

Reverse engineering Heroes 3 - Basics

By Raistlin, put together by Jakub Grzana

Last update: 17.02.2021

Prerequisites:

- IDA Pro (interactive disassembler by Hex-Rays) which is NOT free program!
- IDA Database for Heroes 3 SoD [3.59 by GrayFace](#), [ERA_HD](#)
- Debugger OllyDbg [LINK](#)
- Russian SoD Executable [LINK](#) Password: GoogleDoesNotLikeExe

Note: IDA database for 3.59 and ERA_HD are NOT interchangeable. You must use 3.59 for 3.59. Although version for ERA is still useful, to somehow figure out what's going on, due to better documentation.

Introduction

This tutorial will explain only basics of using IDA on example of changing values hardcoded in game memory. It is just first step for real reverse engineering – but important one.

IDA is a disassembler for computer software which generates assembly language source code from machine-executable code IDA performs automatic code analysis, using cross-references between code sections, knowledge of parameters of API calls, and other information. However, IDA itself will not bring you so much, you need the database, that was created and updated by developers during last 20 years.

To start, get a copy of IDA Pro, download database and debugger. After installation of IDA, you can open database [Heroes3f.idb](#) just by double-clicking on it. IDA allows you to understand what is going on inside exec - it has named variables, functions etc.

Debugger can be used, among other things, to find precise addresses of constants you found in IDA. After installation, you must start instance of Debugger, then „open” executable of Heroes 3 SoD inside this debugger. Any SoD version should do, but i think WoG is based on russian version, so i encourage to use russian. *(i don't have any proofs for that tho)*

Here is proof that there're different versions of SoD exec – it is clearly visible in main menu, I don't know if the differences are meaningful though.

Nazwa	Heroes3.exe	Nazwa	Heroes3.exe
Rozmiar	2727936 bajtów (2664 KiB)	Rozmiar	2744320 bajtów (2680 KiB)
CRC64	75341A774075A370	CRC64	8D8DBC0B5ADA885E
Russian		Polish	

Basic assembler instructions

mov dest, source – copy value from source to dest
jmp – jumps to some place in code (pretty much like goto in C)
call name – call function with given name/address
push val – push value on stack
pop var – pop value from stack, save it in var

eax, ebx ... - registers of processor. You can store values or pointers in them

mov eax, ebx - copies value stored in ebx to eax

mov eax, [ebx] - copies value from memory to eax (ebx is pointer)

Note: since Heroes 3 is 32bit executable, registers are 32bit=4bytes aswell.

Usage of stack in functions:

(eax = 5, ebx = 8, esi = 0x446752)

