

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Vladislav Sidorenko 192896IVSB

A Method for Bypassing the Valve Anti-Cheat System in Video Games

Bachelor's thesis

Supervisor: Kaido Kikkas
Doctor of Philosophy
(Ph.D.) in Engineering

Tallinn 2023

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Vladislav Sidorenko 192896IVSB

Meetod Valve Anti-Cheat kaitstesüsteemist möödapääsemiseks videomängudes

Bakalaureusetöö

Juhendaja: Kaido Kikkas
Tehnikateaduste
doktor

Tallinn 2023

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature, and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Vladislav Sidorenko

24.12.2022

Abstract

Game cheats have existed in almost all of video game history, and the attitude towards them nowadays is very controversial. Cheats are developed by hackers and could be used to gain unfair advantage in games. This thesis paper aims to show a method for bypassing Valve Anti-Cheat system in video games that are based on Valve's "Source" game engine, show how to create both external and an internal exploit for a video game, create dynamic link library manual mapping injector to use with the internal exploit, differentiate between cheat types. The thesis paper shows the usage of such important game-hacking tools as Cheat Engine, ReClass.NET, and IDA Pro.

This thesis is written in English and is 37 pages long, including 6 chapters, 36 figures, and 1 table.

Annotatsioon

Mänguhäkid on olemas olnud peaaegu kõigi videomängude ajaloos ja nende kohta on tänapäeval väga vastakaid arvamusi. Häkke arendavad välja mänguhäkkerid ja neid võidakse kasutada mängudes eeliste saamiseks.

Selle töö eesmärk on pakkuda välja meetod Valve Anti-Cheat kaitsesüsteemist möödapääsemiseks videomängudes, mis põhinevad Valve "Source" mängumootoril. Näidatakse, kuidas teha mängusiseseid ja -väliseid ründeid, luua mängusisese ründe käivitamiseks vajalikku DLL -teeki ning eristada erinevaid mänguhäkkide tüüpe. Töös näidatakse ka mitmete oluliste mänguhäkkimisvahendite (Cheat Engine, ReClass.NET ja IDA Pro) kasutamist.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 37 leheküljel, 6 peatükki, 36 joonist, 1 tabelit.

List of abbreviations and terms

AC Software	An Anti-Cheating software that prevents hackers from using cheats
Aimbot	A type of game cheat that moves the cursor to the correct position (e.g., the head of an enemy)
Autonomous bot	A type of game bot that is playing the game independently
C#	A high-level programming language of the C family
C++	A high-level programming language of the C family
Cave bot	A type of autonomous bot that explores locations and brings loot
Cheat	With “hack” and “game exploit” — usually means either an external or internal program that is capable of giving an advantage to the person that uses it
Cheat Engine	A memory scanner
Cheating tool	A tool for hackers such as Cheat Engine or IDA Pro
CS: GO	Counter-Strike: Global Offensive, Valve Software game
DLL	A Dynamic Link Library
DLL injection	A four-step attack that allows a hacker to execute code inside the target process
ESP	A type of game cheat that usually can draw overlay such as boxes that will display the entity’s location, health bar, other information
eSports	Electronic sports, video games competition
External exploit	Software that creates a process on the hacker's device, which manipulates values in the target process. It usually uses Windows API
Flash games	Web-Browser games
FPS	First-person shooter
Game exploit	See “Cheat”
Hack	See “Cheat”
HEX	Hexadecimal, a number with a radix of 16
IDA Pro	An interactive disassembler

Injector	A tool that is used to insert an internal exploit code into a game's memory or any other process's memory
Internal exploit	Hacking a game internally involves inserting DLL (dynamic link library) files into the game process, which allows the hacker to directly access the process's memory for faster and easier performance
IT	Internet Technologies
Java	A high-level programming language
JNA	A Java library that allows to access process memory
MMORPG	Massively multiplayer online role-playing game
ReClass.NET	Ported to .NET ReClass memory scanner
Triggerbot	A type of game cheat that toggles if some trigger was received (e.g., the enemy is close)
VAC	Valve Anti-Cheat
Wallhack	A type of game cheat that allows a cheater to see through other objects
War bot	A type of autonomous bot that fights
White hat hacker	(also ethical hacker) A security specialist who aims to identify vulnerabilities in a system and fix them

Table of contents

Author's declaration of originality	3
Abstract.....	4
Annotatsioon.....	5
List of abbreviations and terms	6
Table of contents	8
List of Figures.....	10
1 Introduction	12
1.1 Problem Declaration	12
1.2 Goal of the thesis	12
2 Methodology.....	13
2.1 Main Sources	13
2.2 Cheat Engine.....	13
2.3 Programming languages to create a cheat	13
2.4 ReClass.NET	14
2.5 IDA Pro.....	14
3 Background.....	14
3.1 History of game hacking.....	14
3.2 Game Hacking and Cracking difference	15
3.3 Technology	16
3.4 Cheat types	16
3.4.1 ESP hacks	17
3.4.2 Aimbots	17
3.4.3 Triggerbots	17
3.4.4 Aim assist	18
3.4.5 Autonomous bots.....	18
3.5 Advanced hacking	18
3.5.1 Reverse engineering	18
3.5.2 DLL Injection	20

