Seguridad en Aplicaciones web para Sistemas de Gestión Académica

Rene Guaman-Quinche[a], Hernan Torres-Carrion[b]

[a] Facultad de Ciencias Informáticas, Universidad Técnica de Manabí, Av. Urbina y Che Guevara, Portoviejo, Ecuador
eguaman@utm.edu.ec

[b] Área de la Energía, Universidad Nacional de Loja, Ciudadela Universitaria Guillermo Falconí Espinosa La Argelia. Loja, Ecuador
hltorres@unl.edu.ec

Resumen. El presente trabajo identifica dos principales problemas de seguridad informática en los sistemas web de Gestión Académica. Se analizaron los ataques más comunes a los que son expuestos los sistemas web y se propone un diseño que analice las peticiones de entrada con el fin de asegurar la integridad y confidencialidad de la información.

Palabras Clave: Inyección SQL, atacante, políticas de seguridad.

1 Introducción

Los sistemas web son necesarios en el desarrollo laboral tanto en el ámbito empresarial privado como en administraciones públicas. Este trabajo, se enfocará en los sistemas web de gestión académica SWGA (Sistemas web de Gestión Académica). Estos sistemas se encargan de las transacciones, tanto administrativas como académicas. Por tal motivo es necesario establecer mecanismos de protección, con la finalidad de proteger los datos de cada individuo y mantenerlos íntegros, como disponibles con los niveles de confidencialidad adecuada. De ahí la gran importancia de establecer, políticas para implantar la seguridad informática. La proliferación de código maligno y su rápida distribución a través del Internet, así como los miles de ataques e incidentes de seguridad que se producen todos los años, han contribuido a despertar un enorme interés por prevenir y detectar ataques a los SWGA.

En este ámbito, el proyecto abierto de seguridad de aplicaciones web es una comunidad abierta y libre que se enfoca en mejorar la seguridad en las aplicaciones de sistemas web. En el año 2010 se presentó una lista de riesgos y se concentró en las 10 principales amenazas web. Para cada una de ellas, se proporciona información básica acerca de la amenaza, los controles de seguridad y el impacto en las transacciones de una organización.

Actualmente el 80% de los ataques informáticos son llevados a cabo por código malicioso y las configuraciones por defecto que se realizan en el desarrollo de un sistema web hacen que el ataque sea una tarea sencilla. Por todo lo expuesto, se propone un diseño que permita detectar dos de los principales ataques que son la Inyección SQL y Cross-Site Scripting XSS, publicado en lista de fallos de seguridad

informática establecido por la fundación OWASP (Open web Application Security Project) en su artículo "OWASP Top Ten Project" (http://www.owasp.org/index.php/Top_10).

Además, se han propuesto un sinnúmero de soluciones para los numerosos ataques de aplicaciones web durante varios años. Y como se dice: quien tiene la información tiene el poder. Así tanto los administradores de seguridades y los atacantes han medido sus habilidades y se han dedicado grandes esfuerzos para que los sistemas web tengan una fortaleza segura o por el contrario se pueda vulnerar dichos sistemas para obtener la información. El mundo de la seguridad informática es un camino difícil, de allí la importancia que se debería dedicar a todos los aspectos relacionados a la seguridad informática y, en especial, a las seguridades en los SWGA.

2 Estado del arte

2.1 La seguridad de los sistemas web

José Ángel de Bustos hace una reflexión y afirma que para que un sistema informático sea seguro no basta con utilizarlo correctamente. Hace falta que esté libre de fallos, que no tenga puertas traseras y que no posea ninguna funcionalidad "no documentada". La única forma de poder fiarnos de la seguridad de un programa informático es tener a nuestra disposición el código fuente, ya que de esta manera podemos ver cómo ha sido desarrollado [1].

Es cierto que esta reflexión tiene dos caras, la primera es que al tener el código fuente disponible y por ende los intrusos tienen ventajas, ya que pueden descubrir los fallos y aprovecharse de ellos, y también existe gente dedicada a la "caza" de bugs que descubre los fallos, los pone en conocimiento de los usuarios y los arregla, con lo cual los usuarios pueden obrar en consecuencia.