push eax

push ebx

push esi

call 00678453

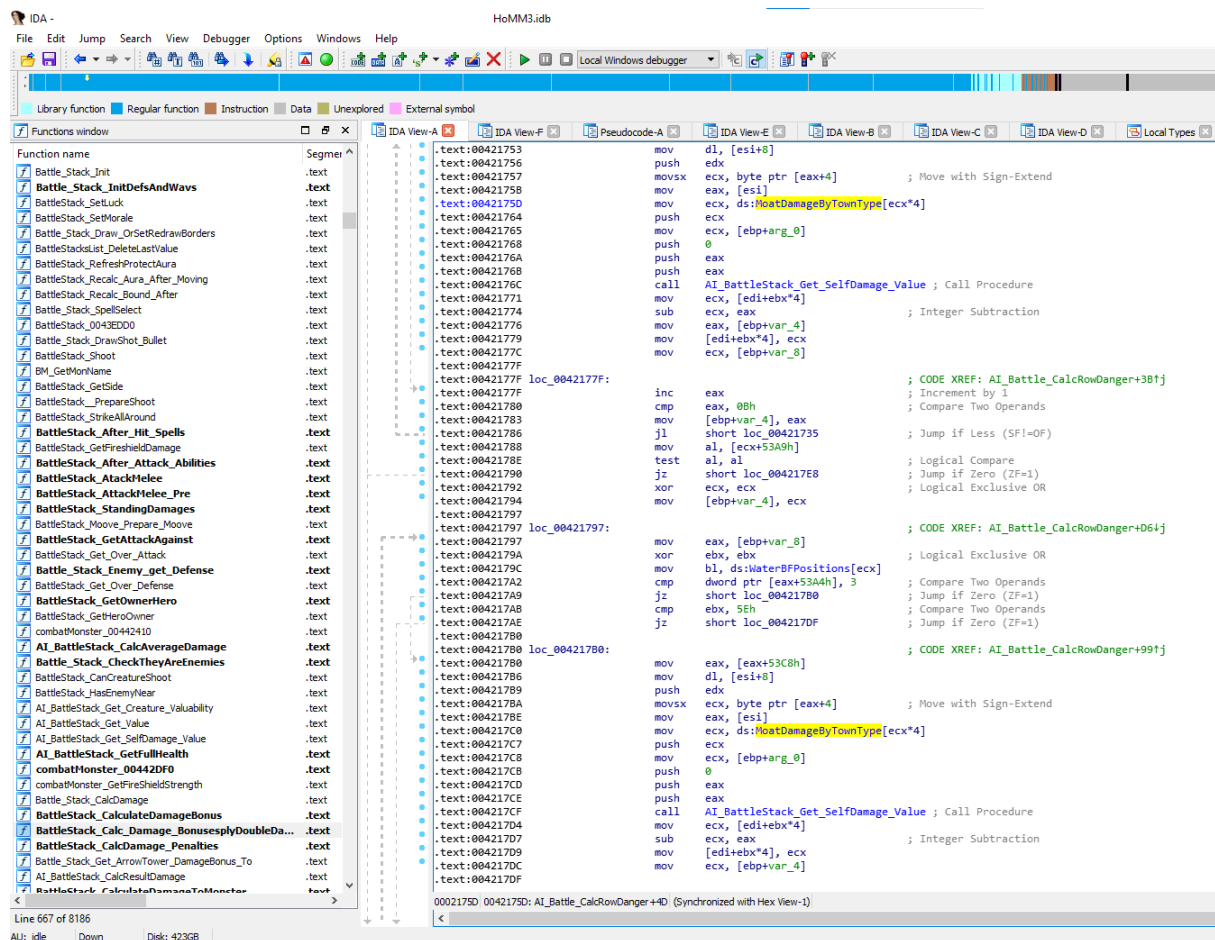
To get access to the arguments in the function use the register esp. The first argument is [esp], the second - [esp+4], the third - [esp+8]

If you need more informations, there're many tutorials on the internet, for example [this one](#).