3.5.3 Confidentiality, Integrity, and Accessibility violation	20
3.6 Anti-Cheating Software	21
3.7 Hackers and Market	22
4 Analysis	23
4.1 AssaultCube.....	23
4.1.1 Memory Scanning: Theory	23
4.1.2 Memory Scanning: Practice in AssaultCube	24
4.1.3 Pointer Scanning: Theory	27
4.1.4 Pointer Scanning: Practice in Assault Cube	28
4.1.5 Creating Assault Cube Exploit using C#.....	29
4.2 Counter-Strike: Global Offensive.....	33
4.2.1 Scanning for Entity Object	33
4.2.2 Using ReClass.NET to Reverse CS: GO Entity List.....	38
4.2.3 Disassembling Binaries and Finding Network Variables Offsets with IDA Pro	42
5 Solution.....	44
5.1 External Exploit.....	45
5.2 Internal Exploit	46
5.3 DLL Injector	47
5.4 How Long Can Hackers Stay Undetected?	48
5.5 Observation.....	48
5.6 White Hat Point of View	48
6 Summary.....	49
References	50
Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis	56
Appendix 2 – Getting Entity List	57
Appendix 3 – Show Entity Health DLL Hack.....	60
Appendix 4 – Setting Up IDA Pro	63
Appendix 5 – Memory Read and Write	65

List of Figures

Figure 1 Wallhack	17
Figure 2 IDA Pro looks	20
Figure 3 Cheat Engine main screen.....	25
Figure 4 Select AssaultCube process.....	26
Figure 5 Memory address list.....	26
Figure 6 Narrowed down to two addresses	27
Figure 7 Pointers.....	28
Figure 8 Selected pointers in the cheating table pane	29
Figure 9 Pointer and its offset	29
Figure 10 Visual Studio "Create a new project" page.....	30
Figure 11 Visual Studio created form.....	30
Figure 12 Add memory.dll package	31
Figure 13 Add application manifest file	32
Figure 14 Activate infinite ammo.....	33
Figure 15 Scan for health value.....	34
Figure 16 Remaining addresses.....	35
Figure 17 See what accesses memory address	35
Figure 18 Assembly instructions that access address.....	36
Figure 19 Find out static addresses.....	36
Figure 20 assembly instructions at 70F4F850.....	37
Figure 21 Entity health address	37
Figure 22 Attach process in ReClass.Net	38
Figure 23 Insert correct memory address	39
Figure 24 Pointer to player object	39
Figure 25 Entity information object	39
Figure 26 Offset 0x100 health.....	40
Figure 27 Fully reversed structure.....	40
Figure 28 Generated C++ code.....	41

Figure 29 Show health hack	41
Figure 30 Search in IDA Pro	42
Figure 31 Network variables	42
Figure 32 Jump to cross-reference operand.....	43
Figure 33 Offset of "m_iHealth" variable	43
Figure 34 Offset of "m_ArmorValue" variable.....	44
Figure 35 Added armor offset to local player entity	44
Figure 36 Glow hack	45

1 Introduction

Game cheats have existed in almost all of video game history, and the attitude towards them nowadays is very controversial. This thesis paper aims to tell about cheating in video games, create an open-source video game cheating tool, explain the technology behind cheating, show how easy it is to create such a tool, and show who is benefitting dealing with cheats and what might be the reasons for someone cheating.

This thesis will strive to create an open-source video game cheating tool. Show various ways of exploiting a game.

The relevance of the topic is driven by the latter growth of game cheats and hackers. Gaming is a big industry, and the market behind developing and selling cheats is growing with this industry.

1.1 Problem Declaration

Anti-Cheating systems in many games are not strong enough, a tech-savvy person with no background in hacking can bypass them by studying game hacking over some time.

According to the old game hacking web forum “UnKnoWnCheaTs”, a big number of Valve's games are hackable.[46] One of the reasons for that is the public code of their game engine, “Source”. As will be demonstrated in the practical part of this thesis, one studying the Source engine’s public code can develop a game hack that will bypass the Valve Anti-Cheat system. This affects players’ experience in Valve’s online games.

1.2 Goal of the thesis

This thesis aims to show a method to bypass the Valve Anti-Cheat system. Describe in practice the ways for making a cheating exploit on the example of Counter-Strike: Global Offensive, and compare C++, C#, and Java for creating such an exploit.

2 Methodology

The methodology part will describe chosen literature, and tools, and explain the difference between C++, C#, and Java for hacking.

2.1 Main Sources

The author of this thesis has chosen various books that describe: the topic background, and general game hacking ways:

- Mark J. P. Wolf and Bernard Perron. 2014. The Routledge Companion to Video Game Studies
- Cano, Nick. 2016. Game Hacking: Developing Autonomous Bots for Online Games.
- Consalvo, Mia, 2007. Cheating: Gaining Advantage in Videogames
- McGraw, Gary and Hoglund, Greg. 2006. Cheating Online Games.

Besides, the author studied a closed web forum GuidedHacking that the author gained access to by buying a subscription, and the oldest game hacking web forum. Forum has hundreds of users, including game hackers with many years of experience in this area:

- <https://guidedhacking.com>
- <https://www.unknowncheats.me>

2.2 Cheat Engine

The main tool for creating game hacks is the Cheat Engine memory scanner. Cheat Engine is a tool that helps to figure out how a game/application works and make modifications to it. The scripting tools of Cheat Engine can create an exploit for many games without using any additional software.

