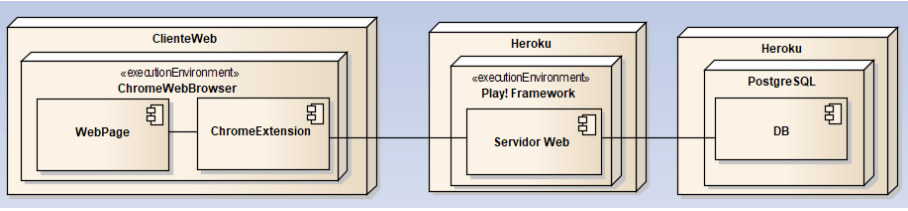

Figura 1. *Diagrama de despliegue de la solución propuesta.*

2.2.2 Análisis de las páginas. Cuando un usuario pide una URL particular, el navegador pide el contenido correspondiente al servidor web que la aloja. El contenido puede incluir *scripts* que corren en el navegador y actualizan el contenido desplegado, y al terminar la ejecución de los *scripts* de la página, el cliente podrá ver el código HTML generado después de las transformaciones, incluyendo posibles inyecciones de código, tanto benignas como maliciosas.

La función de la extensión implementada encuentra todas las referencias externas en el código HTML de la página web, utilizando *jQuery*[11]. A continuación, se construye un mensaje *JSON* con la información de los atributos de interés, que incluye la URL de origen, identifica la página web de la cual se extrajo la información y el conjunto de URL de destino, especificando el tipo de etiqueta (*tag*), la dirección (*link*) y la acción o atributo en la cual se encontró (*action*).

La extensión envía el mensaje *JSON* al clasificador, una aplicación web externa, que determina si la página tiene componentes que permitan la fuga de información, o no. El clasificador maneja dos conjuntos de reglas

11 Librería de *Javascript* que encuentra en un documento HTML los atributos (*src*, *href* y *action*) con un nombre particular.

por dominio web: - generales, que definen aquellos sitios web confiables para el público en general (por ejemplo, www.googletagmanager.com), y – específicas, referidas a URL confiables únicamente para la página que está siendo analizada (por ejemplo, una URL http://www.mibanco.com estaría autorizada para el dominio *mibanco*, pero no lo estaría para una página web de otro banco). El clasificador revisa las URL por el cliente y las compara con las listas general y específica de cada banco. Si alguna de las URL no está en las listas, envía un mensaje al cliente web indicando el hallazgo de un enlace sospechoso que puede ocasionar fuga de información.

El clasificador responde la solicitud notificando al cliente el resultado. La extensión del navegador recibe la respuesta del servidor y despliega un mensaje de alerta si la respuesta indica que hay enlaces sospechosos, de lo contrario no despliega ningún mensaje. La Figura 2 resume los pasos.

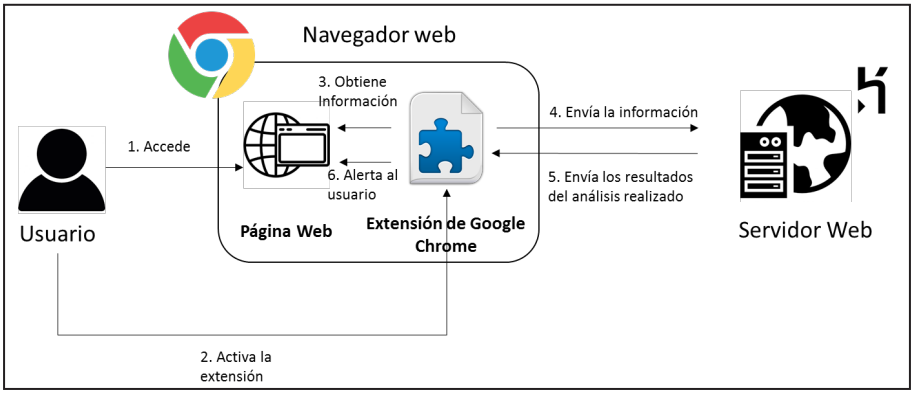

Figura 2. *Diagrama de flujo de información de la solución implementada.*

2.2.3 Código. El código fuente de la propuesta se puede encontrar en el repositorio de GitHub (https://github.com/ComitUniandes/DILB).

2.2.4 Instalación del Cliente. Un usuario solo necesita instalar la extensión de Google Chrome en su navegador y ejecutarla. La extensión se encargará de recolectar y enviar los datos al servidor y presentar el resultado.