Lo que ocurre es que no hay una "cultura" de seguridad en Internet. En la sociedad en que vivimos, hemos aprendido reglas básicas de protección de nuestros objetos personales para evitar pérdidas o robos. En cambio, nuestra experiencia con Internet es muy breve y es una realidad que cada vez más personas necesitarán conocer el manejo de las computadoras, así como las protecciones que día a día se van ofreciendo para garantizar la seguridad en el manejo de la información.

Las aplicaciones web, por definición, permiten el acceso de usuarios a recursos centrales tal como al servidor web y el cual permite el acceso a otros servidores de base de datos. Con los conocimientos y la implementación correcta de medidas de seguridad, se pueden proteger los recursos, así como proporcionar un entorno seguro en el cual los usuarios trabajen cómodos con su aplicación.

Una aplicación web, especialmente que se ejecuta en Internet, es más vulnerable a ataques de los hackers que una aplicación autónoma o cliente-servidor típico. Hay varias razones para esto:

- *Disponibilidad y accesibilidad*: Muchas aplicaciones web están disponibles para los usuarios públicos en cualquier momento del día o de la noche. Como los servidores web tienen que permitir el acceso a usuarios públicos y no tienen la protección completa de los cortafuegos típicos de una empresa.

- *Familiaridad*: La mayoría de los atacantes, incluso los menos sofisticados, conocen las interfaces web. Un navegador web es fácil de obtener y es uno de los programas de aplicación más comunes. El protocolo HTTP está bien definido, y existen muchas herramientas de hacking creadas para ayudar a los atacantes a penetrar y comprometer las aplicaciones web.
- *Facilidad*: La configuración de un servidor web, contenedor web y aplicación web para uso público es extremadamente compleja. Los atacantes, frecuentemente, pueden aprovechar esta complejidad y explotar deficiencias en la configuración de la aplicación o del sistema.
- *Publicidad*: El ego de algunos hackers experimentados es la publicidad, la fama, o un simple deseo de probar que pueden hacer algo que pocas personas pueden hacer [2].

2.2 Características generales de los SWG

Al tratarse de aplicaciones web estamos tratando con un término muy extenso. Entre la amplia gama de aplicaciones web están los SWGA que son sistemas que han sido desarrollados para ser utilizados por distintas unidades o áreas académicas, tomando como referencia dos ejemplos, la Universidad del País Vasco usa el sistema GAUR en sus diferentes campus ubicados en Vizcaya, Gipúzcua y Álava y la Universidad Nacional de Loja que usa el SGA.

Estos sistemas web se encargan de los procesos en el ámbito académico como administrativo y adaptan las funcionalidades de acuerdo con los procesos que desarrollan cada dependencia (alumnos, profesores, administrativos), es decir cada una carga solo los datos que le competen y todo va a una base de datos integrada que permite que no haya duplicidad de información y que el responsable de administrar la información sea quien la cargue al sistema. Toda la información es compartida por las diferentes sedes en forma online

Ahora es necesario centrarnos en la gestión académica y dentro de esta parte del sistema es importante considerar la gestión estudiantil como parte esencial del proceso de aprendizaje de los estudiantes porque gran parte del servicio que se presta a los estudiantes dependerá de la manera en la que se administre los aspectos esenciales que intervienen una gestión académica integral tal como la información personal del estudiante, sus calificaciones de cada periodo académico e incluso su asistencia.

Este tipo de sistemas establece un vínculo entre el docente y el estudiante manteniendo una gestión apropiada para la notificación de novedades, eventos académicos, publicación de notas, asistencias que puedan surgir en el transcurso del periodo académico.

Para poder ir delimitando las seguridades informáticas es necesario especificar que la gestión académica es aquella que se refiere a los datos relacionados con los alumnos y su currículum universitario. De este modo podríamos enumerar ciertos contenidos que pueden ser considerados información académica como: datos personales, datos académicos y datos logísticos.