2.3 Programming languages to create a cheat

For making a cheat it is necessary to access the game's memory, such languages as C++, C#, and Java programming languages are all capable of game hack development.

For manipulating game memory, lower-level languages like C++ and C# are better, as they provide direct memory access, and no additional libraries are needed, unlike Java which needs the JNA library.

2.4 ReClass.NET

ReClass is a tool for reversing the structure of classes, intending to make it easier to understand and analyze memory, define data types, and assign variable names. It is particularly helpful when examining player classes and other objects.

2.5 IDA Pro

IDA, also called IDA Pro, is a tool that helps to understand compiled programs by breaking them down into more easily readable forms. IT professionals and security experts often use it to analyze software and is a favorite among elite game hackers. IDA Pro is a disassembler that converts complex data types and binary instructions into simpler code for analysis and investigation. It can also create maps of execution to show the instructions in a more human-readable form and can be used to analyze software on multiple architectures.

3 Background

The background part will cover the roots of game hacking, show various game cheat types, and describe game hacking tools.

3.1 History of game hacking

The earliest game cheats appeared a very long time ago when the game industry started to develop. At first, game developers created and used them in the form of “cheat codes”. These codes were secret passwords or a combination of controller buttons pressed in a

particular order. Codes were a normal part of a game environment, as they allowed developers to skip levels, and rooms, delete monsters or add weapons. This way it was easier to check a game for bugs and glitches.[1] In the year 1981 the game called “Wizardry: Proving Grounds of the Mad Overlord” came into the light, and it quickly became one of the most popular games for Apple II computers.[2] Soon after the game's release, third-party vendors started to offer programs that could alter characters' statistics and rescue dead characters left in the dungeon. Such programs were so popular, that when the second game from the Wizardry series was released, the Sir-tech¹ inserted a sheet into the game boxes, contents of this sheet were the following:

[It has come to our attention that some software vendors are marketing so-called “cheat programs.” These programs allow you to create characters of arbitrary strength and ability. While it may seem appealing to use these products, we urge you not to succumb to the temptation. It took more than four years of careful adjustment to properly balance Wizardry. These products tend to interfere with this subtle balance and may substantially reduce your playing pleasure. ...][2]

Such commercialized cheat programs were then sometimes advertised in computer games magazines.[3] TV shows such as “Cheat!” appeared, and the show description was saying: [Cheat! keeps gamers ahead of the game with strategies, secrets, and cheats for their favorite video games.][4] The community of cheaters started to grow and develop.

3.2 Game Hacking and Cracking difference

Speaking about the difference between game hacking and game cracking, the main difference is that crackers are not willing to get any advantage in a game, both hackers and crackers indeed use similar tools for doing their job, however, whilst hackers try to find specific values, disassemble them and understand how to get an advantage using their code, crackers try to get through a license authentication process. Thus, crackers are

¹ Sir-tech Software, Inc. - video game developer and publisher founded by Norman Sirotek and Robert Woodhead.[2]

trying to get a game for free. However, white hat hackers and cracker reports found vulnerabilities in the software owner and often get bounties for that.[47]

3.3 Technology

Games consist of thousands of lines of code, which consist of sets of functions and variables carrying values. Some of them represent a player's health, other ones are responsible for in-game currency. It depends on the game and how secure it is. Basically, for Flash games,¹ it is only a deal of installing special software like Cheat Engine² and changing specific values. The Internet has plenty of tutorials for these kinds of games.[5]

The situation for more advanced game types is slightly different, professional game hackers spend hundreds of hours trying to find the vulnerabilities and develop game cheats.[6]

3.4 Cheat types

Advanced game cheats include different kinds of extrasensory perception (ESP) hacks, which give the advantage to see enemies before they see a player e.g., wallhacks (seeing enemy through the wall), radar hacks (seeing enemy on minimap or radar when he is supposed to be hidden), etc. The second important side of hacks are responsive hacks, which react to in-game events or help a cheater to react to them, they include e.g., aim assists (similar to console games aim assist), aimbots (hacks that aim for the player (and sometimes even attack)), trigger bots (attack when an enemy is at gunpoint), etc. On top of that, there are also autonomous bots, usually, it is a piece of software that pretends to be real gamer and performs tasks depending on the settings (e.g., farming gold, going to the dungeon, completing quests, etc.). These bots can either work with other types of hacks (ESPs, responsive hacks) or behave like a legit gamer.

¹ Flash games - web browser games, usually they have from little to no cheat protection.

² Cheat Engine - a game memory scanner that searches for values like player's health, level, in-game money, etc.