2.2.5 Instalación del Servidor. El clasificador[12] debe correr en un servidor web. Para su configuración se considera que el archivo con

12 Aplicación construida en el *Framework Play* para el desarrollo de aplicaciones web con Java. Este *Framework* utiliza un patrón de arquitectura MVC que separa los datos (Modelo) de la interfaz (Vista) y la lógica de la aplicación

la lista blanca se encuentra en la carpeta *./public/data* bajo el nombre *WhiteList.csv* y tiene la estructura presentada en la Tabla 1.

Tabla 1. Estructura del archivo de lista blanca

Lista	Fuente
General	https://www.facebook.com
General	https://www.twitter.com
General	https://twitter.com
www.grupobancolombia.com	https://asistencia.webv2allus.com.co
www.grupobancolombia.com	https://sucursalpersonas.transaccionesbancolombia.com
www.grupobancolombia.com	.grupobancolombia.com
www.grupobancolombia.com	www.sufi.com.co
www.grupobancolombia.com	www.localizacolombia.com
www.grupobancolombia.com	http://www.banistmo.com
www.grupobancolombia.com	http://www.bancoagricola.com
www.grupobancolombia.com	http://www.bam.com.gt
www.davivienda.com	http://davivienda.custhelp.com/
www.davivienda.com	davivienda.com
www.davivienda.com	https://play.google.com/

Si un administrador desea agregar nuevas reglas generales o específicas, solo debe actualizar el contenido del archivo de configuración en la ruta */cargar* (i.e. tesisis.herokuapp.com/cargar), y así la aplicación lee los contenidos del archivo y agrega a la base de datos los nuevos registros de la lista blanca.

3. Resultados y discusión

3.1 Descripción de resultados

Esta sección describe las pruebas realizadas para evaluar la propuesta, las cuales *no evalúan la seguridad de los servidores*, *solo evalúan el comportamiento de la extensión instalada al navegador del cliente* (en todos los casos se ejecutó el navegador con la extensión propuesta). Los casos se describen a continuación:

- Caso 1: Página web sin inyección de código. Al realizar el análisis sobre la página web sin realizar inyecciones de código no se generan alertas teniendo en cuenta que todos los contenidos originales de la página se encuentran autorizados.

(Controlador) favoreciendo el desacoplamiento de los diferentes componentes y la actualización de una aplicación.

- Caso 2: Página web con inyección de código benigno. Se desarrolló el *script* que permite la inserción de código benigno, con un dominio registrado en la lista blanca:

```
var head = document.getElementsByTagName('head')[0];
var script = document.createElement('script');
script.src = "https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js";
head.appendChild(script);
```

No se despliega ningún mensaje de alerta al usuario pues el código insertado se encuentra en la lista blanca del servidor.

- Caso 3: Página web con inyección de código malicioso. El *script* desarrollado para inyectar código no autorizado (proviene de sitios no autorizados), añade un formulario a la página web, apuntando a una URL no autorizada:

```
var body = document.getElementsByTagName('body')[0];
var form = document.createElement('form');
form.action = "https://www.urlmaligna.ru/";
var input = document.createElement('input');
var button = document.createElement('button');
button.type = "submit";
form.appendChild(input);
form.appendChild(button);
body.appendChild(form);
```

El *script* genera el código HTML que se insertó un formulario en el código que compone la página y al final de esta se encuentra el campo de texto (*input*) en la esquina inferior izquierda. Al terminar la ejecución de los *scripts*, la extensión recoge la información de interés y genera el mensaje de alerta. La Figura 3 muestra los resultados de los casos 2 y 3.

3.2 Discusión de resultados

Esta propuesta tiene las siguientes ventajas:

- Analiza el documento HTML después de la ejecución de los diferentes *scripts* de inyección de código. A diferencia de los esquemas de análisis estático, no es necesario construir modelos complejos para predecir el comportamiento de un fragmento de código y así determinar si puede generar una inyección o no, teniendo como consecuencia que la propuesta sea simple y fácil de entender.

Figura 3. *Resultados de inyección de código benigno y malicioso.*

- Al analizar los enlaces externos en el código HTML resultante de la inyección de código, en vez del código inyectado, se logra enfocar el análisis en la detección de fuga de información hacia destinos no autorizados.

- La arquitectura cliente-servidor propuesta permite desacoplar los componentes, de tal forma que si un administrador desea realizar cambios en las reglas para la detección de factores de fuga de información solamente es necesario cambiar la configuración del servidor, y el usuario no tendrá que hacer nada.

