Preproceso de Formularios para el Análisis de Seguridad de las Aplicaciones Web

Fernando Román Muñoz, Luis Javier García Villalba

Grupo de Análisis, Seguridad y Sistemas (GASS)
Departamento de Ingeniería del Software e Inteligencia Artificial (DISIA)
Facultad de Informática, Despacho 431
Universidad Complutense de Madrid (UCM)
Calle Profesor José García Santesmases s/n
Ciudad Universitaria, 28040 Madrid
E-mail: {froman, javiergv}@fdi.ucm.es

***Resumen*—Actualmente es habitual en las organizaciones que previamente a la puesta en producción de una nueva aplicación Web se le realice un análisis de vulnerabilidades. Estos análisis pueden ser de varios tipos, siendo el más habitual el análisis dinámico con herramientas automáticas. En los análisis dinámicos de vulnerabilidades se accede a la aplicación como lo haría un usuario habitual de la misma. Principalmente el análisis dinámico se realiza en dos etapas: rastreo y prueba. En la fase de rastreo se intentan encontrar todos los elementos que componen la aplicación. En la fase de prueba se introducen ciertos valores en la aplicación, y se analizan las respuestas en busca de patrones que se correspondan con vulnerabilidades conocidas. Una de las principales carencias en estos momentos del análisis dinámico con herramientas automáticas, es que durante la fase de rastreo no se detectas todas las páginas de la aplicación. Entre otras causas, esto es debido a que no se introducen siempre valores válidos en los campos de los formularios que tenga la aplicación. Estas herramientas incorporan un conjunto predefinido y limitado de pares campo-valor. Cuando encuentran un campo de formulario lo buscan en este conjunto y si lo encuentran usan el valor asociado. En caso contrario envían un valor por defecto. Algunas de estas herramientas permiten a su usuario ampliar este conjunto de campos y valores. En este trabajo se propone un método para preprocesar los formularios de las aplicaciones Web, de forma que se obtengan valores válidos para sus campos. Posteriormente estos pares campo-valor se añadirían a la herramienta automática para que se realizara un análisis habitual. Este análisis habitual usaría, no sólo el conjunto de valores predefinidos, sino los que se hayan encontrado en el preproceso del los formularios. De esta forma se detectarán páginas de la aplicación que de forma habitual no serían alcanzadas. El preproceso de formularios Web que se propone en este trabajo consiste básicamente en acceder la aplicación con un navegador automatizado, de forma que se ejecute el código cliente de la página. Después se buscan los formularios que contenga. Para cada uno de ellos se obtienen valores para los campos de selección, y una lista de campos de texto. Los valores candidatos para los campos de texto se buscan en una fuente de información externa. A continuación se prueban los formularios con los valores encontrados. Durante las pruebas realizadas se ha comprobado que el método propuesto puede proporcionar valores válidos para los campos de formularios de ciertas páginas Web, de forma que mejoren los resultados de los análisis dinámicos de vulnerabilidades con herramientas automáticas.**

I. Introducción

En la actualidad el uso de las aplicaciones Web se ha extendido a muchos ámbitos de la vida cotidiana, estando disponible en cualquier momento a un gran número de usuarios potenciales. Estas aplicaciones suelen desarrollarse con el objetivo de proporcionar cierta funcionalidad, dejando en muchos casos la seguridad en un segundo plano. Una vez que se tiene la aplicación en producción se tratarán de forma reactiva las vulnerabilidades según vayan detectándose.

Para intentar corregir este comportamiento, se ha atacado el problema desde dos frentes. Por un lado se ha incorporando La Seguridad en las fases del ciclo de vida de desarrollo. Por otro lado se ha introducido una fase de análisis de vulnerabilidades de las aplicaciones Web, antes de su entrada en producción. Una opción bastante popular para el análisis de vulnerabilidades es utilizar una herramienta automática de análisis dinámico de vulnerabilidades. En este trabajo trataremos uno de los aspectos que no suelen tratar bien estas herramientas, como es el rellenar con valores correctos los campos de los formularios que tenga la aplicación.