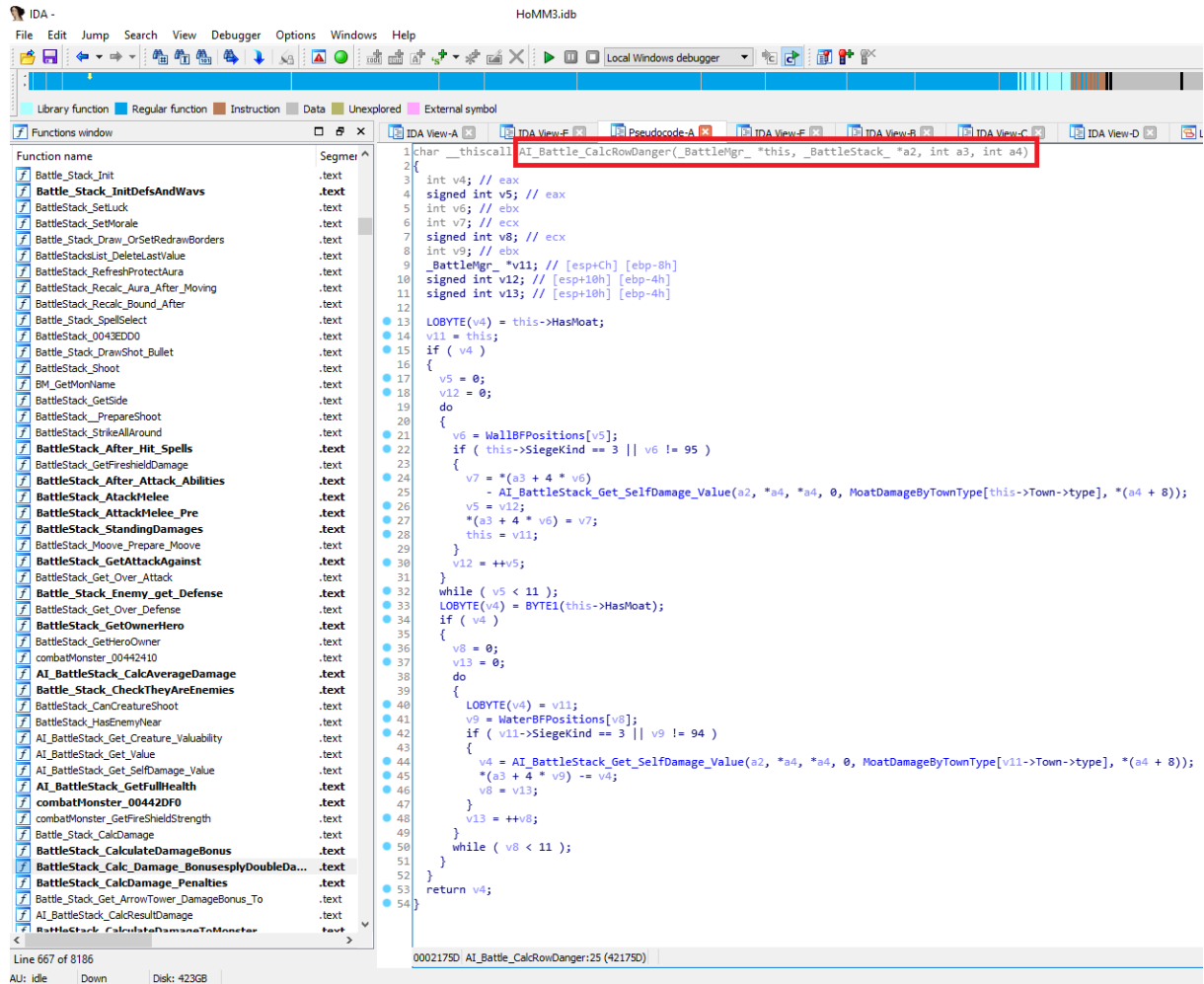
Example 1 – Remove double moat from Fortress

Imagine, that you want to disable double moat in the Fortress, and you do not know which function sets it. The first thing you have to do is open your IDA and start searching. Press **alt+T** and enter the simplest word: Moat.