3.4.1 ESP hacks

ESP (Extra Sensory Perception) hacks give players an advantage by intensifying their ability to percept the game world. For example, cheaters can see enemies before they see cheaters, here's an example:

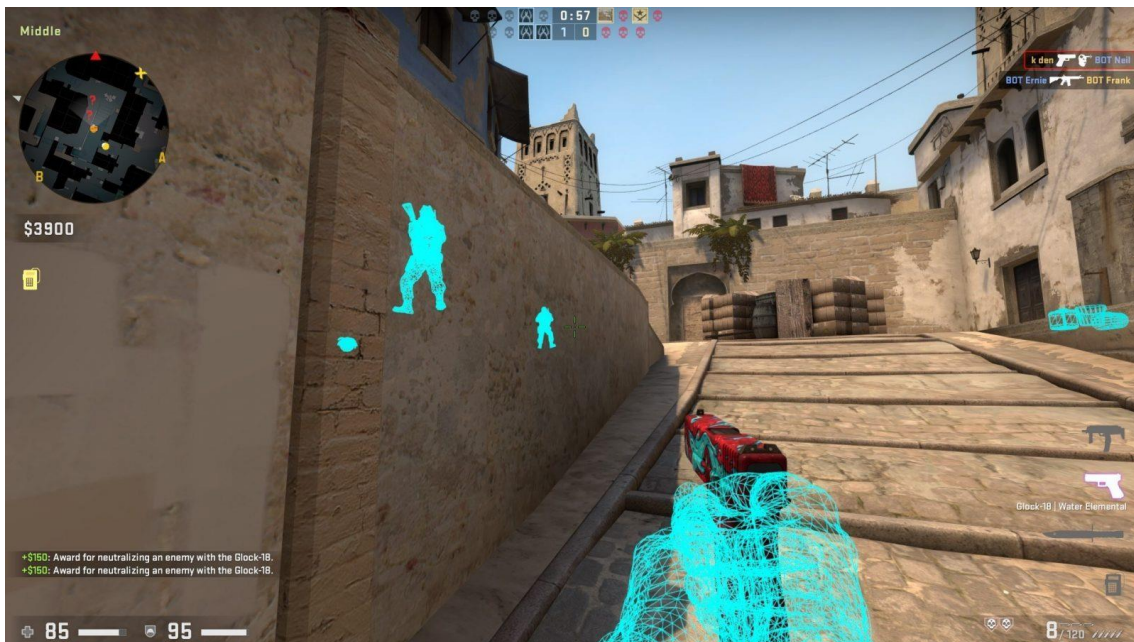


Figure 1 Wallhack

Figure 1 shows that a player sees other players through the object. More advanced ESP hacks can also help find items & loot, the enemy's health, and other tactical information.

3.4.2 Aimbots

An aimbot is a piece of software that locks a player's cursor on the enemy. It helps to maximize the damage, it can, for example, instantly move a cursor to the enemy's head or any other vulnerable spot. Aimbots rely on accessing the game memory to detect information such as the position or visibility of other players to function. [11]

3.4.3 Triggerbots

Triggerbot is a similar software to aimbot, however, instead of moving and locking the player's cursor on the enemy, it triggers some actions, when the player on one's own places the cursor on a desired spot. Example: the player moves a cursor on the enemy, triggerbot clicks a fire button.[37]

3.4.4 Aim assist

Gamers are arguing about this type of cheat, from one hand – aim assists legally exist for the controller players, since it may take years of practice to make precise movements using controller thumb sticks. On the other hand – to a lot of players, it gives people an unfair advantage. According to game developers, this goal assist is not cheating. However, aimbots for keyboard and mouse players can be configured to act exactly as aim assist for controllers. Properly configured, this type of cheat is hard to detect.

In the ESports scene pro players from time to time get caught using this type of cheat, since there's big prize money in the tournaments, for some, it is tempting to use it.[36]

3.4.5 Autonomous bots

This type is often used in MMORPG games, such as World of Warcraft, RuneScape, or Lineage – these games consume a lot of gamers' time to achieve goals. MMORPG games require players to gain in-game values, like gold, treasures, supplies, clothes, etc.

Bots are configured to automatically play games for hours on end, they are usually separated into two groups:

1. Cavebots, which can explore caves and bring home the loot
2. Warbots, which fight enemies

3.5 Advanced hacking

Making a hack is not only limited to searching for some address in memory and changing it, advanced hacking techniques require reverse engineering skills[8], being able to code a DLL (dynamic link library) injector, and bypassing anti-cheating software.

3.5.1 Reverse engineering

By itself, reverse engineering means understanding how something works. In the case of software, a reverse engineer should have experience in reading and write in both high-level and low-level languages, where high-level means any programming language that is considered for software developers (C#, Ruby, Python, Java, etc.), and low-level usually means machine code or assembly language.

Whilst manipulating game memory is not by itself an illegal action, but more than often is just a violation against “Terms of Service”, reverse engineering software might have some serious legal consequences, because it allows not only to manipulate game code and get a fancy gun but also to defeat copy protection or digital rights management schemes.[9](see 3.2)

When it comes to reverse engineering a game, more often than not the following tools are being used:

1. IDA
2. x64dbg
3. Ghidra
4. OllyDbg (for 32-bit games)

Out of these tools, IDA has the most powerful functionality – it has a disassembler, debugger, and, as an additional plugin, a decompiler.