Este artículo se estructura en 10 apartados incluida la presente introducción. En el apartado II introduciremos el concepto de aplicación Web vulnerable. En el III los métodos habituales de detección de vulnerabilidades en estas aplicaciones. En IV y V se describen las herramientas automáticas para realizar esta tarea y, en concreto, de dónde obtienen los valores para rellenar los formularios Web que encuentren. En VI se trata el rastreo de la Web oculta (o profunda), un tema similar al que se aborda en este trabajo. En VII y VIII se detalla el problema que se trata de resolver y se explica la solución aportada. En IX se incluye una breve descripción de la pruebas realizadas y sus resultados. Finalmente, en X, se muestran las conclusiones y el trabajo futuro.

II. Aplicaciones Web Vulnerables

Esquemáticamente la forma de funcionar de una aplicación Web es la siguiente. El usuario, a través del navegador, hace una petición al servidor Web. El servidor Web suele alojar el contenido estático, y para el resto del contenido redirige la

petición al servidor de aplicación. Si es necesario acceder a información se traslada la petición al servidor de la base de datos que realiza la operación necesaria. A continuación el servidor de aplicación genera el contenido de la nueva página que es devuelta a través del servidor Web al navegador del usuario. Esta página puede incluir código ejecutable en el cliente, que también podrá generar dinámicamente parte de la página que se muestre al usuario.

Debido a que estas aplicaciones suelen estar siempre disponibles, cualquier persona puede interactuar con ellas, de manera malintencionada o no. Un usuario puede intentar saltarse el flujo predefinido o realizar peticiones no previstas. Como consecuencia de ello aumenta el riesgo de amenazas como la indisponibilidad, el acceso no autorizado o la divulgación de la información. Para disminuir el riesgo de estas amenazas es necesario implementar las contramedidas adecuadas a cada una de ellas. Si las contramedidas no existen o no están implementadas correctamente, la aplicación Web tendrá una debilidad que podrá ser aprovechada, es decir, tendrá una vulnerabilidad. En sentido inverso, detectando y solucionando las vulnerabilidades, se estarán implementando contramedidas adecuadas, con lo que se mitigarán los riesgos. Ejemplos de vulnerabilidades las podemos encontrar en listas como [16]. En estas listas se encuentran las más populares como la inyección (SQL, LPDAP, IMAP, etc.), el *Cross-Side Scripting* (XSS) o el *Cross-Site Request Forgery* (CSRF).También incluyen y otras no tan conocidas como la pérdida de autenticación y gestión de sesiones, o la referencia directa insegura a objetos.

III. Detección de Vulnerabilidades en Aplicaciones Web

Los métodos para detectar las vulnerabilidades en las aplicaciones Web pueden ser de dos tipos. Si se analiza el código fuente de la aplicación será un análisis estático. Por el contrario si se analiza la aplicación ejecutándola, será dinámico.

En los análisis estáticos se analiza el código de la aplicación para detectar fragmentos que se correspondan con patrones de vulnerabilidades conocidas. Sus principales ventajas son que no necesitan ejecutar la aplicación, que cubren todo el código y que en teoría pueden recorrer todos los caminos de ejecución. Su principal desventaja es que cada herramienta de este tipo sólo puede analizar un conjunto reducido de lenguajes. Además al no ejecutar el código puede producir demasiados falsos positivos.

En el análisis dinámico se ejecuta la aplicación desde un navegador (real o simulado). Se envian valores a la aplicación, y se registran y analizan las salidas producidas. Este tipo de análisis no depende del lenguaje empleado en la aplicación Web. Una de sus principales desventajas es que sólo se pueden analizar las páginas de la aplicación a la que se llegue. Esto obliga a realizar previamente un rastreo lo más completo posible de la aplicación. Otra desventaja es que no indica dónde está el problema en el código.

Como se indican en guías como las que publica [16] un análisis dinámico de vulnerabilidades se divide principalmente en dos fases. Una primera fase pasiva, también llamada de rastreo ("crawling"), y una segunda activa. En la primera fase se pretende localizar el mayor número posible de elementos de los que componen la aplicación. El objetivo es localizar todos los directorios y archivos de la aplicación, y los puntos de entrada de cada uno de ellos (por ejemplo, cabeceras, campos y cookies). En la segunda fase se realizan determinadas pruebas sobre los componentes de la aplicación para encontrar vulnerabilidades como las que figuran en las listas antes mencionadas. Se realizarán una serie de pruebas sobre estos elementos para comprobar por ejemplo la fortaleza del método de autenticación, o si filtra correctamente la información que introduzca un usuario.