No se considerará como información académica aquella que esté relacionada con los alumnos, como pueden ser los importes de la matriculación y créditos cursados,

los datos de la domiciliación bancaria, el importe de becas que le hayan sido concedidas. Este tipo de datos del alumno se ubicará dentro del entorno administrativo. En España, el Reglamento 994/1999 establece las medidas de seguridad para la protección de la información académica [3].

Al conocer los procesos que se realizan en éstos sistemas podemos entender que los principales atacantes son por lo general los mismos usuarios que interactúan con el sistema. Un ejemplo de ello es un posible estudiante que desee modificar sus notas o asistencias, por ello trata de encontrar alguna vulnerabilidad y poder modificar los datos académicos de su expediente. Otra vulnerabilidad puede nacer desde los propios docentes como por ejemplo corregir alguna nota o asistencia mal ingresada.

Como hemos analizado, este tipo de vulnerabilidades son típicos ataques que se realizan con el fin de modificar los datos de los repositorios de almacenamientos, como consecuencia un ataque de Inyección SQL es un ataque común para este tipo de sistemas.

2.3 Ataques vía web

Los ataques a las páginas web de una organización son casi siempre los más vistosos, en cuestión de minutos piratas de todo el mundo se enteran de cualquier problema de una página web principal. Por supuesto, la noticia de la modificación salta inmediatamente a los medios, que gracias a ella pueden rellenar alguna cabecera sensacionalista sobre los piratas de la red, y así se consigue que la imagen de la empresa atacada caiga notablemente; y la del grupo de piratas suba entre la comunidad nacional o internacional.

La mayor parte de estos ataques tiene éxito gracias a una configuración incorrecta del servidor, errores de diseño del mismo, al código de la propia aplicación web. En organizaciones grandes estos ataques suelen ser bastante complejos (alta disponibilidad, balanceo de carga, sistemas propietarios de actualización de contenidos...) y difíciles de administrar correctamente, mientras que si la empresa es pequeña es muy posible que haya elegido un servidor web simple en su instalación y administración pero en el cual es casi imposible garantizar una mínima seguridad. Sea por el motivo que sea, la cuestión es que cada día es más sencillo para un pirata ejecutar órdenes de forma remota en una máquina, o al menos modificar contenidos de forma no autorizada.

Cualquier analizador que podamos ejecutar contra nuestros sistemas es capaz de revelar información que nos va a resultar útil a la hora de reforzar la seguridad de nuestros servidores web.

En cualquier servidor web es muy importante el usuario bajo cuya identidad se ejecuta el servicio httpd, ese usuario no debe ser nunca el root del sistema, pero tampoco un usuario sin privilegios; se ha de tratar siempre de un usuario dedicado y sin acceso real al sistema. Por supuesto, las páginas HTML nunca deben ser de su propiedad, y mucho menos ese usuario ha de tener permiso de escritura sobre los mismos, con un acceso de lectura es más que suficiente en la mayoría de los casos.

Hemos de tener en cuenta que si el usuario que ejecuta el servidor puede escribir en las páginas web, y un pirata consigue ejecutar órdenes bajo la identidad de dicho usuario, podría modificar las páginas web sin ningún problema.

Una característica importante de la detección de ataques vía web es que no suelen generar muchos falsos positivos, por lo que la configuración de la base de datos inicial es rápida y sencilla

OWASP afirma que las amenazas para las aplicaciones web cambian constantemente. Los factores clave en esta evolución son los avances hechos por los atacantes, la liberación de nueva tecnología, así como la instalación de sistemas cada vez más complejos. Los atacantes pueden potencialmente usar muchas diferentes rutas a través de su aplicación para causar daño en su negocio u organización. Cada una de estas rutas representa un riesgo que puede, o no, ser lo suficientemente serio como para merecer atención. A veces, estas rutas son triviales de encontrar y explotar y a veces son extremadamente difíciles. De manera similar, el daño causado puede ir de ninguno hasta incluso sacarlo del negocio. Para determinar el riesgo para su organización, puede evaluar la probabilidad asociada con cada agente de amenaza, vector de ataque y debilidad de seguridad y combinarla con una estimación del impacto técnico y de negocios en su organización. Juntos, estos factores determinan el riesgo total.

