Detección automatizada de desfiguraciones en un sitio web mediante una solución basada en Software Libre

Paul F. Bernal B.[1], Gustavo X. Chafla A.[2], Ernesto Pérez E.[3]

[1] Ingeniero especialista del CSIRT-CEDIA, Consorcio Ecuatoriano para el Desarrollo de Internet Avanzado, La Condamine 12-109 sector Subida del Vado, Cuenca, Ecuador, EC010150.

[2] Docente Coordinador de Maestría, Pontifica Universidad Católica del Ecuador, Av. 12 de Octubre 1076 y Roca , Quito, Ecuador, EC17012184.

[3] Docente de la FISEI, Universidad Técnica de Ambato, Ave. de los Chasquis y Río Payamino, Ambato, Ecuador, EC180103.

Autores para correspondencia: paul.bernal@cedia.org.ec, gxchafla@puce.edu.ec, ernesto.perez@uta.edu.ec

Fecha de recepción: 21 de septiembre de 2014.- Fecha de aceptación: 17 de octubre de 2014.

RESUMEN

La desfiguración de sitios web es uno de los ataques más populares hoy en día y se basan en realizar un cambio en el código HTML de una página web, que se presenta como un cambio visual en la imagen del mismo. En el presente trabajo se propone desarrollar un sistema mediante el cual se puedan detectar y/o reportar dichos cambios de una forma temprana y automatizada. El Software Libre (SL) ofrece la posibilidad de crear soluciones y es el marco sobre el cual se propone una solución.

Palabras clave: Software libre, desfiguración, Linux, seguridad, CSIRT, desarrollo, sitio web.

ABSTRACT

Web site defacements are one of the most popular attacks nowadays and rely on making a change in a web site's HTML code, which is presented as a visual change on the site. In this paper it is proposed to develop a system for detecting and/or reporting such changes in an early and automated way. Free Libre Open Source Software (FLOSS) offers the ability to create solutions and is the framework on which a solution is proposed.

Keywords: Free software, defacement, Linux, security, CSIRT, develop, website.

1. INTRODUCCIÓN

A mediados del 2013, el Consorcio Ecuatoriano para el Desarrollo de Internet Avanzado (CEDIA) emprendió la Fase II del proyecto de implementación de un CSIRT (Equipo de Respuesta ante Inicidentes de Seguridad Informática por sus siglas en Inglés), el cual se encarga de gestionar los eventos relacionados con la seguridad de las redes y sistemas de sus miembros, que son mayormente Universidades (Pérez, 2014). Una de las iniciativas principales fue la de diseñar y construir un sistema de recepción, procesamiento y envío de alertas de seguridad, mismo que entró en funcionamiento en el último trimestre de 2013. Este sistema actualmente procesa reportes de diferentes fuentes (feeds) conocidas por los CSIRT como son ShadowServer, Xroxy, HideMyAss, Zone-H, CertBR, CleanMX y SpamCop, que envían periódicamente el detalle de los incidentes relacionados a diferentes ámbitos, entre ellos: Proxies abiertos (openProxies), netbots, open resolvers, spam, y desfiguraciones (defacements).

De esta última categoría, Zone-H ha venido reportando incidentes al menos desde 2001, con un total a mediados de 2013 de casi 8 millones y medio de casos (Kovacs s.f.). Durante ese tiempo, se ha mostrado una clara tendencia anual de crecimiento de estos eventos (Zone-H.com s.f.) como se puede observar en la Fig.1, en los últimos 4 años superan el millón de ataques por año, excepto en 2014 que es el año en curso.