Este análisis dinámico se puede realizar de forma manual, siguendo alguna guía como la mencionada anteriormente, utilizando una herramienta de análisis automático de vulnerabilidades como las que se indican en el apartado siguiente o una combinación de ambos.

IV. Herramientas Automáticas

Una de las aproximaciones tradicionales para la detección de vulnerabilidades en aplicaciones Web ha sido el uso de herramientas de análisis dinámico. Esta herramientas se puede usar de forma más o menos automática. Como datos de entrada necesitan al menos la URL o las URLS de la aplicación que se desea analizar. También será necesario facilitar a la herramienta credenciales de acceso, si procede y se dispone de ellas. Con esta información realizarán el análisis siguiendo las fases indicadas en el apartado anterior. Primero se localizan los componentes de la aplicación, y después se prueba sobre ellos patrones de vulnerabilidades conocidas.

Para que estas herramientas puedan encontrar vulnerabilidades como XSS, inyección SQL u otros tipos de inyección [4] [16], es necesario realizar un rastreo inicial lo más completo posible de la aplicación Web [2].

Las herramientas actuales en general no son capaces de realizar este rastreo completo [3] [8], principalmente por dos motivos. En primer lugar estas herramientas no son capaces de rellenar adecuadamente los formularios, especialmente los que tienen restricciones muy fuertes en sus campos. Por ejemplo los campos de un tipo y tamaño determinados, o campos cuyo valor depende del que se haya puesto en otro campo anterior. En segundo lugar a menudo no ejecutan correctamente el código de cliente que envía la aplicación al navegador, por ejemplo el código *JavaScript*.

V. Rastreo de Formularios en el Análisis Dinámico

Los formularios Web están formados por campos, en los que se introduce o selecciona una serie de valores, que son enviados al servidor Web para su proceso. Estos campos pueden ser básicamente de los siguientes tipos: oculto o visible al usuario, obligatorio o no, de selección (única, múltiple, combos, botones de selección, etc.) de texto (libre o clasificado), o con sus valores dependientes de los valores de otros. Una vez

que los valores llegan al servidor, este los procesa y devuelve una respuesta distinta en función de si los valores son válidos o no. Devolverá una página (la misma u otra) con distinto contenido, si los valores han tenido éxito; y devolverá el mismo formulario que habrá que completar correctamente o un mensaje de error, si los valores son inválidos o no son suficientes.

Como se ve en la tabla I las soluciones para encontrar las páginas que se encuentran detrás de los formularios son dos principalmente. La primera opción es rastrear la aplicación recopilando los enlaces que se encuentren, e introducir valores prefijados en los campos de los formularios. La segunda es guardar la navegación que realice un usuario de la aplicación. Y en una fase posterior de rastreo automático utilizar los valores que haya introducido el usuario en los campos de los formularios. En el primer caso se suele complementar con la selección automática de valores para los campos de selección (lista de opciones, botones, o cajas) por parte de la herramienta.

Un caso particular de formulario son los de inicio de sesión. En estos formularios el usuario introduce generalmente un identificador y una contraseña que la aplicación debe validar. Las herramientas actuales de análisis de vulnerabilidades suelen abordar este formulario de forma distinta al resto de formularios. Para ello permiten al usuario de la herramienta por ejemplo grabar una macro con el inicio de sesión, que posteriormente usará cuando sea necesario.

En la tabla I se indican a modo de ejemplo, las características de algunos escáneres de uso habitual en lo referente al relleno automático de formularios. Las herramientas y versiones que se incluyen son: Acunetix Web Security Scanner versión 7 [1], Burp Suite 1.3 Free Edition [5] y HP WebInspect versión 8.10 [10]. Estas herramientas se han seleccionado del conjunto de herramientas que habitualmente se usan para el análisis de vulnerabilidades por que incorporaran la opción de definir valores para los campos de los formularios.

Tabla I
CARACTERÍSTICAS DE LOS CAMPOS EN HERRAMIENTAS AUTOMÁTICAS

	Gn	Sd	Dvs	VU	Dc	Dv	Is
Acunetix	x	-	x	x	x	x	x
Burp Suite	-	x	x	-	x	-	-
HP WebInspect	x	-	x	x	x	-	x

La descripción de cada columna de la Tabla I es:

Gn Grabar navegación: la herramienta puede grabar una navegación de usuario y usar los valores usados por el usuario en el rastreo posterior.