OWASP enfoca los riesgos más serios y más comunes para un amplio tipo de organizaciones incluyendo los SWGA, estos riesgos fueron publicados en el 2007 y se basó en los datos de la lista CVE (Es una lista ampliamente aceptada que contienen vulnerabilidades de aplicaciones web. Organizado por la Corporación MITRE), que durante 5 años ha estado siguiendo los tipos de errores que conllevan a vulnerabilidades en los sistemas web. El resumen de resultados estableció que en el año 2005 y 2006 que XSS fue la vulnerabilidad más crítica con el 18,5%, e Inyección SQL fue la segunda con 13.6% [4].

Inyección SQL y XSS permiten a los atacantes el acceso no autorizado a datos (leer, insertar, modificar o borrar), acceder a las cuentas de base de datos, a suplantar a otro usuario (por ejemplo, el administrador), imitan a las aplicaciones web para obtener el acceso con el servidor web, etc.

La vulnerabilidad Inyección SQL se presenta en la incorrecta validación de las variables de entrada del usuario y las que contienen sentencias SQL. Con el propósito de atacar a los servidores web o base de datos, los hackers utilizan códigos maliciosos en los formularios web o URL para construir sentencias SQL a través de una vulnerabilidad de inyección SQL. Es una clase más general de vulnerabilidades que puede ocurrir en cualquier lenguaje de programación o script. El código siguiente ilustra un tipo de vulnerabilidad de Inyección SQL:

```
Sentencia = SELECT*FROM usuario where nombre= ' " + nombreUsuario + " ';"
```

El “usuario” es la variable en la cual el usuario debe ingresar datos, y una vez que los hackers explotan esta variable (un ejemplo 'a'='a), la sentencia sql se verá en la obligación de ejecutarla.

```
DROP TABLE usuario; SELECT * FROM infoUsuario WHERE 'a' = 'a
```

Esta sentencia eliminará la tabla “usuario” y revela la información de los usuarios.

La figura 1 muestra la forma de detectar vulnerabilidades de Inyección SQL a través de rastreadores web (web Crawler) y estos a su vez interactúan con las aplicaciones web, simulan el ataque mediante el análisis de la información recogida por web Crawler, el motor de detección construye un código y lo envía a los servidores web. El motor de detección espera la respuesta y el análisis de entrada de texto, una vez que las palabras claves puedan detectar los datos de respuesta e identificar la vulnerabilidad.

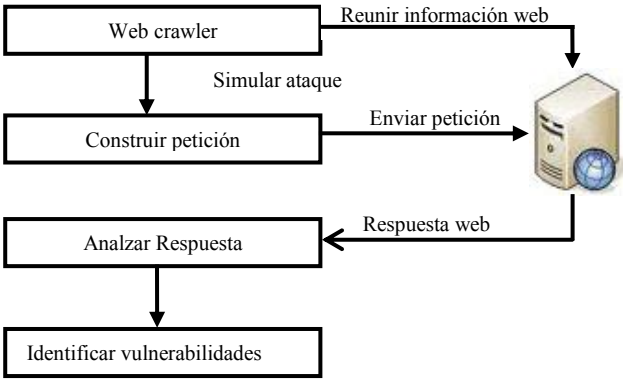

Fig. 1. Detección de Vulnerabilidades de Inyección SQL usando un simulador de códigos de ataques.

Se ha logrado detectar 30 tipos de ataque de código. Se describe algunos ejemplos de inyección sql[5].

- ‘OR 1=1-
- ‘%22
- ‘)OR
- ‘
- %3B

Si analizamos, el atacante cuando construye una sentencia SQL, por lo general suele usar espacios, comillas simples, comillas dobles o guiones y por lo general, es necesario hacer un análisis por separado de la sentencia SQL y la Inyección SQL y separar la cadena SQL a través de un arreglo.

Tabla 1. Resultado del análisis de una consulta SQL

0	1	2	3	4
Select * from	tabla	where	Atributo =	Entrada del usuario

Ahora si se agrega un ataque (inyección SQL) a una sentencia SQL, veremos su forma de construir dicho ataque.