Sin embargo, se han identificado algunas limitaciones de la solución implementada y que deberán ser tenidas en cuenta para extender la aplicación a futuro:

- La aplicación no reconoce código *Javascript* inyectado que aproveche las características DOM del documento HTML y extraiga información sin necesidad de insertar componentes HTML adicionales. Esto requeriría analizar a fondo el código inyectado en la página web y no solamente los cambios que la ejecución de este realice sobre la página.
- Es necesario adicionar un mecanismo de autenticación del servidor que clasifica las URL, por ejemplo, un certificado digital, para evitar suplantaciones del servidor.
- Por ser un prototipo, la herramienta se ejecuta a petición del usuario del navegador. Como consecuencia, un atacante puede realizar una inyección de código justo después de haber activado la extensión. Sin embargo, el usuario puede ejecutar la aplicación cuantas veces desee y ésta detectará cualquier cambio malicioso realizado a la página web hasta ese momento.

La extensión planteada busca la detección temprana de posibles fugas de información por medio de la alteración del código HTML de la página web. La propuesta no detecta ataques de Origen Cruzado ni XSS reflejado dado que estos ataques no realizan inyecciones o modificaciones al código HTML.

4. Conclusiones

La seguridad de las aplicaciones web depende tanto del fortalecimiento y protección de los servidores y canales de comunicación, como del fortalecimiento y protección de los clientes. Este trabajo se enfoca en la segunda problemática: cómo proteger a los clientes y en particular a los clientes de servicios bancarios; es necesario desarrollar soluciones para proteger los navegadores y así proteger a los usuarios de ataques del estilo *Man In the Browser-MIB*.

El contexto de los clientes de aplicaciones web bancarias tiene una característica particular, es posible establecer un conjunto pequeño y

bien definido de servidores que están autorizados para enviar información que permita construir una página web, y para recibir información del usuario. Como consecuencia es posible crear y administrar de forma simple una lista blanca que declare los servidores autorizados. Si bien esta característica limita el alcance de la propuesta, también permite que sea efectiva en el contexto bancario.

Este trabajo propone una extensión al navegador para analizar el código HTML resultante, después de la ejecución de los scripts que inyectan código. Es conveniente analizar una página después de la ejecución de los *scripts*, porque no requiere la construcción de un modelo complejo que permita el análisis estático de todos los scripts. La propuesta es simple, fácil de administrar y efectiva para detectar código html que no está autorizado.

Referencias bibliográficas

AMRO, Bela (2017). Malware detection techniques for mobile devices [online]. In: International Journal of Mobile Network Communications & Telematics, IJMNCT, Vol. 7, No.4-6 (dec). Chennai (Tamil Nadu, India): AIRCC Publishing Corporation. p. 1-10. ISSN: 1839-5678 <https://arxiv.org/ftp/arxiv/papers/1801/1801.02837.pdf>, <http//:doi.org/10.5121/ijmnct.2017.7601> [consult: 12/10/2018]

BARTH, Adam; WEINBERGER, Joel & SONG, Dawn (2009). Cross-Origin JavaScript Capability Leaks: Detection, Exploitation, and Defense [online]. In: 18th Conference on Security Symposium, SS'09 (10-14/08/2009). Montreal (Canada): USENIX Association. MONROSE, Fabian (ed.). Proceedings of the SS'09. Berkeley (CA, USA): USENIX Association. <https://www.adambarth.com/papers/2009/barth-weinberger-song.pdf>, <https://www.usenix.org/conference/usenixsecurity09/technical-sessions/presentation/cross-origin-javascript-capability-leaks> [consult: 08/08/2018]

BOUTIN, Jean Ian. (2014). The Evolution of Webinjects [online]. In: 24th Virus Bulletin International Conference (24-26/09/2014). Seattle (WA, USA): Virus Bulletin. Virus Bulletin Conference (sep), Abingdon (OX, UK): Virus Bulletin. p. 25-34. <https://www.virusbulletin.com/uploads/pdf/conference/vb2014/VB2014-Boutin.pdf> [consult: 07/09/2018]

CABRERA LAGUAPILLO, Mauricio Sebastián & LARCO ANDRADE, Yessenia Carolina (2018). Análisis de las técnicas de ocultación de código malicioso para evasión de sistemas de protección de usuario final y NIDS [en línea]. Trabajo de titulación (Ingeniero en Sistemas Informáticos y de Computación). Quito (Ecuador): Escuela Politécnica Nacional, Facultad de Ingeniería de Sistemas. 104 p. + Anexos. <http://bibdigital.epn.edu.ec/bitstream/15000/19514/1/CD-8906.pdf> [consulta: 10/10/2018]