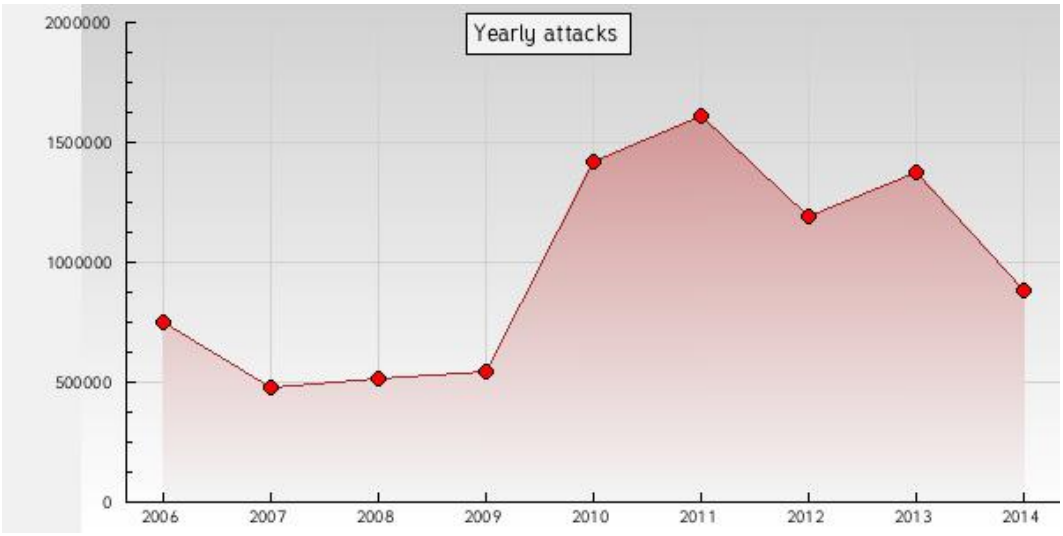

Figura 1. Ataques de desfiguración por año según Zone-H.

La misma tendencia se mantiene mes a mes durante lo que va del 2014, sin bajar de los 50 mil ataques mensuales, excepto en octubre que es el mes en curso, según se puede evidenciar en la Fig. 2.

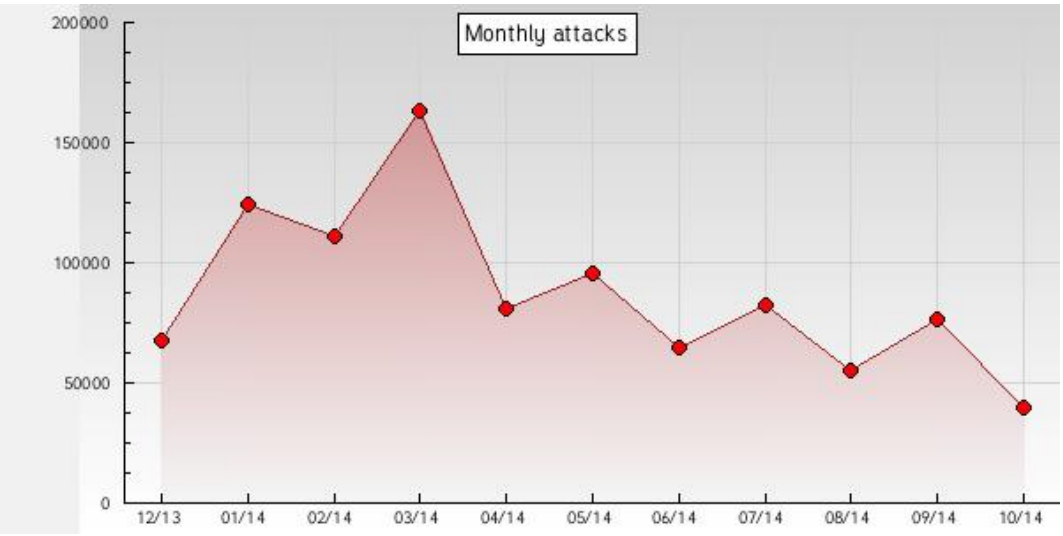

Figura 2. Ataques de desfiguración mensuales en 2014 según Zone-H.

Otro aspecto es la permanencia o duración del ataque: cuánto tiempo pasa entre que el sitio es desfigurado y esta situación se corrige. En casi la mitad de los sitios atacados, los efectos de estos ataques suelen permanecer por un lapso considerable de tiempo, con un 43% de los ataques durando al menos una semana y más del 37% permaneciendo luego de la segunda semana (Bartoli *et al*., 2009).