Tabla 2. Ataque con una consulta de inyección SQL

0	1	2	3	4	5	6	7	8
Select * from	tabla	where	atributo =	Entrada del usuario	or	1	=	1

Por otra parte, los ataques de tipo XSS son ataques contra aplicaciones web en los que el atacante toma el control sobre el navegador de un usuario con el objetivo de ejecutar código o script malicioso dentro de un entorno de confianza del sitio web asociado a la aplicación final. Si dicho código es ejecutado satisfactoriamente, el atacante puede obtener acceso de forma activa o pasiva, a recursos de los navegadores web asociados con la aplicación, tales como los cookies e identificadora de sesión [6].

Existen tres tipos conocidos de secuencias de comandos en sitios cruzados: reflejado, almacenado e inyección DOM. XSS reflejado es el más fácil para explotar una página reflejará información suministrada por el usuario:

Echo $_REQUEST[‘entradaUsuario’]

El XSS almacenado toma la información y la almacena, en un fichero, una base de datos, u otro sistema de almacenamiento, y luego en una etapa posterior, muestra dicha información sin filtrado alguno. Esto es extremadamente peligroso en sistemas de administración de contenidos, blogs, o foros, donde un gran número de usuarios accederá a la información introducida por otros usuarios.

Con ataques basados en inyección DOM, el código JavaScript del sitio web y sus variables son manipulados, en lugar de los elementos HTML. Alternativamente, los ataques pueden ser una mezcla o un híbrido de los tres tipos mencionados anteriormente. El peligro con la secuencia de comandos en sitios cruzados no es el tipo de ataque, sino la posibilidad del mismo. Un comportamiento inesperado del navegador web puede introducir sutiles vectores de ataque.

Los ataques son implementados generalmente en JavaScript, que es un poderoso lenguaje de secuencia de comandos. JavaScript permite a los atacantes manipular cualquier aspecto de la página mostrada, incluyendo agregar nuevos elementos, manipular cualquier aspecto del árbol interno DOM, y borrar o cambiar la manera en la cual una página es visualizada. JavaScript permite la utilización de XmlHttpRequest , el cual es utilizado en sitios con tecnologías AJAX, incluso si el sitio de la víctima no utiliza AJAX actualmente. Algunas veces es posible evadir la política que indica al navegador web enviar la información a su origen utilizando XmlHttpRequest y por lo tanto reenviar la información de la víctima a sitios hostiles, y crear complejos gusanos y virus maliciosos que se mantendrán activos mientras el navegador web se encuentre abierto.

Ahora es necesario verificar que todos los parámetros en la aplicación sean validados y/o codificados antes de ser incluidos en páginas HTML. Vamos a exponer dos tipos de verificaciones que son:

- *Verificación automatizada*: herramientas automáticas de testeo de penetración son capaces de detectar XSS reflejado a través de inyección de parámetros, pero generalmente fallan a la hora de encontrar XSS persistente, particularmente si la salida del vector XSS inyectado es restringida a través de pruebas de autorización. Las herramientas automáticas de escaneo de código fuente pueden encontrar APIs débiles o peligrosos pero normalmente no pueden determinar si la validación o codificación que se ha realizado, lo cual puede resultar en falsos positivos. Ninguna herramienta es capaz de encontrar XSS basado en DOM, lo cual significa que las aplicaciones basadas en Ajax frecuentemente se encontrarán en riesgo si sólo se ha realizado una verificación automatizada.
- *Verificación manual*: si se ha utilizado un mecanismo centralizado de validación y codificación, la manera más eficiente de controlar la seguridad es revisando el código. Si en cambio, se ha utilizado una implementación distribuida, entonces la verificación tomará considerablemente mucho más tiempo. El testeo toma mucho tiempo ya que la superficie de ataque en la mayoría de las aplicaciones es muy amplia.