Sd Solicitar datos durante el rastreo: si así se le indica, la herramienta pedirá durante el análisis la introducción de los valores de los campos de los formularios.

Dvs Definir valores: El usuario puede configurar la herramienta con valores asociados a campos, que posteriormente se usarán.

VU Valores por URL: Permite, dependiendo de la URL a analizar, definir unos valores específicos para los campos de los formularios que localice.

Dc Definir el campo: La herramienta permite usar "Wildcards" para el nombre de los campos.

Dv Definir el valor: La herramienta permite usar "Wildcards" para el valor de los campos.

Is Inicio de sesión: la herramienta permite grabar previamente una macro de inicio de sesión.

El principal inconveniente de estos tipos de rastreo de formularios es que se confía plenamente en el usuario recorra la aplicación entera. Durante su navegación el usuario debe introducir valores que vayan a ser correctos posteriormente en el rastreo automático que haga la herramienta. Adicionalmente tienen el problema de que no suelen disponer de un método para determinar cuándo los valores introducidos en una página han tenido éxito o no.

En la Figura 1 podemos ver un ejemplo de la pantalla de la herramienta Burp Suite en la que se configuran los campos y los valores a usar en la fase de rastreo.

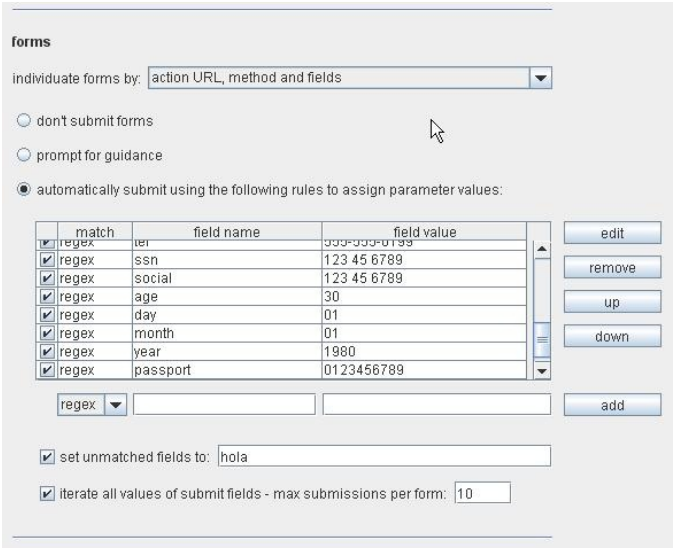

Figura 1. Introducción de campos y valores en la herramienta Burp Suite.

VI. WEB OCULTA

Un caso particular del problema de obtener valores válidos para los formularios se da en la exploración de la Web oculta o profunda (*deep Web crawling* en inglés). El objetivo del rastreo de la Web oculta es obtener la información que está detrás de formularios de búsqueda y eventualmente incorporarla a los buscadores tradicionales.

Hay dos formas principalmente para atacar la Web oculta [18]. Se pueden explorar a priori los formularios de búsqueda [7]; o se puede realizar la busqueda en tiempo real. En el primer caso los datos obtenidos se indexarán en el buscador para usarlos posteriormente. Como consecuencia los datos que se muestren puede que no sean actuales. En el segundo caso se realiza la búsqueda sobre un formulario que redirige la consulta a varios formularios predefinidos, según el dominio que se consulte.

Este tipo de solución sólo será posible para un conjunto finito de dominios.

En el primer caso habrá que obtener los valores que poner en cada campo. Se pueden obtener de distintas fuentes: los que aparezcan en los campos de selección [12] [19]; mantener una serie de valores para campos habituales como meses, días de la semana. códigos postales, etc., también en [12] [19]; extraerlos de las páginas de respuesta usando la medida TF-IDF *(Term Frequency-Inverse Document Frequency)* [12] o una similar; u obtenerlos de fuentes de información adicionales [19]. Una vez que se tiene una base de datos de campos y valores, se puede añadir a cada par campo-valor un peso según el éxito que haya tenido cuando se ha usado [19].

Adicionalmente se puede definir un método para saber si los campos enviados proporcionan respuestas suficientemente distintas. De forma que no haya que probar con todo el producto cartesiano de campos [12].