Este parámetro no necesariamente es alto debido al tiempo de reacción o el tiempo que le toma al equipo técnico solucionar el inconveniente, sino más bien tiene que ver con el lapso entre que el sitio es desfigurado y el responsable se percata del problema o pueda ser alertado del mismo por terceras partes.

Un medio que permita una detección temprana y automatizada de cambios a un sitio web, sería una forma efectiva de minimizar estos tiempos y sus consecuencias.

2. EL PROCESO

Para realizar una detección automatizada de cambios a un sitio web, se pueden abordar varias aproximaciones al problema, pero si se nota que independientemente de la tecnología usada para construir y distribuir un sitio web, lo que siempre se entrega al usuario es código HTML y lo que el usuario espera ver es la representación gráfica de ése código en el navegador web, entonces se comprende que lo más natural sería enfocar el problema hacia esas dos perspectivas.

Cabe aclarar que al referirse a código HTML se habla de todo el código consitutivo de una página web, dentro del cual es común encontrar, entre otros, código incrustado o enlazado de hojas de estilo (CSS) y JavaScript (JS), que cada vez más se usan para mejorar la visualización y funcionalidades en los sitios. Pero hay que tener claro que incluso para estas invocaciones o incrustaciones, se requiere de una etiqueta HTML (<style> y <script> respectivamente en los casos mencionados), por lo cual se habla de código HTML para contemplar todo el código a ser capturado y analizado.

2.1. El sitio web como código HTML

La primera forma propuesta para detectar cambios en un sitio web, sería el análisis de cambios a nivel del propio código HTML que se obtiene por parte del navegador web ante cada petición de un sitio que hace el usuario. En este sentido se requiere dividir la labor en dos fases: la de captura/obtención del código HTML y la del análisis de cambios entre dos versiones obtenidas en tiempos diferentes.

2.2. El sitio web como la representación gráfica del código

Considerando que HTML es un lenguaje de marcado y por la naturaleza misma del mismo, es frecuente que cambios grandes en el código produzcan resultados visuales sutiles; o que cambios pequeños en código produzcan cambios sustanciales en cómo se visualiza un sitio; es importante que se complemente el análisis de cambios en el código, con un analizador de cambios a nivel gráfico del sitio. Igualmente se divide la labor en las mismas anteriores dos fases: la obtención de una instantánea (snapshot) de cómo se refleja gráficamente un sitio web y el análisis propiamente de los cambios que pudieron producirse entre dos instantáneas tomadas en momentos separados.

3. CRITERIOS DE PRESELECCIÓN DE HERRAMIENTAS

Del universo de posibles herramientas que podrían usarse para el estudio y potencial aplicación en el proyecto, era necesario realizar una selección o filtrado previos, que permitieran obtener rápidamente un subconjunto a ser analizado y del cual extraer la herramienta más apropiada o incluso una combinación de ellas. Las siguientes condiciones han sido establecidas como filtros previos, en base principalmente a las características del proyecto que está orientado a una solución aplicable bajo las premisas del SL, la mantenibilidad, la automatización y necesidades de integración como módulo en el sistema de alertas de seguridad del CSIRT-CEDIA:

- La herramienta debe distribuirse libremente bajo una licencia de SL.
- La herramienta puede o no tener costo, pero se preferirá una gratuita si comparte características similares con otra.
- La herramienta debe ser un proyecto activo y mantenido de forma razonablemente periódica.
- Tendrán preferencia aquellas que se encuentren en un estado maduro/estable de desarrollo.
- La herramienta deberá correr sobre Bash scripting o al menos ofrecer una interface/API de programación que la vuelva accesible desde Bash.

- Las herramientas que pertenezcan, sean parte o estén disponibles en repositorios para una o varias distribuciones de Linux, cumplirán por lo mismo la mayoría de las premisas previas y por ende serán más opcionadas.
- Finalmente, la herramienta debe poder ser usada a través de línea de comandos (CLI), dado que se requiere un grado de automatización en su uso, una interfaz gráfica no nos será de utilidad.