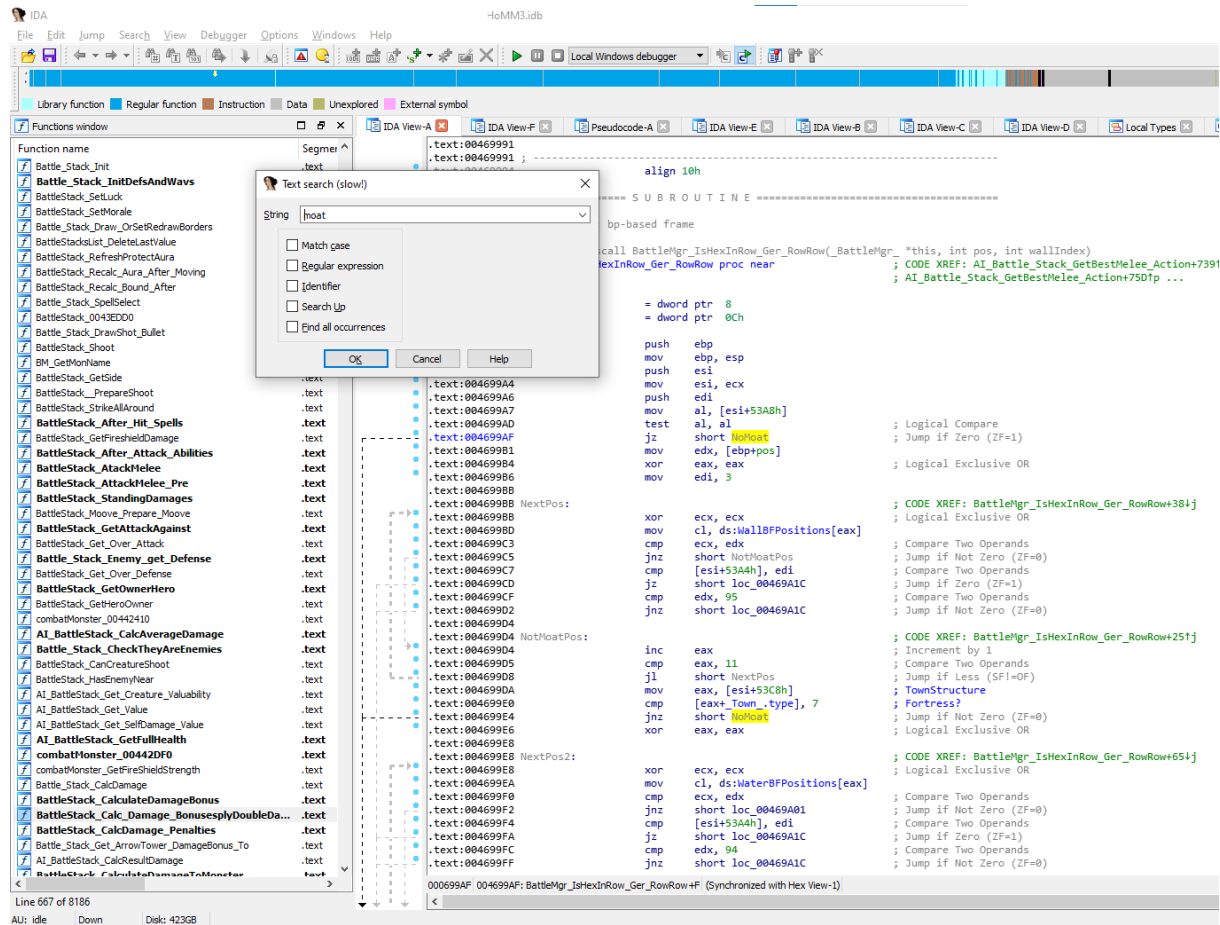
When you did it you did it and pressed "Ok", you see a random function, which does something with the moat



Use **Tab** key to get into pseudocode, representing what's going on here – you will see that it's something you don't need.



Use **Tab** again to go back into IDA view. After a few attempts more, you will find something more interesting



Press **Tab** again and – bingo! This is the right function

```
IDA View-A x IDA View-F x Pseudocode-A x IDA View-E x IDA View-B x
1 char __thiscall BattleMgr_IsHexInRow_Ger_RowRow(int this, int pos, int wallIndex)
2 {
3     int v3; // eax
4     char result; // al
5
6     if ( *(this + 21416) )
7     {
8         v3 = 0;
9         while ( WallBFPositions[v3] != pos || *(this + 21412) != 3 && pos == 95 )
10        {
11            if ( ++v3 >= 11 )
12            {
13                if ( (*(this + 21448) + offsetof(_Town_, type)) != 7 )
14                    goto NoMoat;
15                v3 = 0;
16                while ( WaterBFPositions[v3] != pos || *(this + 21412) != 3 && pos == 94 )
17                {
18                    if ( ++v3 >= 11 )
19                        goto NoMoat;
20                }
21                break;
22            }
23        }
24        if ( wallIndex )
25            *wallIndex = v3;
26        result = 1;
27    }
28    else
29    {
30    NoMoat:
31        if ( wallIndex )
32            *wallIndex = -1;
33        result = 0;
34    }
35    return result;
36 }
```