CASTELLANOS, John Henry; WÜCHNER, Tobias; OCHOA, Martín & RUEDA, Sandra (2016). Q-Floid: Android Malware detection with Quantitative Data Flow Graphs [online]. In: MATHUR, Aditya & ROYCHOUDHURY, Abhik (eds.). Proceedings of the Singapore Cyber-security Conference (SG-CRC) 2016, Amsterdam (The Netherlands): IOS Press. Cryptology and Information Security Series, Vol. 14. p. 13-25. e-ISBN: 978-1-61499-617-0 2016. <http://doi.org/10.3233/978-1-61499-617-0-13>, <http://ebooks.iospress.nl/volumearticle/42050> [consult: 10/10/2018]

CHROME WEB STORE. (2018) WOT: Web of Trust, valoraciones de reputación de sitios web [online]. Mountain View (CA, USA): Google Inc. (actualización: 23/07/2018). <https://chrome.google.com/webstore/detail/wot-web-of-trust-website/bhmmomiinigofkjcapegjjndpbikblnp?hl=es> [consulta: 23/11/2018]

CONTINELLA, Andrea; CARMINATI, Michele; POLINO, Mario; LANZI, Andrea; ZANERO, Stefano & MAGGI, Federico (2017). Prometheus: Analyzing WebInject-based Information Stealers [online]. In: Journal of Computer Security, Vol. 25, No. 2. Amsterdam (The Netherlands): IOS Press. p. 117-137. e-ISSN: 1875-8924 <https://doi.org/10.3233/JCS-15773>, <https://content.iospress.com/articles/journal-of-computer-security/jcs15773> [consult: 20/11/2018]

DAMODARAN, Anusha; DITROIA, Fabio; VISAGGIO, Corrado Aaron; AUSTIN, Thomas H. & STAMP, Mark (2017). A comparison of static, dynamic, and hybrid analysis for malware detection [online]. In: Journal of Computer Virology and Hacking Techniques, vol. 13, No. 1 (feb). Paris (France): Springer Verlag p. 1-12. e-ISSN: 2263-8733. <https://doi.org/10.1007/s11416-015-0261-z>, <https://link.springer.com/article/10.1007%2Fs11416-015-0261-z> [consult: 13/08/2018]

HERRERA LOAIZA, Domingo Daniel (2017). Estudio de Vulnerabilidades en transacciones bancarias para dispositivos móviles con Sistema Operativo ANDROID [en línea]. Tesis (Ingeniero en Sistemas). Loja (Ecuador): Universidad Nacional de Loja, Facultad de la Energía, las Industrias y los Recursos Naturales No Renovables. 70 p. + anexos <http://dspace.unl.edu.ec/jspui/bitstream/123456789/18940/1/Herrera%20Loaiza%2c%20Domingo%20Daniel.pdf> [consulta: 12/08/2018]

KUROSE, James F. & ROSS, Keith W. (2012). Computer Networking: A Top-Down Approach. 6 ed. Boston (MA, USA): Pearson Education. 862 p. ISBN: 978-0-13-285620-1ODA, Terri; WURSTER, Glenn; VAN OORSCHOT, P. C. & SOMAYAJI, Anil (2008). SOMA: Mutual Approval for Included Content in Web Pages [online]. In: 15th ACM Conference on Computer and Communications Security, CCS'08 (27-31/10/2008). Alexandria (VA, USA): ACM. SYVERSON, Paul (ed.). Proceedings of the CCS'08. New York (NY, USA): ACM. ISBN: 9781595938107 <http://people.scs.carleton.ca/~paulv/papers/ccs08.pdf> [consult: 09/08/2018]

LIU, Shukai; YAN,Xuexiong; WANG,Qingxian; ZHAO,Xu; CHAI,Chuansen & SUN, Yajing (2016). A protection mechanism against malicious HTML and Javascript code in vulnerable web applications [online]. In: Mathematical Problems in Engineering, Vol. 2016, Art. ID 7107042, London (UK): Hindawi Publishing Corporation, p. 1-14 e-ISSN: 1563-5147 <http://dx.doi.org/10.1155/2016/7107042>, <https://www.hindawi.com/journals/mpe/2016/7107042/abs/> [consult: 12/1072018]

OPEN WEB APPLICATION SECURITY PROJECT, OWASP (2017). OWASP Top Ten – 2017: Los diez riesgos más críticos en Aplicaciones Web [en línea]. Bel Air (MD, USA): OWASP Foundation. 24 p. <https://www.owasp.org/images/5/5e/OWASP-Top-10-2017-es.pdf> [consulta: 20/11/2018]