4. CAPTURA DEL CÓDIGO HTML

Del análisis previo resultante de la aplicación de los criterios del punto III, las herramientas que pueden usarse para captura de código HTML de un sitio/página web son:

- lftp (LFTP s.f.),
- httrack (HTTrack s.f.),
- cURL (cURL s.f.),
- wget (Wget s.f.),
- aria2 (Aria2 s.f.) y
- uget (uGet s.f.).

De entre ellas, cURL y wget son las más conocidas. Tradicionalmente wget ha sido la herramienta más opcionada para éstas tareas, aunque últimamente cURL está tomando dicho lugar, principalmente por las ventajas de rendimiento y seguridad (Haxx s.f.). Sin embargo, a pesar de que cURL puede hacerlo -aunque no de una manera tan simplificada-, wget dispone de la opción de descargar todo un sitio en un solo archivo monolítico, con el uso combinado de las opciones -p y -O:

```
$ wget -p www.elsitioweb.com -O elsitioweb.out
```

Con lo cual se descarga todo el código HTML del archivo de inicio del sitio web, junto con todos sus archivos relacionados -como CSS y JS-, consolidados en un solo archivo llamado elsitioweb.out, con lo que se puede realizar un análisis más profundo del código constitutivo del mismo. Será necesario también automatizar la descarga periódica para una lista de sitios a monitorear, lo cual se puede realizar implementando una tarea programada.

5. ANÁLISIS DE CAMBIOS EN CÓDIGO HTML

El análisis de cambios de código entre diferentes versiones de un mismo archivo es una tarea inherente a la programación: todo programador y todo proceso de programación ha requerido históricamente de los medios para poder establecer rápida y fácilmente los cambios provocados entre una versión actual y una previa de un archivo de código fuente. Por esto mismo, las herramientas de diferenciación han ido evolucionando conjuntamente con las de programación y hoy en día es común verlas integradas en los entornos de programación. Pero esta evolución ha hecho que se vuelvan bastante visuales y difícilmente entreguen resultados numéricos significativos de cambio, al menos para el propósito de detección automatizada.

De esta variedad de herramientas que analizan y reportan los cambios de código, la mayoría lo hace de forma gráfica, algo no tan adecuado para los requerimientos del proyecto. El hecho es que para los efectos del proyecto se requiere detectarlos, pero principalmente obtener valores numéricos de cambio; idealmente un porcentaje de qué tan diferente es la nueva versión respecto a la anterior. El trabajo a realizarse en este sentido deberá estar orientado a eso, en lo cual justamente se fundamenta el proyecto. Adicionalmente, deberá ser una herramienta que permita el uso e interacción programáticas, de tal forma que los procesos de análisis y obtención/cálculo de resultados sean susceptibles de automatización.

Se propone analizar la posibilidad de llevar a cabo un proceso mediante una combinación de herramientas, pues no todas realizan el mismo tipo de trabajo. Unas realizan un análisis a nivel de líneas, otras lo hacen a nivel de palabras y también a nivel de caracteres. El grado de cambios en un mismo escenario variará sustancialmente de acuerdo al tipo de análisis que se haga.

Por la misma naturaleza de HTML, no se puede garantizar un porcentaje alto de efectividad en los valores de cambio obtenidos para todos los casos. Para ciertos sitios será más aproximado un análisis por líneas, para otros uno de palabras o tags. Por ello es bueno considerar una combinación que arroje más elementos de juicio al administrador.

Un ejemplo sencillo sería comparar las siguientes dos versiones de un archivo con una única línea de código que cambió sobre un total de seis líneas:

1. Versión 1 (versión previa):

```
<html>
 <head><title>Prueba</title></head>
 <body>
  <h1 id="titulo">El Sitio Web</h1>
 </body>
</html>
```

2. Versión 2 (versión actual):

```
<html>
 <head><title>Prueba</title></head>
 <body>
  <h2 id="titulo">El Sitio Web</h2>
 </body>
</html>
```

Se podrían elementalmente obtener tres resultados de tres comparativas diferentes:

- *Comparativa por líneas.* Siendo la única línea que cambió sobre un total de 6 en el archivo original, el análisis es sencillo:

$$(1 / 6) * 100 \cong 17\%$$

- *Comparativa por palabras (tags).* En este caso se detectan 2 palabras o etiquetas cambiadas, sobre un total de 15 palabras/etiquetas, indicaría que los cambios se pueden calcular así:

$$(2 / 15) * 100 \cong 13\%$$

- *Comparativa por caracteres.* Y el porcentaje de los cambios a nivel de caracteres se obtendría de relacionar los 2 caracteres modificados con el total de 94 caracteres significativos -incluyendo espacios y descartando los de indentación- en el archivo original:

$$(2 / 94) * 100 \cong 2\%$$

Los resultados varían significativamente entre sí, en función del tipo de análisis realizado y del propio código HTML. Considerando que el resultado final del código visto en un navegador será muy similar, de los tres valores en este caso, el menos preciso es el primero porque cuenta la línea

completa, sin importar qué tanto ha cambiado la misma internamente. Pero igualmente un cambio de unos pocos caracteres en HTML puede generar grandes cambios en lo que finalmente se visualiza, por lo cual es necesario contrastar estos resultados con un análisis gráfico de los resultados visuales del código.

6. CAPTURA DE LA IMAGEN DEL SITIO

Al igual que en el caso de la captura de código, se seleccionó el siguiente subconjunto de herramientas acorde a las consideraciones planteadas:

- ImageMagick (ImageMagick s.f.),
- CutyCapt (CutyCapt s.f.),
- khtml2png (khtml2png s.f.),
- PyWebShot (PyWebShot s.f.),
- python-webkit2png (python-webkit2png s.f.) y
- PanthomJS (PhantomJS s.f.).

De las cuales destacaron ImageMagick y PhantomJS, la primera por ser una herramienta tradicional en la manipulación de imágenes, es muy eficiente y dispone de una muy amplia gama de capacidades y opciones. Sin embargo, a diferencia de PhantomJS, sólo es capaz de capturar un sitio web, a través de un elemento externo que lo renderice y además requiere que se le especifique el área de captura, algo que para un sistema automatizado podría resultar ser un inconveniente. Por su lado PhantomJS hace el trabajo de forma eficiente y simplificada, pero, al estar basada en JavaScript (de ahí el JS en el nombre), requiere que se le alimente con un archivo .js que contenga las especificaciones del sitio (url) a capturarse, así como del archivo resultante:

```
var page = require('webpage').create();
page.open('http://www.google.com.ec', function ()
{
   page.render('google.png');
   phantom.exit();
});
```

Este archivo debe ser provisto como parámetro a PhantomJS para su procesamiento. Básicamente renderiza el contenido del sitio, de forma similar a como lo haría un navegador web común, y realiza la captura sin necesidad de un navegador ni tampoco de disponer de una interface gráfica:

```
$ phantomjs google.js
```

Con lo cual se genera un archivo llamado google.png con la imagen capturada de cómo se visualizaría el sitio web en ese momento en un navegador estándar (ver Fig. 3).

Figura 3. Captura de google.com.ec obtenida con PhantomJS**.**

7. ANÁLISIS DE CAMBIOS EN IMÁGENES

Se puede decir que es fácil detectar cambios entre una captura anterior y una actual de un mismo sitio web, siempre y cuando esta comparativa se realice con el ojo y cerebro humanos, el resultado no tomará más que unos instantes. Pero para el caso propuesto tenemos problemas inherentes: 1.- no es un proceso muy eficiente pues requeriría de una persona dedicada a esta tarea, 2.- no es sencillo obtener un valor numérico de cambios, dado que sería el resultado de un análisis apreciativo del observador, y 3.- no es un proceso susceptible de automatización.

Por otro lado, realizar el análisis digital de los cambios a nivel de imagen entre dos versiones de un mismo sitio requiere de herramientas que hacen uso de complejos y diversos algoritmos matemáticos. Afortunadamente estas herramientas existen y están disponibles, encapsulando esta complejidad y simplificando su uso.

