STEGOSPLOIT

SAUMIL SHAH

HACK.LU 2015
STEGOSPLOIT

BROWSER EXPLOITS USING ONLY IMAGES

SAUMIL SHAH
HACK.LU 2015
Saumil Shah
CEO, Net-Square

hacker, trainer, speaker, author, photographer educating, entertaining and exasperating audiences since 1999.
Why Stegosity Isn’t An Exploit

Internet pictures can hide code that leaves you open to hacks (update: criticism)

Edit: I screwed up. The “false claims” I’m attacking in this piece are about a new (unreleased) version of Stegosity which I wasn’t aware of. I’ll leave this article as it and post a new one soon. Sorry for not researching this well enough, and thanks for understanding my mistake.

Stop Saying Stegosity Isn’t An Exploit
UNFORTUNATELY, NO ONE CAN BE TOLD... WHAT STEGOSPLOIT IS
Stegosploit is...

not a 0-day attack with a cute logo
not exploit code hidden in EXIF
not a PHP/ASP webshell
not a new XSS vector

Stegosploit is ... 

"Browser Exploits Delivered as Pictures."
"A good exploit is one that is delivered with style"

 Protocol-spanning, syntax-based generalized exploit methodologies are the new black.

Saumil Shah @therealsaumil

#stegosplot tools will be released in the next PoC||GTFO. The only fitting publication for the purpose. cc @travisgoodspeed @angealbertini
Hacking with pictures, in style!

• ONLY image files – on network and disk.
• Exploit hidden in pixels.
  – no visible aberration or distortion.
• Image "auto runs" upon load.
  – decoder code bundled WITH the image.
• Exploit automatically decoded and triggered.
• ...all with just ONE IMAGE.
Steganography
Polyglots

Two or more data formats in a single container...

...that co-exist happily without breaking each other's spec or syntax.
Stegospelting a browser exploit

Case study: CVE-2014-0282
- IE CInput Use-After-Free
- hidden in a JPG

Case study: CVE-2013-1690
- FF onreadystatechange UAF
- hidden in a PNG
The Stegosploit Toolkit

STEGANOGRAPHY TOOLS
- image_layer_analysis.html
- iterative_encoding.html
- image_decoder.html

- analyse an image's bit layers
- steganographic encoder
- test for any encoding errors

POLYGLOT TOOLS
- imajs_jpg.pl
- imajs_png.pl

- make a JPG+HTML+JS polyglot
- make a PNG+HTML+JS polyglot

EXPLOITS
- exploits.js
- cve_2014_0282.template
- decode_pixels.js

- collection of browser exploits
- exploit HTML template
- JS Steganography decoder
Step I.

Hiding the Exploit Code in the Image
Hiding an Exploit in an Image

- Simple steganography techniques.
- Encode exploit code bitstream into lesser significant bits of RGB values.
- Spread the pixels around e.g. 4x4 grid.
Hiding an Exploit in an Image

IE Use-After-Free CVE-2014-0282
The "Bit Layer" View

1 pixel = 8 bits (grayscale)
The "Bit Layer" View
Encoding at Bit Layer 7

Exploit code converted to bitstream.

Pixel bits of layer 7 are overwritten with exploit bitstream.
Encoding data at bit layer 7

Significant visual aberration
Encoding at Bit Layer 2

Exploit code converted to bitstream.

Pixel bits of layer 2 are overwritten with exploit bitstream.
Encoding data at bit layer 2

No perceptible visual aberration
Encoding on JPG

- JPG – lossy compression.
- Pixels may be approximated to their nearest neighbours.
- Overcoming lossy compression by ITERATIVE ENCODING.
- Can't go too deep down the bit layers.
- IE's JPG encoder is terrible!
- Browser specific JPG quirks.
Encoding on PNG

• Lossless compression.
• Can encode at bit layer 0.
  – minimum visual distortion.
• Independent of browser library implementation.
• Single pass encoding.

• JPG is still more popular than PNG!
Step 2.

Decoding the encoded Pixel Data
HTML5 CANVAS to the rescue!

• Read image pixel data using JS.

• In-browser decoding of steganographically encoded images.
decode_pixels.js

L=2,C=3,G=3,a=[],x=y=0,z=1<<L,I=parseInt,S=String.fromCharCode;window.onload=
function(){P.onclick=function({V=document.createElement("canvas");k=P.parentNode;
k.insertBefore(V,P);W=V.width;H=V.height;m=V.getContext("2d");
m.drawImage(P,0,0);k.removeChild(P);m.getImageData(0,0,W,H).data;c=function(p,x,y)
{n=(y*w+x)*4;r=(p[n]&z)>>L;g=(p[n+1]&z)>>L;b=(p[n+2]&z)>>L;return S([r,g,b,r][C]+48]);
k=function(l){for(i=j=0;j<l*8;j++)a[i++]=c(m,x,y);x+=G;if(x>=W){x=0;y+=G}};k(6);
k(X(a))};try{CollectGarbage()catch(e){}setTimeout(new Function(X(a)),99)};function
X(c){s="";d=c.join(s);for(i=0;i<d.length;i+=8)s+=S(I(d.substr(i,8),2));return s}
Step 3.