VII. Descripción del Problema

Lo que buscamos es una solución que aporte un conjunto de valores válidos para cada formulario Web con que se encuentre un escáner de vulnerabilidades durante la fase de rastreo, que pueda sustituir al rastreo manual previo. Estos valores deberán cumplir ciertos requisitos. Primero deben ser del tipo esperado por el campo. Por ejemplo si un campo se llama "email", se necesitará un valor similar a una dirección de correo electrónico. Segundo se deberán tener en cuenta las relaciones entre campos. Por ejemplo el valor de un campo "localidad" deberá ser consecuente con el valor de un campo anterior "provincia". Y en tercer lugar será necesario tener un criterio para determinar si los valores enviados han tenido éxito o no.

En la aportación que se va a describir a continuación se tendrán en cuenta las siguiente consideraciones: se van a rellenar los campos en el mismo orden que lo haría un usuario humano, comenzando con el primer campo del formulario según se lee y bajando hasta no encontrar ningún campo más; no se van a modificar los valores de los campos ocultos que no son visibles a un usuario humano; por ahora no se intentarán resolver los campos de Captcha (*Completely Automated Public Turing test to tell Computers and Humans Apart*); y no se van a atacar los formularios de inicio de sesión.

VIII. Descripción de la Solución

Para los requisitos indicados en el apartado anterior, definimos las siguientes características de nuestra solución:

I. Los valores de los campos de texto los vamos a extraer de una fuente de información externa. En este caso usaremos las Tablas de Fusión de Google [9], donde para cada campo de texto se intentará extraer un valor que sea del tipo buscado. Siguiendo con el ejemplo de apartado anterior, al buscar una tabla de fusión con un campo "email" posiblemente los valores que encontremos en ese campo de la tabla sean realmente direcciones de correo electrónico. Como fuente de información alternativa podría usarse un buscador de respuestas como WolframAlpha [21]. En este caso se ha optado por usar las Tablas de Fusión por que su estructura (columnas con valores) se asemeja a la que buscamos (campos con valores); y por que una misma tabla podría proporcionar valores para varios campos, como se ve más adelante. En la Figura 2 podemos ver un ejemplo de consulta a las Tablas de Fusión de Google, donde buscando por un nombre de campo obtenemos varias tablas que contienen valores para ese campo.

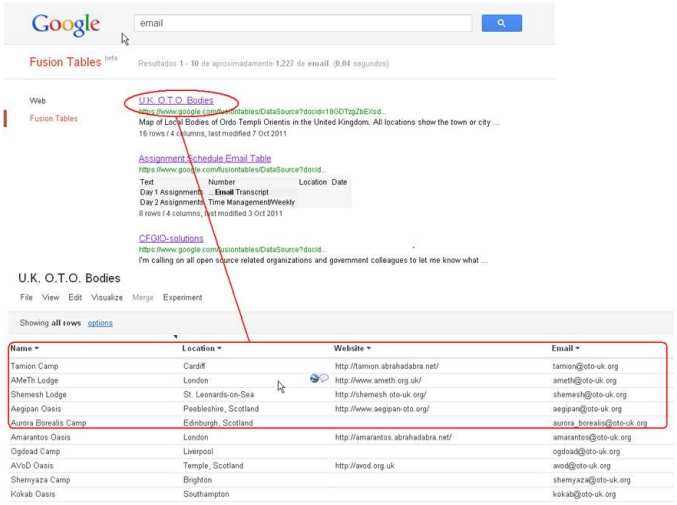

Figura 2. Búsqueda de campos en las Tablas de Fusión de Google.

II. Para obtener campos de texto relacionados entre ellos, por ejemplo una población con su provincia, también sirve la información que proporcionan las Tablas de Fusión, ya que si buscamos tablas con los campos "provincia" y "población" es probable que los registros de la tabla que encontremos cumplan la relación.

III. Si la relación se da en campos de selección es habitual que, por ejemplo al seleccionar una provincia en un campo "provincia', la página ejecute cierto código *JavaScript* que obtenga las poblaciones de esa provincia para un campo "población". En este caso se obtendrán los valores simulando un navegador que seleccione un valor para cada campo de selección que tenga la página.