Al tratarse de una tarea mucho más compleja que la del análisis de cambios en código, el consumo de recursos será un factor importante a considerarse, así como la relevancia de los resultados pues la mayoría ofrecerá gráficos como salida, en lugar de valores numéricos. En este caso, ambos son importantes, aunque para la automatización es de interés el resultado numérico. Se reduce bastante el universo de candidatos al buscar aquellos que idealmente ofrezcan los dos tipos de resultado, pues a diferencia del análisis a nivel de código -que permite una inferencia de valores en base al código-, no es tan sencillo inferir resultados numéricos de los resultados gráficos.

También hay variantes por si se realiza un análisis sobre imágenes a todo color o en escala de grises (GS por sus siglas en inglés). Las de color deben ser descompuestas en las diferentes capas de color constituyentes para ser analizadas independientemente y se obtienen resultados independientes por capa. En el caso de las imágenes en GS, se pueden analizar directamente en una sola capa. Esto tiene implicaciones no solo en los resultados, sino también en el rendimiento. Por ello lo más aceptado es invertir en procesamiento previo para una conversión a GS, con lo cual además se minimizan posibles desviaciones porque al combinar las capas de color, los cambios comunes en ellas tienden a consolidarse.

De ésta forma el problema se puede reducir a usar una herramienta que simplemente entregue un resultado en función del número de píxeles que difieren entre la versión anterior y la actual de la captura de un mismo sitio. Llevar ese valor a un porcentaje es razonablemente simple, requiriendo solamente de realizar un cálculo o usar una herramienta complementaria para obtener el número total de pixeles y poder contrastarlos con el número de modificados.

Otro tema a tomarse en cuenta es el hecho de que no son pocos los sitios que cambian frecuentemente en cuanto a la longitud vertical total que muestran. Los blogs son un ejemplo actual. Esto introduce el problema de comparar dos imágenes de diferente tamaño, lo cual por un lado puede ser sencillamente inmanejable por algunas herramientas y por otro lado provocarían valores de cambio no reales pues se marcaría como desviación ésta variación de tamaño. Esto puede solventarse de forma relativamente simple, también con un pre procesamiento de la imagen, para achicar la más grande al tamaño de la más pequeña, requiriendo además un proceso adicional de cálculo de tamaños.

Finalmente, será importante poder obtener una imagen resultante de los cambios, algo que visualmente muestre los cambios entre una y otra versión.

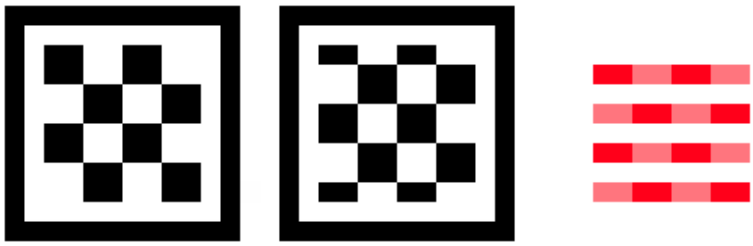

Figura 4. Ejemplo de imagen (12px*12px) anterior, actual y resultado de diferencias.

Esta tercera imagen intencionalmente coloreada en tonos rojos, es el resultado de contrastar la primera con la segunda y además representa el número de pixeles modificados, 32 en este caso, sobre un total de 144 en una imagen de 12x12 pixeles. De ésta forma se puede calcular los cambios con:

$$(32 / 144) * 100 \cong 22\%$$

Este valor numérico es esencial para la automatización y eventual emisión de reportes al alcanzar/superar cierto límite establecido para un sitio, pero el resultado visual permite una comprensión más directa de los cambios por parte del administrador.

8. OBTENIENDO RESULTADOS

Con las herramientas escogidas se desarrolló un prototipo de sistema de detección de cambios, que realiza la captura de código e imagen de un sitio. Esta captura se realiza diariamente a través de una tarea programada. De esta forma se puede analizar los cambios, contrastando las capturas del día actual, con las del día anterior.