Images that "Auto Run"
IMAJS – The Concept

<img> sees pixels
<script> sees code

#YourPointOfView

Holy
Sh**
Bipolar
Content!
IMAJS-JPG!

I ❤ JPG

JPG +HTML +JS +CSS

Hat tip: Michael Zalewski @lcamtuf
JPG Secret Sauce

shhh..
don’t tell anyone
# JPG Secret Sauce

<table>
<thead>
<tr>
<th>SOI</th>
<th>FF D8</th>
</tr>
</thead>
<tbody>
<tr>
<td>APP0</td>
<td>FF E0</td>
</tr>
<tr>
<td></td>
<td>length</td>
</tr>
<tr>
<td></td>
<td>J F I F \0</td>
</tr>
<tr>
<td></td>
<td>versn</td>
</tr>
<tr>
<td></td>
<td>U</td>
</tr>
<tr>
<td></td>
<td>Xres</td>
</tr>
<tr>
<td></td>
<td>Yres</td>
</tr>
<tr>
<td></td>
<td>H</td>
</tr>
<tr>
<td></td>
<td>V</td>
</tr>
<tr>
<td>DQT</td>
<td>FF DB</td>
</tr>
<tr>
<td></td>
<td>quantization tables</td>
</tr>
<tr>
<td>DQT</td>
<td>FF DB</td>
</tr>
<tr>
<td></td>
<td>quantization tables</td>
</tr>
<tr>
<td>SOF0</td>
<td>FF C0</td>
</tr>
<tr>
<td></td>
<td>start of frame</td>
</tr>
<tr>
<td>DHT</td>
<td>FF C4</td>
</tr>
<tr>
<td></td>
<td>Huffman tables</td>
</tr>
</tbody>
</table>
## JPG Secret Sauce

<table>
<thead>
<tr>
<th>SOI</th>
<th>SOI byte sequence: FF D8</th>
</tr>
</thead>
<tbody>
<tr>
<td>APP0</td>
<td>APP0 byte sequence: FF E0</td>
</tr>
<tr>
<td></td>
<td>length</td>
</tr>
<tr>
<td></td>
<td>J F I F \0</td>
</tr>
<tr>
<td></td>
<td>versn</td>
</tr>
<tr>
<td></td>
<td>U Xres</td>
</tr>
<tr>
<td></td>
<td>Yres</td>
</tr>
<tr>
<td></td>
<td>H V</td>
</tr>
<tr>
<td>DQT</td>
<td>DQT byte sequence: FF DB</td>
</tr>
<tr>
<td></td>
<td>quantization tables</td>
</tr>
<tr>
<td>DQT</td>
<td>FF DB</td>
</tr>
<tr>
<td>SOF0</td>
<td>SOF0 byte sequence: FF C0</td>
</tr>
<tr>
<td></td>
<td>start of frame</td>
</tr>
<tr>
<td>DHT</td>
<td>DHT byte sequence: FF C4</td>
</tr>
<tr>
<td></td>
<td>Huffman tables</td>
</tr>
</tbody>
</table>

*HTML random random random random random random...*

*random* `<head` random>` decoder script

*and other HTML stuff goes here...*

*<script type=text/undefined>`...*

*... more random data ...*
IMAJS-PNG!

I PNG

PNG +HTML +JS +CSS
PNG Secret Sauce - FourCC

<table>
<thead>
<tr>
<th>PNG Header</th>
<th>89 50 4E 47 0D 0A 1A 0A</th>
</tr>
</thead>
<tbody>
<tr>
<td>IHDR</td>
<td>length IHDR chunk data</td>
</tr>
<tr>
<td>IDAT chunk</td>
<td>length IDAT pixel data</td>
</tr>
<tr>
<td>IDAT chunk</td>
<td>length IDAT pixel data</td>
</tr>
<tr>
<td>IDAT chunk</td>
<td>length IDAT pixel data</td>
</tr>
<tr>
<td>IEND chunk</td>
<td>0 IEND CRC</td>
</tr>
</tbody>
</table>

www.fourcc.org
PNG Secret Sauce - FourCC

PNG Header

IHDR chunk data

extra tEXt chunk

length text _00<html random random ... random><head random> decoder script and other HTML stuff goes here...

<script type=text/undefined>... CRC

IDAT chunk

length IDAT pixel data CRC

IEND chunk

0 IEND CRC

Inspiration: http://daeken.com/superpacking-js-demos
Step 4.
The Finer Points of Package Delivery
A Few Browser Tricks...