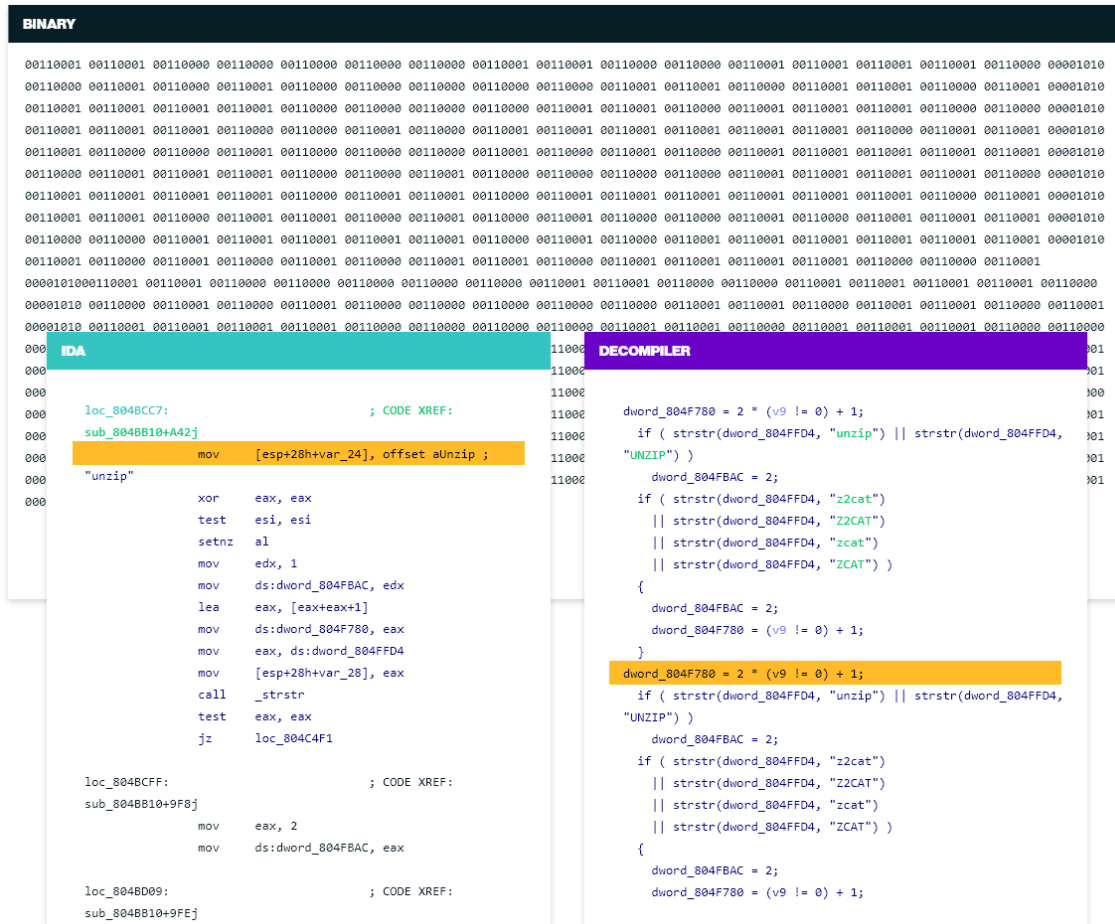


Figure 2 IDA Pro look

3.5.2 DLL Injection

DLL injection is a four-step attack:

1. Attach to the process
2. Allocate Memory within the process
3. Copy the DLL into the process's memory
4. Instruct the process to execute DLL

DLL files may contain any malware code in them.

3.5.3 Confidentiality, Integrity, and Accessibility violation

Looking carefully, it is obvious that a hacker can, and actually, many of them do violate confidentiality, integrity, and accessibility.

Using tools that were described earlier (like Cheat Engine or more like Wireshark), it is possible to find the server's address and player's IP addresses, which can allow a hacker

to determine the geolocation of a player, its client properties, etc. Also, it can be a target for DDoS Botnet attacks.[35] This is a violation of accessibility and confidentiality.

Perhaps the easiest in implementation, though a dangerous spam bot can also be made using a chat message game address and writing there basically anything, including fishing websites, which are dangerous in terms of all CIA principles.

3.6 Anti-Cheating Software

If before cheats were something new, even something that was advertised in newspapers and on TV, then now the majority of people condemn cheating and big game companies like Valve or Electronic Arts focus on taking some measures against cheating and developing sophisticated detection suites called anti-cheat software. The purpose of these anti-cheats is to detect cheats and prevent gamers from utilizing cheats, usually punishing them by using a banhammer, sometimes for a lifetime.[10]

The concept of anti-cheat technology is developing, some older AC software were performing quite simple tasks, e.g., Intel anti-cheat technology as of 2007 was working like this – chipset records all input from the keyboard and mouse, and the game does the same. If they don't match, and something is manipulated within in-game values, then a cheater is caught.

Some of the notable modern AC software are BattleEye, GameGuard, PunkBuster, VAC (Valve Anti-Cheat), and EasyAntiCheat.

For example, the VAC system scans a player's computer in the background for cheats while the game is running. In this sense it works similarly to anti-virus software, it has a database of known cheats to detect, and it also detects game file modifications and dynamic link libraries.[16]

In the real world, bypassing anti-cheating software is relatively easy to do. First and the most important rule – do not be a “script kiddie”¹ and write a hack code manually. There are a ton of cheats on the Internet, however using own, the personal cheat will already help to avoid 99% of bans.