Here is the check

```

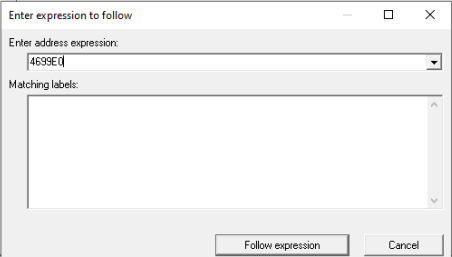
1 char __thiscall BattleMgr_IsHexInRow_Ger_RowRow(int this, int pos, int wallIndex)
2 {
3     int v3; // eax
4     char result; // al
5
6     if ( *(this + 21416) )
7     {
8         v3 = 0;
9         while ( WallBFPositions[v3] != pos || *(this + 21412) != 3 && pos == 95 )
10        {
11            if ( ++v3 >= 11 )
12            {
13                if ( (*(this + 21448) + offsetof(_Town_, type)) != 7 )
14                    goto NoMoat;
15                v3 = 0;
16                while ( WaterBFPositions[v3] != pos || *(this + 21412) != 3 && pos == 94 )
17                {
18                    if ( ++v3 >= 11 )
19                        goto NoMoat;
20                }
21                break;
22            }
23        }
24        if ( wallIndex )
25            *wallIndex = v3;
26        result = 1;
27    }
28    else
29    {
30    NoMoat:
31        if ( wallIndex )
32            *wallIndex = -1;
33        result = 0;
34    }
35    return result;
36 }
  
```

Mouse-click on it, then press **Tab** once again. You will get this:

.text:004699D5	cmp	eax, 11	; Compare Two Operands
.text:004699D8	j1	short NextPos	; Jump if Less (SF!=OF)
.text:004699DA	mov	eax, [esi+53C8h]	; TownStructure
.text:004699E0	cmp	[eax+_Town_.type], 7	; Fortress?
.text:004699E4	jnz	short NoMoat	; Jump if Not Zero (ZF=0)
.text:004699E6	xor	eax, eax	; Logical Exclusive OR
.text:004699F8			

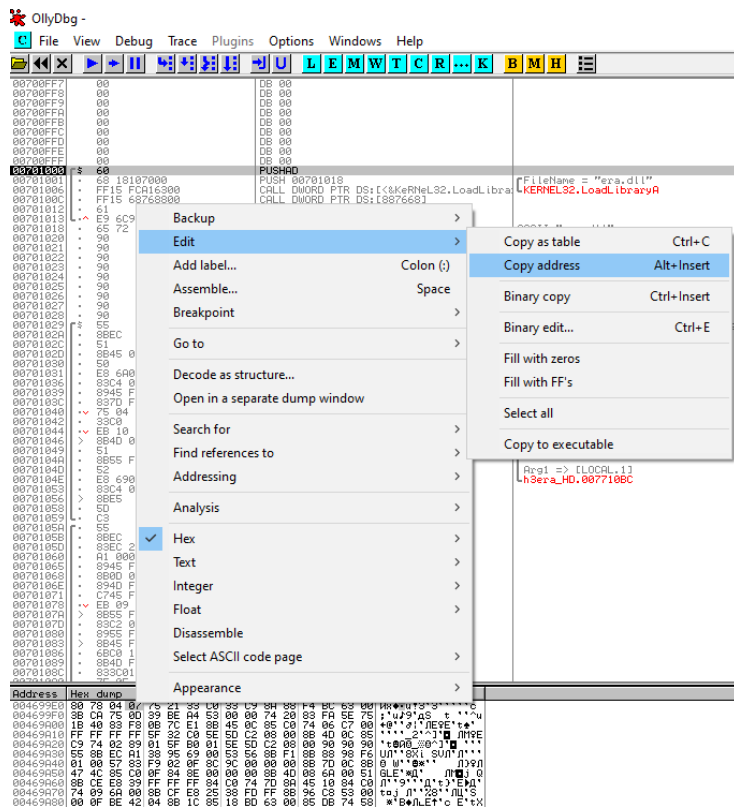
Select and copy address of comparison (cmp asm instruction)

The screenshot shows the top portion of the OilyDbg application window. The title bar reads "OilyDbg -". Below it is a standard menu bar with the following items: File, View, Debug, Trace, Plugins, Options, Windows, and Help. Each menu item has a small icon to its left. The "File" menu icon is a document, "View" is a magnifying glass, "Debug" is a bug, "Trace" is a play button, "Plugins" is a puzzle piece, "Options" is a gear, "Windows" is a window icon, and "Help" is a question mark. Below the menu bar is a toolbar containing various icons for file operations (like open, save, print), navigation (back, forward, home, search), and development tools (run, step through, breakpoints). The toolbar also includes keyboard shortcuts represented by letters: U, L, E, M, W, T, C, R, K, B, M, H, and a series of three vertical bars.