La mejor protección para XSS es una combinación de validación de listas blancas ("whitelists") de toda la información entrante y una apropiada codificación de la información saliente. La validación permite la detección de ataques, y la codificación previene cualquier inyección de secuencia de comandos de ejecutarse exitosamente en el navegador. Prevenir XSS en una aplicación entera requiere una solución de arquitectura consistente en:

- *Validación de entrada*: utilizar un mecanismo estándar de validación de entrada para validar toda la información entrante contra longitud, tipo, sintaxis, y reglas de negocio antes de permitir que la información sea visualizada o almacenada. Utilizar una estrategia de validación de "aceptar solo lo bueno", rechazando la información inválida en lugar de rehabilitar posible información hostil. No olvidar que los mensajes de errores pueden también contener información inválida.
- *Codificación fuerte de salida*: asegurar que toda la información suministrada por el usuario sea apropiadamente codificada antes de devolver la página, mediante la codificación de todos los caracteres a excepción de un pequeño subgrupo. También, configurar la codificación de caracteres para cada página de salida, lo cual reducirá la exposición a algunas variantes.
- *Especificación de la codificación de salida*: no permitir que el atacante elija esto en nombre de los usuarios.
- *No utilizar la validación de lista negra "blacklist"*: para detectar XSS en la entrada o para validar la salida. Buscar y reemplazar solo algunos caracteres ("<" ">" y otros caracteres similares o frases tales como "script") es una técnica débil y ha sido atacada satisfactoriamente con anterioridad. Incluso una etiqueta "<b>" sin verificar es inseguro en algunos contextos. XSS tiene un sorprendente número de variantes que hacen sencillo evitar la validación de listas negras.
- *Cuidado con los errores canónicos*: Los valores de entrada deben ser decodificados y canonizados a la representación interna de la aplicación antes de ser validados. Asegurarse que la aplicación no decodifique la misma entrada dos veces. Tales errores pueden ser usados para evadir los esquemas de listas blancas

"whitelists" introduciendo entradas peligrosas luego de haber sido controladas [7].

3. Solución

Este análisis básico puede ser fácilmente implementado definiendo una lista de caracteres no aceptables, luego, el proceso de análisis simplemente rechaza todo lo que está incluido en dicha lista para mejorar el proceso de filtrado de una petición realizada por el usuario. De todas maneras, consideramos que esta estrategia es básica, demasiada limitada y fácil de evadir por algún atacante experto. Scott, D [8] proponen un servidor proxy que se situaría en el servidor de la aplicación web con la finalidad de filtrar el flujo de datos de entrada y salida. Dicho proceso de filtrado tiene en cuenta un conjunto de políticas de seguridad diseñada por los desarrolladores de las aplicaciones web.

En consecuencia, la falta de análisis sobre las estructuras de un SWGA puede ser utilizada por atacantes expertos para evadir sus mecanismos de detección y realizar peticiones maliciosas. El simple uso de expresiones regulares (código de Inyección SQL) puede ser utilizado para evadir estos filtros. Hay que tener en cuenta que todo análisis que se realice hay que evitar los falsos positivos.

Podemos observar en la Fig. 2. que define un diagrama del modelo clásico de seguridad y que ha cambiado ligeramente, se ha agregado un servidor de seguridad, el mismo que se encargará y se hará responsable del manejo de las políticas de seguridad, los códigos de diseño de un conjunto de restricciones de validación y las peticiones de entrada que los analistas programadores hayan ingresado o configurado. Además dentro de las restricciones de validación es necesario imponer restricciones en los datos de las cookies, los parámetros URL.