Content Sniffing

Expires and Cache-Control

Clever CSS
# Content Sniffing

<table>
<thead>
<tr>
<th>Test description</th>
<th>MSIE6</th>
<th>MSIE7</th>
<th>MSIE8</th>
<th>FF2</th>
<th>FF3</th>
<th>Safari</th>
<th>Opera</th>
<th>Chrome</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is HTML sniffed when no Content-Type received?</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Content sniffing buffer size when no Content-Type seen</td>
<td>256 B</td>
<td>~∞</td>
<td>~∞</td>
<td>1 kB</td>
<td>1 kB</td>
<td>~130 kB</td>
<td>1 kB</td>
<td>~∞</td>
<td></td>
</tr>
<tr>
<td>Is HTML sniffed when a non-parseable Content-Type value received?</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Is HTML sniffed on application/octet-stream documents?</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Is HTML sniffed on application-binary documents?</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Is HTML sniffed on unknown/unknown (or application/unknown) documents?</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Is HTML sniffed on MIME types not known to browser?</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Is HTML sniffed on unknown MIME when .html, .xml, or .txt seen in URL parameters</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Is HTML sniffed on unknown MIME when .html, .xml, or .txt seen in URL path?</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Is HTML sniffed on text/plain documents (with or without file extension in URL)</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Is HTML sniffed on GIF served as image/jpeg?</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Is HTML sniffed on corrupted images?</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Content sniffing buffer size for second-guessing MIME type</td>
<td>256 B</td>
<td>256 B</td>
<td>256 B</td>
<td>n/a</td>
<td>n/a</td>
<td>~∞</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>May image/svg+xml document contain HTML xmlns payload?</td>
<td>(YES)</td>
<td>(YES)</td>
<td>(YES)</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>(YES)</td>
</tr>
<tr>
<td>HTTP error codes ignored when rendering sub-resources?</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>
Dive Into Cache

GET /stego.jpg

HTTP 200 OK
Expires: May 30 2015

GET /stego.jpg

o hai

o hai
PAYLOADS GO BACK IN TIME
I'M IN UR BASE

GET /lolcat.png
200 OK
Expires: 6 months

Exploit code encoded in image.
EVIL

AUG 2015

ATTACK TIMELINE

....KILLING UR DOODZ

GET /lolcat.png

Load from cache

Decoder script references image from cache.
SAFE

DEC 2015
<table>
<thead>
<tr>
<th>Anti-Virus</th>
<th>Update</th>
<th>Detected</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>VT Yara</td>
<td>PelID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VT Yara</td>
<td>Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VT Yara</td>
<td>Mobile</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VT Yara</td>
<td>Trojans</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVG</td>
<td>12.0.1794.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ClamAV</td>
<td>0.96.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comodo</td>
<td>1.0.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drweb</td>
<td>6.0.2.2 - linux</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESET</td>
<td>4.0.77</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F-Prot</td>
<td>4.6.5.141</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ikarus</td>
<td>1.3.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kaspersky</td>
<td>8.0.1-50</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PoC || GTFO 0x08

Stegosploit
Exploit Delivery via Steganography and Polyglots

by Saumil Shah - saumil@net-square.com, @therealsaumil
June 2015

TL;DR:
Stegosploit creates a new way to encode “drive-by” browser exploits and deliver them through image files. These payloads are undetectable using current means. This paper discusses two broad underlying techniques used for image based exploit delivery - Steganography and Polyglots. Drive-by browser exploits are steganographically encoded into JPEG and PNG images. The resultant image file is fused with HTML and Javascript decoder code, turning it into an HTML+Image polyglot. The polyglot looks and feels like an image, but is decoded and triggered in a victim’s browser when loaded.

The Stegosploit Toolkit v0.2, released in Issue 0x08 of PoC||GTFO, contains the tools necessary to test image based exploit delivery. A case study of a Use-After-Free memory corruption exploit (CVE-2014-0282) is presented with this paper demonstrating the Stegosploit technique.

1. Introduction
*A good exploit is one that is delivered with style* -- Saumil Shah

http://stegosploit.info
Conclusions - Offensive

• Lot of possibilities!
• Weird containers, weird encoding, weird obfuscation.
• Image attacks emerging "in the wild".
• CANVAS + CORS = spread the payloads.
• Not limited to just browsers.
• PDF+Flash / HTML+JS+FLASH (@angealbertini?)
Browsers and W3C - Wake Up!

BROWSERS
• Don't be afraid to "BREAK THE WEB".
• Reject content that does not conform to strict standards/specs.

W3C
• Establish STRICT parsing rules.
• Browser compliance and user-awareness is YOUR responsibility.
Conclusions - Defensive

• DFIR nightmare.
  – how far back does your window of inspection go?
• Can't rely on magic numbers, file extensions, file types.
• Quick "fix" – re-encode all images!
Greets!

@Level2LU
@lcamtuf
@angealbertini
@0x6D6172696F
PoC || GTFO crew

#HackLU CREW!
THANK.YU, HACK.LU!

Saumil Shah

@therealsaumil
saumil@net-square.com

Photography
flickr.com/saumil
www.spectral-lines.in