[illegible]

On this example, we can see that Town type used in comparison (*which is hardcoded*) is stored in byte on address 0x4699E3 (*every two hex values represent one byte*) Making the comparison to always return false would be more complicated, so easiest solution is to change that hardcoded value to some non-existent town, for example Town=99.

Copy address by rightclick on the value -> edit -> copy address



Now you can use it. There're many ways to change value in memory in WoG 3.59: by Lua mem, by ERM UN:C, by using Copiers array... but let's stick to easiest solution for now.

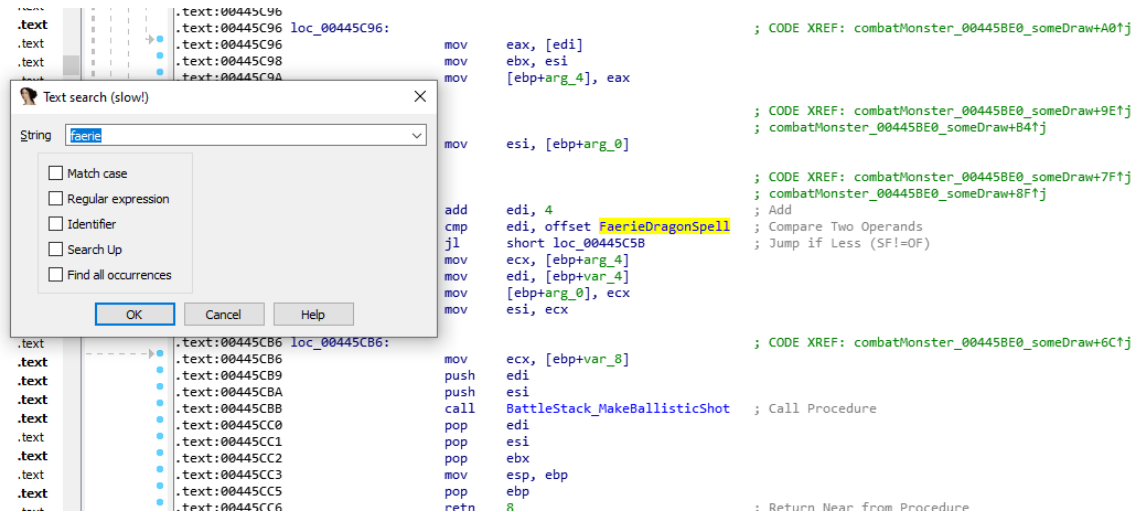
*(Byte *)0x4699E3 = 99;

Done! This will remove damage taken from first row of moat in fortress – it will still stop when stepped upon though.