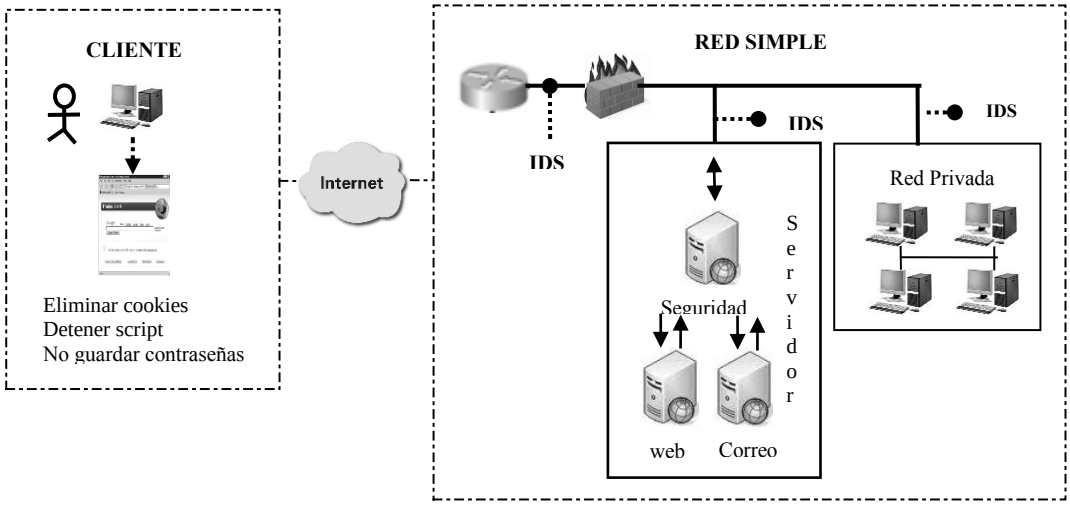

Fig. 2. Esquema de seguridad aplicando un servidor de seguridad de red o sistema informático.

En el algoritmo 1 ilustramos un ejemplo de filtrado de entradas.

Algoritmo 1. Adaptación del algoritmo de detección de Inyección de Kang S.[9]

```
<?php
   $nombre = $_POST["usuario"];
   $clave = $_POST["password"];
   $query1 = "SELECT * FROM usuario WHERE cedula='cedula' and
password='clave';";
   $query2 = "SELECT * FROM usuario WHERE cedula='".$nombre."'
and password='".$clave."';";
   $tokens1 = split("[\\s']|(--)", $query1);
   $tokens2 = split("[\\s']|(--)", $query2);
   if(count($tokens1) != count($tokens2)){
       echo $msj . "<br/>";
?>
       <script language="javascript">alert('Es una Inyección
SQL');
   //aquí me envía error de sintaxis </script>
<?php
}else{
?>
  //aquí se debería hacer el enlace de nuevo a la aplicación web
       <script language="javascript">alert('NO es una inyección
SQL'); </script>
<?php
}
?>
```

Para terminar esta apartado, los usuarios de los sistemas web se preguntarán cómo un atacante puede acceder a la información que almacenan los sistemas web. Tomemos como ejemplo un servidor de base de datos que se encuentra detrás de una serie de seguridades de redes clásicas como Firewall, IDS u otros mecanismos de protección. Existe un canal de comunicación entre el servidor web y el servidor de base de datos y por la cual el atacante utiliza para llegar hasta los datos guardados en la base de datos. Ahora hay que entender la naturaleza o arquitectura que comúnmente tiene un sistema web; existen tres niveles claramente definidos, el primero de ellos es la vista o interfaz de usuario y es el encargado de la interacción entre los usuarios y el sistema. El GAUR y el SGA son aplicaciones típicas de los SWGA. El segundo de los niveles es donde reside la lógica de la aplicación, esta correrá sobre los servidores web, los de correo electrónico y los de aplicación. Y el tercer nivel tendremos el almacén de datos, es decir, el repositorio de la información de una institución universitaria, que podría ser la UPV o la UNL, cada nivel existe una conexión.

Un escenario para un ataque en un SWGA lo mostraremos en el siguiente ejemplo.

Para poder ingresar al SGA hay que iniciar una sesión como se puede observar en la figura 3.

Fig. 3. Interfaz de autenticación de usuarios

Normalmente un usuario para ingresar a este sistema, introduce su número de identificación y su contraseña en los campos respectivos de la aplicación, al seleccionar la opción "Ingresar" el usuario a través del navegador envían una petición al Servidor de Seguridad, el mismo que se encarga de analizar y evaluar la petición realizada por el navegador web Cliente. Si no existe código malicioso, es enviada al servidor de base de datos para que la sentencia SQL sea ejecutada.