IV. Para determinar si el envío de los valores ha tenido éxito se considera el criterio siguiente: si los valores son válidos y suficientes, llevarán a un nuevo contenido y serán los valores buscados; si son válidos pero insuficientes (falta alguno), devolverá la misma página con los valores válidos ya insertados en sus campos; y si no son válidos, devolverá el mismo formulario, con los campos inválidos vacíos. Por simplicidad se han tratado sólo estas tres opciones, aunque en cualquiera de los dos últimos casos, también la aplicación puede devolver una página de error.

V. En el caso particular de los campos de contraseñas, se ha optado porque se genere una contraseña de cierta complejidad para cada formulario que tenga un campo de este tipo, y se pone el mismo valor en todos los campos

de contraseña que tenga ese formulario.

A partir de estas características tenemos la descripción del algoritmo que se representa en la figura 3:

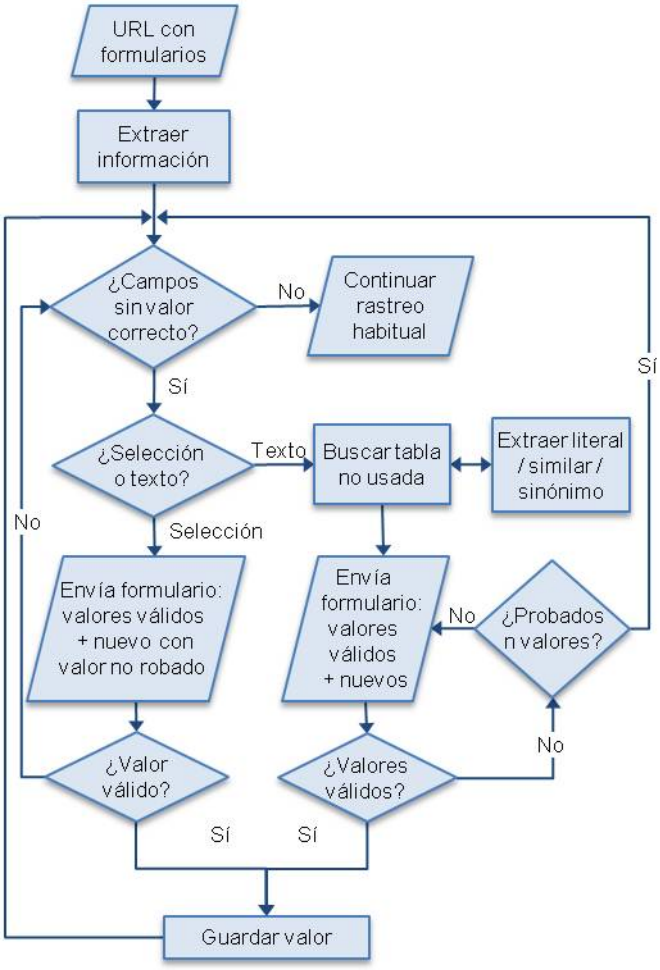

Figura 3. Proceso de búsqueda y selección de valores válidos.

a. Inicialmente se analiza la página, registrando los formularios y los campos con sus características. Para ello se siguen tres pasos: se visita la página usando un navegador en modo texto, que registra los formularios y sus campos; en segundo lugar un navegador gráfico automatizado selecciona valores en los campos de selección ejecutando el código JavaScript que exista (ver punto III anterior); y por último se analiza el DOM de la página para extraer los posibles valores de cada campo.
b. Para buscar los campos de texto en las Tablas de Fusión, se comienza buscando el nombre del campo, pero como en muchos casos el nombre asignado en el formulario no se corresponde con el literal que se muestra, como segunda opción se busca el literal más parecido usando la distancia de Levenshtein [20]; en tercer lugar el literal más cercano [11] y, por último, se buscan sinónimos [17].
c. Cuando se localice una tabla de fusión que tenga como columna el campo buscado, o (b), se tomarán valores para él y también para todos los demás campos de texto del formulario que aparezcan en esa tabla.
d. Comenzando con el primer campo se van buscando valores correctos para cada uno de ellos. En cada iteración se envían los campos con valores ya encontrados como correctos según IV, junto con el siguiente campo sin valor válido de la siguiente forma: si es un campo de selección se toma un valor de (a); y si es un campo de texto se envía ese campo, junto con los demás campos de texto todavía sin valor encontrado con los valores que se hayan obtenido en (c).
e. Si los valores de (c) no son correctos, en la siguiente iteración volverá a estar ese campo sin valor correcto, y se usarán valores distintos a los encontrados en la iteración anterior.
f. Si el formulario es de alta, por ejemplo en el registro de usuarios, al encontrar los valores correctos para cada campo, realmente se dará de alta un nuevo usuario, por lo que no podrán usarse una segunda vez. En este caso se propone guardar el siguiente valor de la tabla de fusión para cada valor válido.