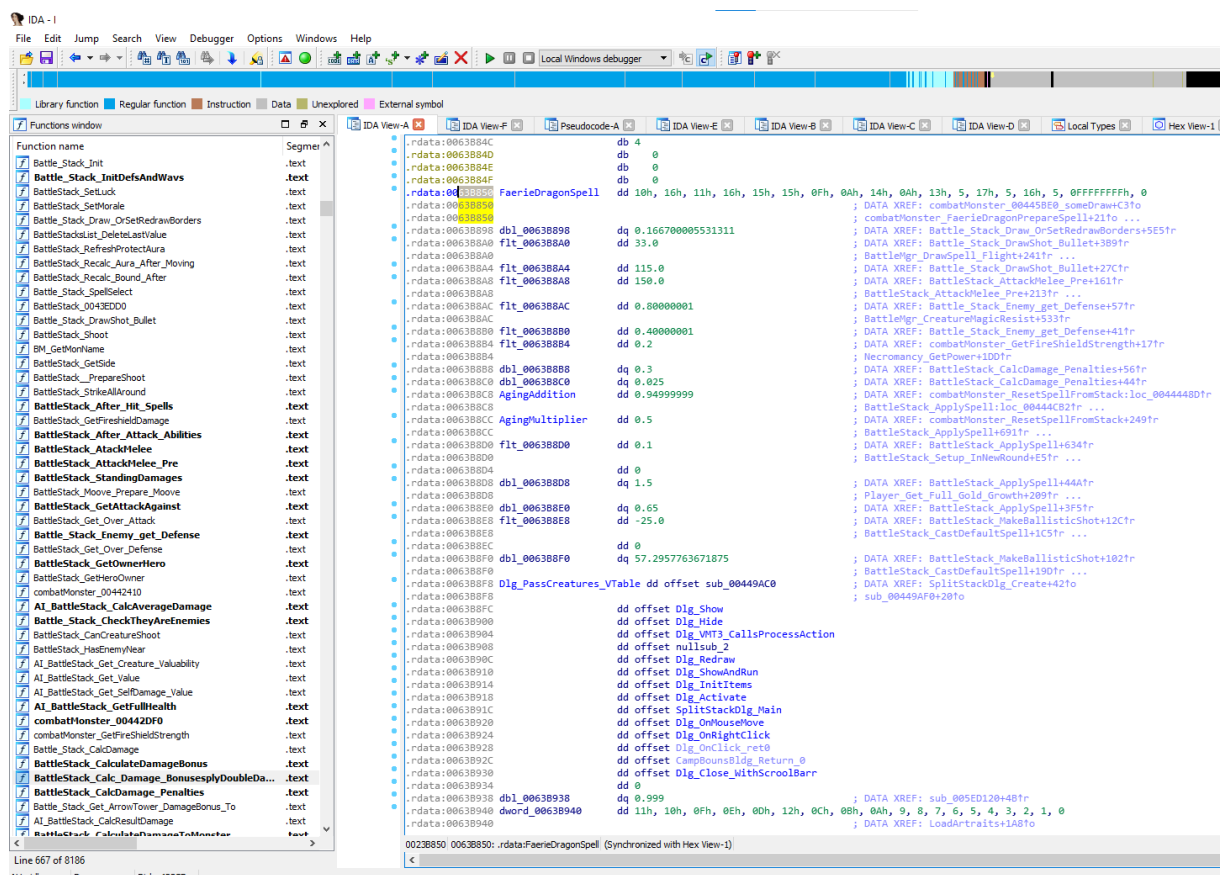
Example 2 – Make Faerie Dragon to spell „Magic arrow” instead of „Chain Lightning”

Note: works only for h3era_HD.exe, still nice for education, but don't try to recreate in 3.59.