OPEN WEB APPLICATION SECURITY PROJECT, OWASP (2018). Cross Site Scripting (XSS) [online]. Bel Air (MD, USA): OWASP Foundation. Last revision: 06/05/2018. <https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)> [consult: 08/08/2018]

PROVOS, Niels; MAVROMMATIS, Panayiotis; RAJAB, Moheen Abu & MONROSE, Fabian (2008). All Your iFRAMEs Point to Us [online]. In: 17th Conference on Security Symposium, SS'08 (28/07-01/08/2008). San José (CA, USA): USENIX. Van OORSCHOT, Paul (ed.). Proceedings of the SS'08. Berkeley (CA, USA): USENIX Association. p. 1-15. <https://www.usenix.org/legacy/event/sec08/tech/full_papers/provos/provos.pdf> [consult: 08/08/2018]

PROVOS, Niels; MCNAMEE, Dean; MAVROMMATIS, Panayiotis; WANG, Ke & MODADUGU, Nagendra (2007). The Ghost in the Browser: Analysis of Web-based Malware [online]. In: First Workshop on Hot Topics in Understanding Botnets, HotBots '07 (11-13/04/2007). Cambridge (CA, USA): USENIX Association. Proceedings of the HotBots'07. Berkeley (CA, USA): USENIX Association. <https://www.usenix.org/legacy/event/hotbots07/tech/full_papers/provos/provos.pdf> [consult: 08/08/2018]

SHIM, Jaewoo; LIM, Kyeonghwan; CHO, Seong-je; HAN, Sangchul & PARK, Minkyu (2018). Static and Dynamic Analysis of Android Malware and Goodware written with Unity Framework [online]. In: Security and Communication Networks, Vol. 2018, Article ID 6280768. London (UK): Hindawi Limited/John Wiley & Sons, Inc. p. 1-12. e-ISSN: 1939-0122 <https://doi.org/10.1155/2018/6280768>, <http://downloads.hindawi.com/journals/scn/2018/6280768.pdf> [consult: 08/10/2018]

SYMANTEC. (2016). Internet Security Threat Report 2016 [online]. Mountain View (CA, USA): Symantec Corporation. Vol. 21 (abr). 80 p. <https://www.symantec.com/security-center/threat-report>, <https://www.symantec.com/content/dam/symantec/docs/security-center/archives/istr-16-april-volume-21-en.pdf> [consult: 20/11/2016]

SYMANTEC. (2017). Internet Security Threat Report 2017 [online]. Mountain View (CA. USA): Symantec Corporation. Vol. 22 (abr). 75 p. <https://www.symantec.com/security-center/threat-report>, <https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf> [consult: 20/11/2018]

VAN ACKER, Steven; HAUSKNECHT, Daniel & SABELFELD, Andrei (2018). Raising the Bar: Evaluating Origin-wide Security Manifests [online]. In: 2018 Annual Computer Security Applications Conference, ACSAC 34 (03-07/12/2018). San Juan (Puerto Rico): ACSAD. Silver Spring (MD, USA): Applied Computer Security Associates, ACSAD. <https://danielhausknecht.eu/papers/originmanifest_acsac2018.pdf> [consult: 16/10/2018]

WILLIAMS, Emma J.; HINDS, Joanne & JOINSON, Adam N. (2018). Exploring susceptibility to phishing in the workplace [online]. In: International Journal of Human-Computer Studies, Vol. 23 (dec). Amsterdam

(The Netherlands): Elsevier B.V. p. 1-13. ISSN: 1071-5819 <https://doi.org/10.1016/j.ijhcs.2018.06.004>, <https://www.sciencedirect.com/science/article/pii/S1071581918303628> [consult: 12/10/2018]

ZHANG, Xiaohan; ZHANG, Yuan; MO, Qianqian; XIA, Hao; YANG, Zhemin; YANG, Min; WANG, XiaoFeng; LU, Long y DUAN Haixin (2018). An Empirical Study of Web Resource Manipulation in Real-world Mobile Applications [online]. In: 27th USENIX Security Symposium (15-17/08/2018). Baltimore (MD, USA): USENIX Association. Proceedings of the USENIX Security '18. Berkeley (CA, USA): USENIX Association. p. 1183-1198. ISBN: 978-1-931971-46-1 <https://www.usenix.org/conference/usenixsecurity18/presentation/zhang-xiaohan>, <http://0b4af6cdc2f0c5998459-c0245c5c937c5dedcca3f1764ecc9b2f.r43.cf2.rackcdn.com/sec18/sec18_full_proceedings.pdf> [consult: 12/10/2018]