Pero si el usuario es un atacante, el ingresará en uno de sus campos de la aplicación una inyección como por ejemplo 'or'1'='1 y si el sistema web no valida correctamente esta vulnerabilidad, se concatena el código ingresado con la sentencia SQL para poder ser ejecutada a posteriori. Al enviarse la petición entra en acción el servidor de seguridad, recepta la petición y analiza su contenido y si encuentra código incrustado o código malicioso, rechaza la petición, enviado una respuesta de error al navegador cliente como podemos observar en la figura 4

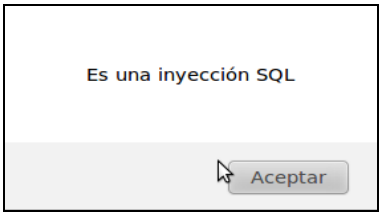

Fig. 4. Mensaje de Error "Es una Inyección SQL"

4. Conclusiones

Los diseños, arquitecturas o soluciones deben encargarse de verificar las seguridades tanto en el lado del cliente como del servidor. En el cliente se debe realizar un conjunto de acciones sobre los recursos del navegador pertenecientes a la aplicación web como validaciones de filtrado de datos dentro de las aplicaciones usando tecnología Ajax para evitar saturar con demasiadas peticiones al servidor. El objetivo no es sólo prevenir ataques de Inyección SQL, sino también otras tecnologías

y lenguajes utilizados en las aplicaciones web y que pueden ser potencialmente peligrosos para la protección de recursos desde el navegador hacia el servidor.

En el futuro, es necesario analizar también el código de las aplicaciones web, con estos resultados será posible construir un modelo realista para prevenir no solo los del tipo Inyección SQL sino agregar más funcionalidades para prevenir las ocho vulnerabilidades de la lista presentada por OWASP. Esto nos ayudará a extender nuestro diseño para completar una arquitectura de seguridad y sería usado para evaluar todas las peticiones disponibles en un dominio público.

Agradecimientos

Este trabajo ha sido auto-financiado por los autores. El presente artículo forma parte del Trabajo de Fin de Máster en Sistemas Informáticos Avanzados.

Los autores desean expresar su agradecimiento a los técnicos de la Unidad de Telecomunicaciones e Información por su participación en la ejecución de la propuesta y a los Profesores del Máster en Sistemas Informáticos Avanzados por las tutorías brindadas.

Referencias

1. *Bustos, S. Seguridad y Software Libre.* http://www.kriptopolis.org/seguridad-y-software-libre*,* Kriptópolis (2002), Accedido el 01 de Junio de 2010
2. Saura, J., Implantación de seguridad en entornos web, Universidad Politécnica de Cartagena, Colombia, 165pp (2006)
3. Canaleta, X., Gestión académica y protección de datos, Universidad Ramón Llull, España, Abril (2010)
4. Steve, C., Martin, R., "Vulnerability Type Distributions in CVE", http://cwe.mitre.org/documents/vuln-trends/index.html#summary (2007), Accedido el 13 de Junio de 2010
5. AlfaroWang Xin, otros, Hidden web Crawling For Inyeción SQL Detection, IEEE, China (2010)
6. Garcia, A., Prevención de ataques de Cross-Site Scripting en aplicaciones web, Universidad Autónoma de Barcelona (2007)
7. Wang Xin, otros, Hidden web Crawling For Inyeción SQL Detection, IEEE, China (2010)
8. Scott, D. and Sharp, R. Abstracting application-level web security. 11th Internation Conference on the WWW, (2002)
9. Huerta, A., Villalón. Seguridad en Unix y redes, Capítulo 16, Página 259, España (2002)
10. Scott. D., Sharp. R., Specifying and Enforcing Application-Level web Security Policies, IEEE Knowledge and Data Engineering, United Kingdom (2003)
11. Segu-Info, http://www.seguinfo.com.ar/politicas/polseginf.htm (2008), Accedido el 11 de Juno de 2010
12. Protección de datos. http://protecciondedatos.org/info.php, Accedido el 09 de Junio de 2010
13. Furnell, S..: Vulnerability Exploitation: the Problem of Protecting our Weakest Links, Computer&Fraud, United Kingdom (2003)
14. Gómez, A., Seguridad Informática Básica, España, 1ª Edición (2010)
(2004)