Primero se creó un script que permita capturar tanto el código como la imagen (screenshot) de un sitio <http://paulbernal.com/ticec14/getSite.sh>, que se puede usar de forma muy simple:

```
$ ./getSite.sh www.google.com
```

Lo cual genera una carpeta que lleva por nombre la fecha actual en el formato 'YYYYDDMM' en donde se almacenan los archivos de la captura:

```
$ ls 20141023/
www.google.com.html www.google.com.out www.google.com.png
```

El archivo .out contiene la captura con wget y el modificador -O, el archivo .html se obtiene sin el modificador -O y el archivo .png contiene la captura de la imagen del sitio.

Ahora se pueden realizar comparaciones de código e imagen usando las herramientas seleccionadas y algo de procesamiento adicional. Se realizaron capturas de dos sitios, el primero (Sitio1) que no ha sufrido desfiguración y el segundo (Sitio2) que sí ha sido desfigurado, para así poder obtener resultados de ambos escenarios.

8.1. Comparativa de código

Para medir los cambios detectados en código se realizó un script <http://paulbernal.com/ticec14/getCodeChanges.sh> que usa comm (GNU s.f.) para encontrar las diferencias en función de contrastar las líneas cambiadas con las totales:

```
$ /getCodeChanges.sh 20141022/www.sitio1.html 20141023/www.sitio1.html
6.8512
```

El valor de cambio calculado entre un día y otro, a nivel de código para el Sitio1 es de 6.8512%. Sin embargo para el Sitio2 que fue desfigurado, el cambio arroja un apropiado 100%:

```
$ /getCodeChanges.sh 20141022/www.sitio2.html 20141023/www.sitio2.html
100.0000
```

8.2. Comparativa de imágenes

Al igual que con el análisis de código, se desarrolló un script <http://paulbernal.com/ticec14/getImageChanges.sh> que realice las tareas de obtención y cálculo de diferencias entre las capturas de un día y el anterior. Se usa ImageMagick (ImageMagick s.f.) para realizar las tareas de comparación. El cálculo del valor de cambio se realiza contrastando el número de píxeles modificados en relación al total de la imagen original:

```
$ /getImageChanges.sh 20141022/www.sitio1.png \
20141023/www.sitio1.png
9.2484
```

Para el Sitio1 el valor de cambio a nivel del imagen es de 9.2484% que es consistente con el valor obtenido de cambio en código. Sin embargo, el valor obtenido para el Sitio2, el valor de cambio en imagen es de sólo 41.6695% que representa menos de la mitad de la imagen:

```
$ /getImageChanges.sh 20141022/www.sitio2.png \
20141023/www.sitio2.png
41.6695
```

Este resultado puede ser considerando inconsistente sabiendo que el sitio fue efectivamente desfigurado y obtuvo 100% en el análisis de código. Entonces se decidió generar la imagen resultante de la comparativa para poder tener una idea visual de los cambios. Primero se identifican los tamaños de las imágenes -que típicamente son diferentes en un defacement- y se requieren para homologar los tamaños, requisito para la comparativa imagenográfica:

```
$ identify 20141022/www.sitio2.png|awk {'print $4'}
916x483+0+0
$ identify 20141023/www.sitio2.png|awk {'print $4'}
400x483+0+0
```

Luego hay que efectivamente redimensionar la imagen más pequeña al tamaño de la más grande:

```
$ convert 20141023/www.sitio2.png -resize \
916x483+0+0! 20141023/www.sitio2-tmp.png
```

Finalmente realizar la comparativa a nivel de la imagen para obtener la imagen resultante que muestre las diferencias:

```
$ compare -dissimilarity-threshold 1 \
20141022/www.sitio2.png \
20141023/www.sitio2-tmp.png \
20141023/www.sitio2-changes.png 2>&1
```

El archivo resultante muestra en tonos rojizos los píxeles que han cambiado entre la imagen anterior y al actual, dejando en grises o marca de agua aquellos que son comunes, tal cual se puede ver en la Fig. 5.

Figura 5. Imágenes: original, desfigurada y resultante de la comparativa del Sitio2.

Visualmente los cambios cubren más allá del 41.67% obtenido del cálculo numérico y esto se explica sobre la base de que el cálculo de cambios utiliza algoritmos diferentes para detectar los pixeles cambiados/comunes que los que se usan para la generación de la imagen de coincidencias/diferencias.