IX. Implementación y Pruebas

Para la implementación de la solución, se ha elegido el lenguaje de programación Ruby empleado actualmente en varias herramientas de análisis de vulnerabilidades [14] pues incorpora librerías específicas para la navegación y análisis de páginas Web.

Ya que el rastreo de formularios de búsqueda está ampliamente estudiado [12], en este caso el algoritmo implementado se ha probado en formularios de alta, baja y modificación de tiendas virtuales [13] [15], encontrando de forma automática valores válidos para los campos que llevan a páginas que las herramientas habituales no son capaces de encontrar.

En las Tablas II y III se muestra el funcionamiento del algoritmo sobre un formulario modelo de registro de usuarios. Las características de los campos de este formulario se indican en la Tabla II. El formulario tiene campos tanto de texto como de selección, y dentro de estos tipos, de diferentes subtipos. Los valores de cada uno de estos campos se obtendrán de distintas fuentes de información, que se indican en la columna "Origen de los Valores" de la tabla. Los valores para los campos de texto los obtiene de alguna tabla de fusión de Google, para los de los campos de contraseñas los calcula el algoritmo, y los de selección (listas de opciones, botones, cajas, etc.) del propio formulario.

En la Tabla III se puede ver cómo se van obteniendo los valores para cada campo. Para cada iteración se indica el campo que se busca, los campos que se encuentran en esa búsqueda, y los campos enviados al servidor.

X. Conclusiones y Trabajo Futuro

En este trabajo se propone un método de preproceso de formularios Web para obtener de forma automática valores para sus campos, que puedan usarse posteriormente en las

Tabla II
CAMPOS DEL FORMULARIO PROBADO: SUS TIPOS, Y ORIGEN DE LOS VALORES UTILIZADOS

Nombre del campo	Tipo del campo	Subtipo de campo	Origen de los valores
firstname (C1)	Texto	Texto	T. de Fusión
lastname (C2)	Texto	Texto	T. de Fusión
email (C3)	Texto	Texto	T. de Fusión
address (C4)	Texto	Texto	T. de Fusión
password (C5)	Texto	Contraseña	Calculado
confirm (C6)	Texto	Contraseña	Calculado
country_id (C7)	Selección	Desplegable	Página
zone_id (C8)	Selección	Desplegable	Página
newsletter (C9)	Selección	Botón de selección	Página

Tabla III
ENVIANDO EL FORMULARIO 6 VECES SE OBTIENEN LOS VALORES DE LOS 9 CAMPOS NO OCULTOS

Iteración	Busca	Encuentra	Envía
1	C1	C1, C2 y C4	C1,C2 y C4
2	C3	C3	C1 al C4
3	C5	C5 y C6	C1 al C6
4	C7	C7	C1 al C7
5	C8	C8	C1 al C8
6	C9	C9	C1 al C9

siguientes fases del análisis de vulnerabilidades de una aplicación Web. Estos valores serán "mejores" que los que puedan incluir las herramientas actuales en dos aspectos. En primer lugar cumplirán mejor con las restricciones sobre el tipo de campo impuestas por la aplicación que se esté analizando, ya que previamente se habrá comprobado que son válidos para el campo del formulario en cuestión. En segundo lugar estarán relacionados entre ellos correctamente (porque estén relacionados en una tabla de fusión, o porque se haya ejecutado el código *JavaScript* asociado.

El siguiente paso será realizar pruebas sobre más aplicaciones Web, y si es posible pruebas masivas sobre un conjunto suficientemente representativo de aplicaciones Web. Para ello inicialmente se obtendrían valores candidatos para los campos de sus formularios, usando el método aquí descrito (ejecución de código cliente y consultas a Tablas de Fusión de Google). Estos valores candidatos se incorporarán a alguna herramienta de uso habitual como Burp Suite. A continuación se realizarán rastreos sobre ese conjunto de aplicaciones, para comparar el resultado con los obtenidos sin incorporar esta lista de campos-valores.