There are different ways of bypassing anti-cheat, however, if speaking more modern anti-cheats, everything comes to mapping a cheating driver before the anti-cheat loads.[17]

3.7 Hackers and Market

However, whilst getting a few extra gold coins in a flash game is like fun, making advanced game cheats usually has other purposes and intentions.

Gaming is a big industry, with a revenue of \$131 billion (in 2018)[8] and 2.6 billion video gamers around the world.[9] In such a big money pit there are a variety of opportunities to earn money. Game studios sell their games, gamers record themselves playing them, some of them even making a decent living participating and winning in global eSports tournaments,[10] and some are making a pretty penny developing and selling cheats.[6]

Let's take MMORPG games as an example. These types of games usually have in-game currency (e.g., golden coins) and/or items, and to obtain them, players have to spend hours, days, and weeks farming. The gamer communities often have an interest in buying gold or items (e.g., because of the item statistics or item rarity). This is why hackers can either use their software to farm themselves and then sell their in-game profits or sell their software to players who wish to seamlessly obtain levels or farm gold and items with minimal interference. Due to the massive communities surrounding popular MMORPGs, these game hackers can make between six and seven figures annually.[6]

If speaking about competitive types of games, then it is even more clear, as many of them have eSports tournaments, a prize pool which is increasing from year to year and

¹ Usually a “script kiddie” is a person who is using already existing solutions/software/scripts to perform a hacking attack.

sometimes can reach as much as millions of USD.[11] Taking that into consideration, the price eSports players can pay to get private, timely updated, undetected hacks to win tournaments can be very high.

One more example is Twitch streamers and YouTubers that lure an audience by showing how “skillful” they are. Live streamers and video content creators gain popularity not only because of how fun they are but also because of their skills. Professional gamers (not necessarily eSports players) are known to become popular and get some donations from viewers for being good at games, and some of them resort to cheating to achieve that popularity (sometimes achieving a ban).[12]

4 Analysis

The analysis part will describe game hacking, first in Assault Cube, a common game to start practicing game hacking, it does not have any anti-cheating software, then in Counter-Strike: Global Offensive, which is protected by Valve Anti-Cheat.

The Assault Cube part is a brief example of scanning memory and looking for pointers, creating an external trainer.[22] The CS: GO part is thorough, where besides Cheat Engine the author will use an Ida Pro disassembler, and ReClass memory scanner.

4.1 AssaultCube

AssaultCube is a free multiplayer FPS game, it can be downloaded from the original website.[18] Assault Cube is a recommended game to start learning game hacking. In this paper, the author uses the 1.2.0.2 version of the game.

4.1.1 Memory Scanning: Theory

Unlike humans, the software cannot determine a game's state just by looking at the screen. Games are built using code, our computers “perceive” it as thousands of lines of zeros and ones.[19] Our computers can understand numeric representation of a game’s state,

thus when a hacker wants to know which piece of code represents our game health, ammo, or character moving speed, they need to use memory scanners.[6]

Memory scanning can help hackers with creating a cheating exploit, for example, if they were playing a game with shooting mechanics, perhaps they would first seek for ammo value location in the game's memory using a memory scanner, then create software to change this value according to their needs. Take a look at the next pseudocode:

```
ammo = readMemory(game, Ammo_Location);  
loop (true)  
    if (Infinite_Ammo_Checkbox == true)  
        WriteMemory(ammo, 8000);  
    Wait(2);
```

In the pseudocode above, the hacker first writes read memory of ammo in a game, then they create a loop, which rewrites the value into this part of memory.[6]

4.1.2 Memory Scanning: Practice in AssaultCube

For the practice part, the author is going to use Cheat Engine version 7.4 as a memory scanner, and AssaultCube as a game to be hacked.

Cheat Engine is freeware and can be easily downloaded and installed from the original website.[20] Let's take a look at this software's main screen.