9. CONCLUSIONES

Si bien existen y están disponibles las herramientas para captura y análisis de código e imagen de un sitio web, éstas no representan usa solución directa a la necesidad planteada de la detección automatizada de cambios en un sitio web.

El uso de las herramientas de SL seleccionadas para solucionar la necesidad de detección de cambios en un sitio web, contribuyen a la creación de un sistema que monitoree y reporte variaciones signicativas a los administradores y minimizar el tiempo de reacción ante tales eventos.

Tanto para código como para imágenes, se requiere estudiar las diferentes formas de análisis de cambios y determinar la forma o combinación de ellas que genere los resultados más acertados, consistentes y aplicables a la mayor cantidad de escenarios posibles. En ambos casos son claramente necesarios procesos de procesamiento pre y post análisis.

- En el caso del código: organizarlo o preformatearlo -eliminando variaciones no significativas- ayudará a mejorar la precisión de los resultados; así como el cálculo numérico posterior derivado del número de cambios reportado.
- En el caso del análisis gráfico: la homologación de tamaños y/o la experimentación con diversos algoritmos de cálculo ayudarán a mejorar la consistencia de los resultados.

El desarrollo de soluciones basadas en SL es viable y contribuye al cambio hacia una filosofía de investigación y producción de tecnología, evitando limitarse únicamente al uso/consumo de la misma.

AGRADECIMIENTOS

Agradecemos la disposición de CEDIA por permitirnos desarrollar la investigación usando su infraestructura y ofrecernos las facilidades necesarias, así como la visión de permitirnos producir nuestras propias herramientas basadas en SL.

REFERENCIAS

Aria2. *The next generation download utility.* s.f. Disponible en http://aria2.sourceforge.net.

Bartoli, A, G Davanzo, E. Medvet, 2009. *The reaction time to Web site defacements*. IEEE Internet Computing.

cURL. *A utility for getting files from remote servers.* s.f. Disponible en http://curl.haxx.se/.

CutyCapt. *A Qt WebKit Web Page Rendering Capture Utility.* s.f. Disponible en http://cutycapt.sourceforge.net/.

GNU. *7.4 comm: Compare two sorted files line by line.* s.f. Disponible en http://www.gnu.org/software/coreutils/manual/html_node/comm-invocation.html.

Haxx. *Compare cURL Features with Other Download Tools.* s.f. Disponible en http://curl.haxx.se/docs/comparison-table.html.

HTTrack. *Website Copier.* s.f. Disponible en http://www.httrack.com/.

ImageMagick. *Convert, Edit, And Compose Images.* s.f. Disponible en http://www.imagemagick.org/.

khtml2png. *Make screenshots from webpages.* s.f. Disponible en http://khtml2png.sourceforge.net/.

Kovacs, E. *Over 665,000 Defacements Submitted to Zone-H.org in 2013.* s.f. Disponible en http://news.softpedia.com/news/Over-665-000-Defacements-Submitted-to-Zone-H-org-in-2013-358614.shtml.

LFTP. *Sophisticated file transfer program.* s.f. Disponible en http://lftp.yar.ru/.

Pérez, E., 2013. *Sistema de procesamiento y reporte de alertas de seguridad para el CSIRT-CEDIA.* Convención Internacional de Ciencias Técnicas, Santiago de Cuba, 111-222.

PhantomJS. *Headless WebKit scriptable with a JavaScript API.* s.f. Disponible en http://phantomjs.org.

Python-webkit2png. *Python script that takes screenshots (browsershots) using webkit.* s.f. Disponible en https://github.com/AdamN/python-webkit2png/.

PyWebShot. *Command line webpage screenshot and thubnail generator.* s.f. Disponible en https://github.com/coderholic/PyWebShot.

uGet. *Open Source Download Manager.* s.f. Disponible en http://ugetdm.com/.

Wget. *A utility for retrieving files using the HTTP or FTP protocols.* s.f. Disponible en http://www.gnu.org/software/wget/.

Zone-H.com. *Yearly & Monthly & Daily attacks Stats.* s.f. Disponible enhttp://www.zone-h.org/stats/ymd.