La principal limitación del método propuesto, además de las restricciones ya indicadas (no resolver campos de Captcha ni tratar los formularios de autenticación), es que no es posible encontrar en las Tablas de Fusión de Google valores para todos los campos. Este problema se podrá resolver parcialmente incorporando fuentes alternativas de información como WolframAlpha, y por el uso cada vez más extendido de las Tablas de Fusión, que incorpora nueva información de dominios diferentes. En el caso de los campos de Captcha, se podría intentar incorporar una herramienta como Decaptcha, que se describe en [6]. Y para los formularios de autenticación, las herramientas habituales ya incorporan soluciones razonables.

AGRADECIMIENTOS

Los autores agradecen la financiación que les brinda el Subprograma AVANZA COMPETITIVIDAD I+D+I del Ministerio de Industria, Turismo y Comercio (MITyC) a través del Proyecto TSI-020100-2011-165. Asimismo, los autores agradecen la financiación que les brinda el Programa de Cooperación Interuniversitaria de la Agencia Española de Cooperación Internacional para el Desarrollo (AECID), Programa PCI-AECID, a través de la Acción Integrada MAEC-AECID MEDITERRÁNEO A1/037528/11.

REFERENCIAS

[1] Acunetix. http://www.acunetix.com/.
[2] P. Baral, Web Application Scanners: A Review of Related Articles. IEEE Potentials, March 2011.
[3] J. Bau, E. Bursztein, D.Gupta, J. Mitchell, "State of the Art: Automated Black-Box Web Application Vulnerability Testing", in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, Berkeley, USA, May 16-19, pp. 332-345, 2010.
[4] P. E. Black, E. Fong, V. Okun, R. Gaucher, "Software Assurance Tools: Web Application Security Scanner functional Specification Version 1.0", *U.S. Department of Commerce National Institute of Standards and Technology*, 2008.
[5] Burp Suite. http://portswigger.net/burp/.
[6] E. Bursztein, M. Martin, J. C. Mitchell, "Text-based CAPTCHA Strengths and Weaknesses", *ACM Computer and Communication Security*, Chicago, USA, October 17-21, 2011.
[7] M. J. Cafarella, A. Halevy, J. Madhavan, "Structured Data on the Web", *Communications of the ACM*, Vol. 54 No. 2, pp. 72-79, 2011.
[8] A. Doupé, M. Cova, G. Vigna, "Why Johnny Can't Pentest: an Analysis of Black-box Web Vulnerability Scanners", *in Proceedings of the 17th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Bonn, Germany, July 8-9, pp. 111-131, 2010.
[9] H. Gonzalez, A. Y. Halevy, C. S. Jensen, A. Langen, J. Madhavan, R. Shapley, W. Shen, J. Goldberg-Kidon, "Google Fusion Tables: Web-centered Data Management and Collaboration", *in Proceedings of the International Conference on Management of Data*, Indianapolis, USA, June 6-11, pp. 1061-1066, 2010.
[10] HP Web Inspect; https://www.fortify.com/products/Web_inspect.html.
[11] Y. Huang, S. Huang, T. Lin, C. Tsai, "Web Application Security Assessment by Fault Injection and Behavior Monitoring", *in Proceedings of the 12th International Conference on World Wide Web*, Budapest, Hungary, pp. 20-24, May 2003.
[12] J. Madhavan, D. Ko, L. Lucja Kot, V. Ganapathy, A. Rasmussen, A.Y. Halevy, "Google's Deep Web crawl", *in Proceedings of the VLDB Endowment*, Vol. 1, No. 2, pp. 1241-1252, 2008.
[13] Magento. http://www.magentocommerce.com/.
[14] Metasploit. http://www.metasploit.com/.
[15] Opencart. http://www.opencart.com/.
[16] OWASP. https://www.owasp.org/.
[17] Synonymlab. http://www.synonymlab.com.
[18] A. Wright, "Searching the deep Web", *Communications of the ACM*, Vol. 51, No. 10, pp. 14-15, 2008.
[19] X. Xiang Peisu, H. Qinzhen, T. Ke, "A Framework of Deep Web Crawler", *27th Chinese Control Conference*, Kunming, China, July 16-18, 2008.
[20] L. Yujian, L. Bo. "A Normalized Levenshtein Distance Metric", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 29, No. 6, 2007.
[21] WolframAlpha. http://www.wolframalpha.com/.