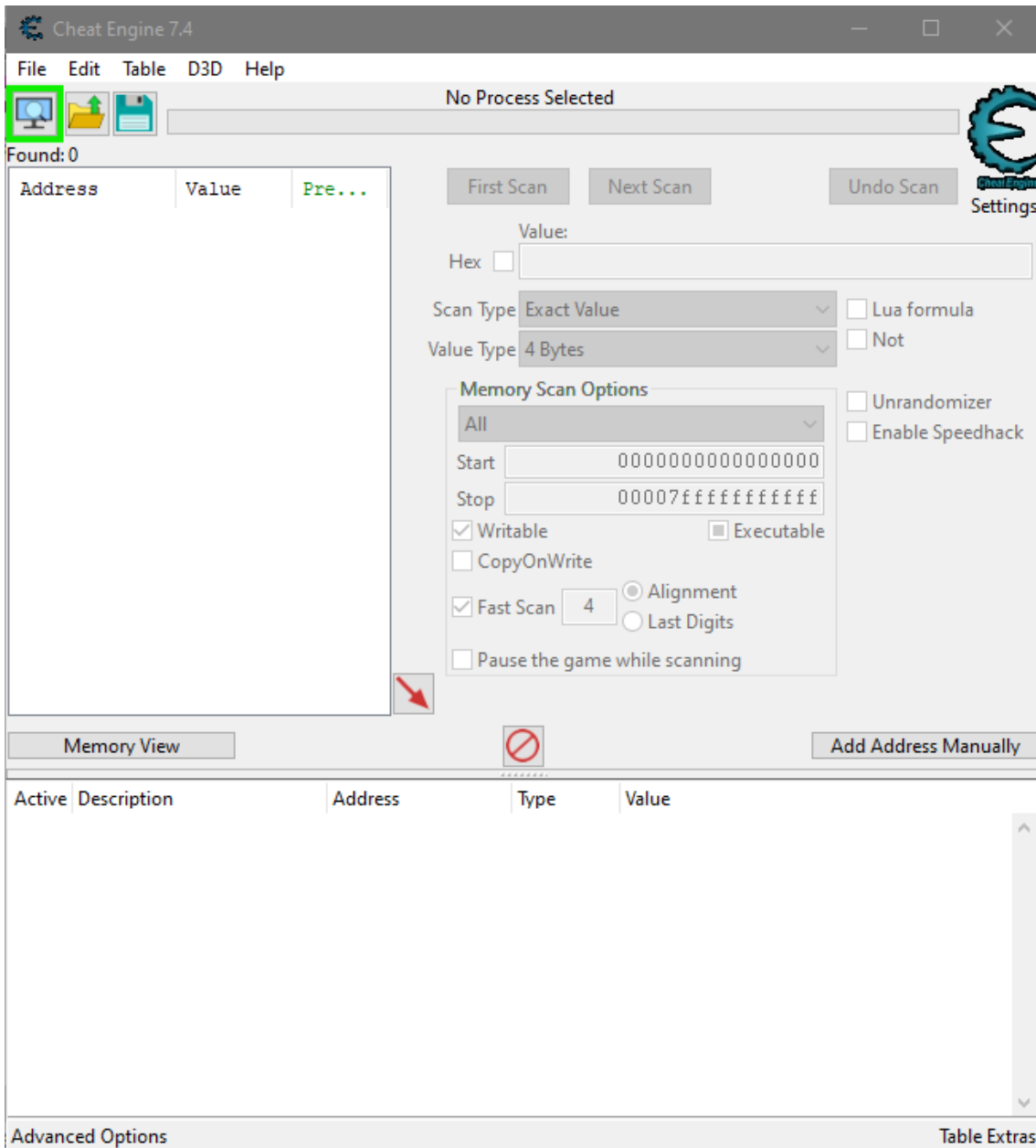


Figure 3 Cheat Engine main screen

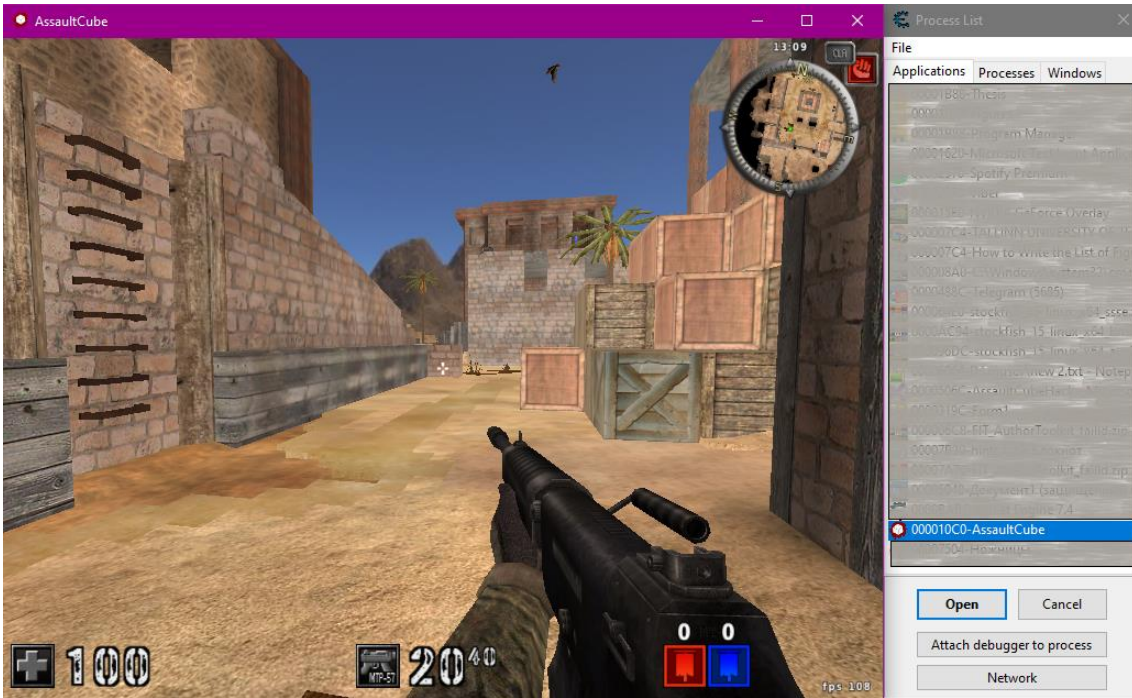


Figure 4 Select the AssaultCube process

As can be seen in Figure 4., the author has 100 health points and 20 bullets of ammo loaded. Then the author created a new scan for ammo value, thus scanning the game process for value “20”. [21]