Just start searching (**alt+T**) and find what you need

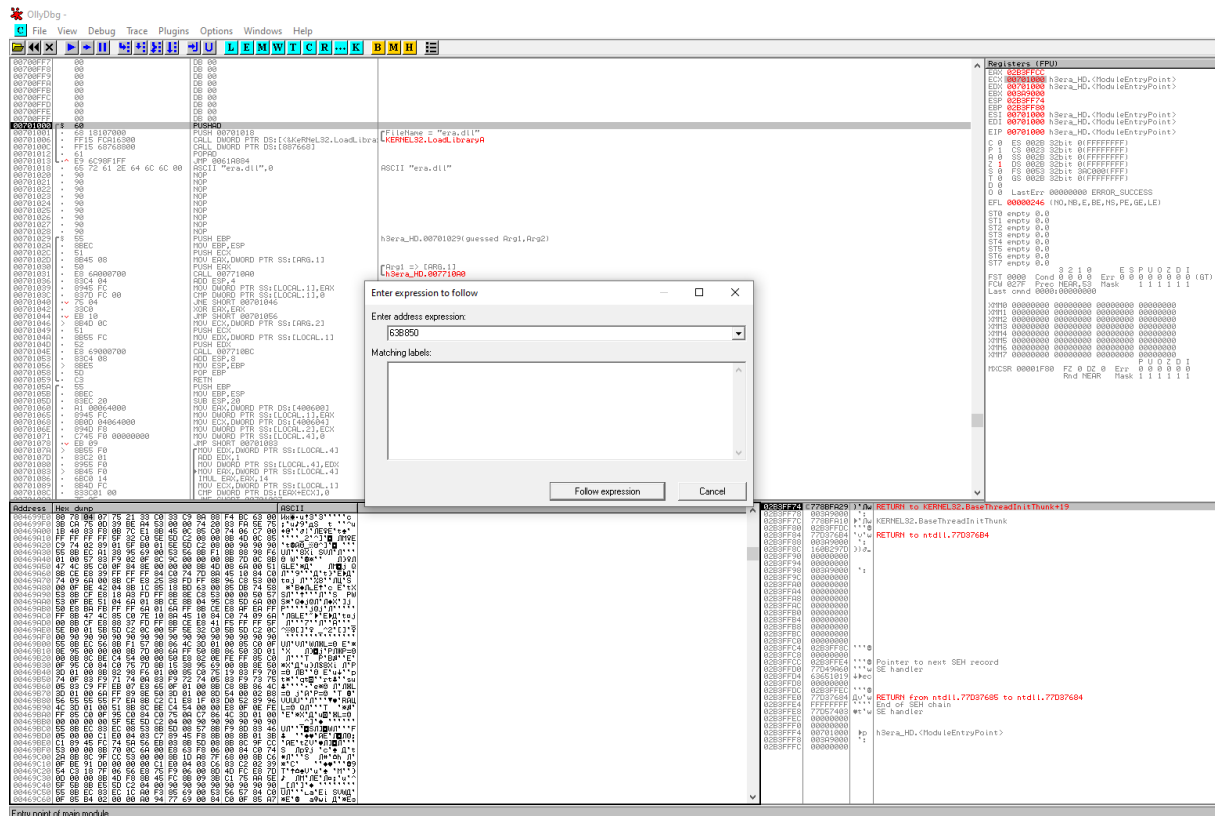


double click on the "FaerieDragonSpell", and you will go to the offset

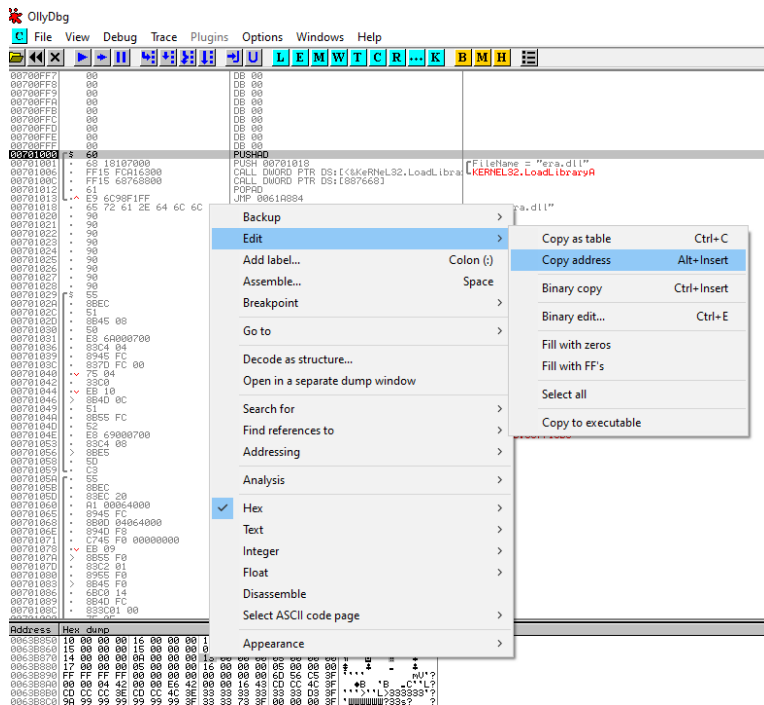


Select and copy the address

Go to Olly, click on the Hex Dump, press ctrl+G and follow the expression



the value 0x13=19 is the chain lightning (Format SP), copy it's address and do what you want



*(int *)0x6B878 = 0x0F; //replace Chain Lightning with Magic Arrow