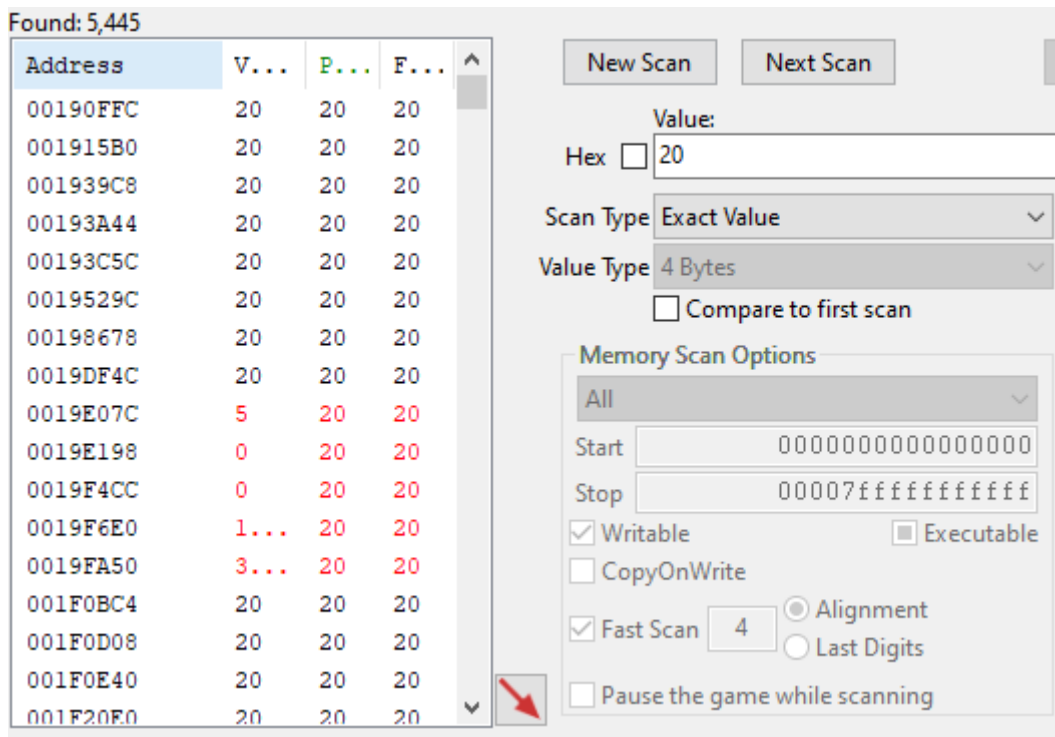


Figure 5 Memory address list

Initially, the author got 5,445 addresses with the same value. Now one would want to narrow down the amount of these addresses. The author spent some ammo and created a next scan for an updated value. The author has to perform this action until as few addresses as possible.

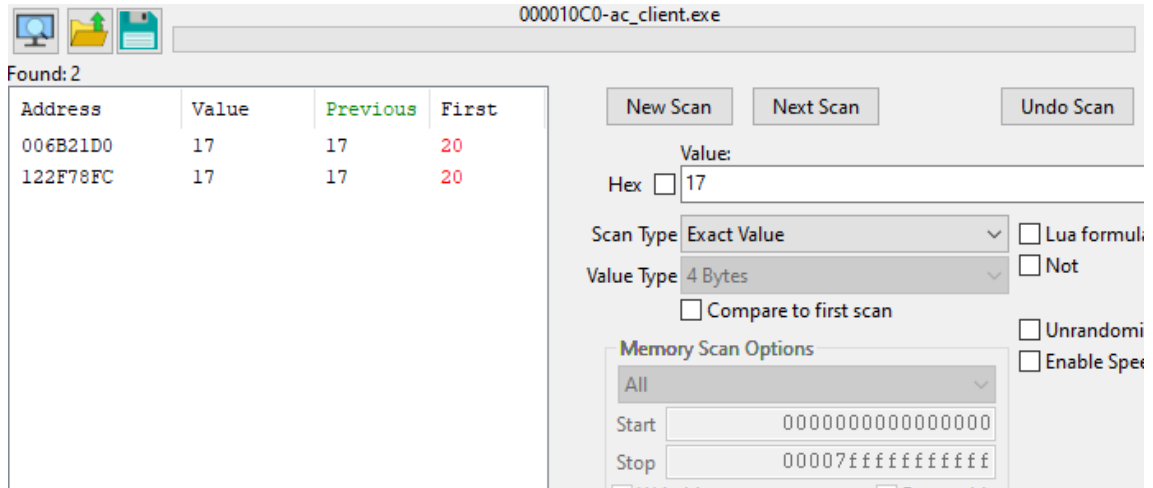


Figure 6 Narrowed down to two addresses

After several scans, the author got two addresses, which could possibly represent the ammo. Let's add both of these values to our *cheat table pane* by double-clicking them. Next, a hacker wants to perform some actions with these values, so they would know which value is the ammo value. This can be simply done by modifying the value number or freezing this value by clicking the "Active" checkbox. Now in-game ammo quantity is either updated or frozen, depending on the actions performed.

At this point, the memory value is already modified.

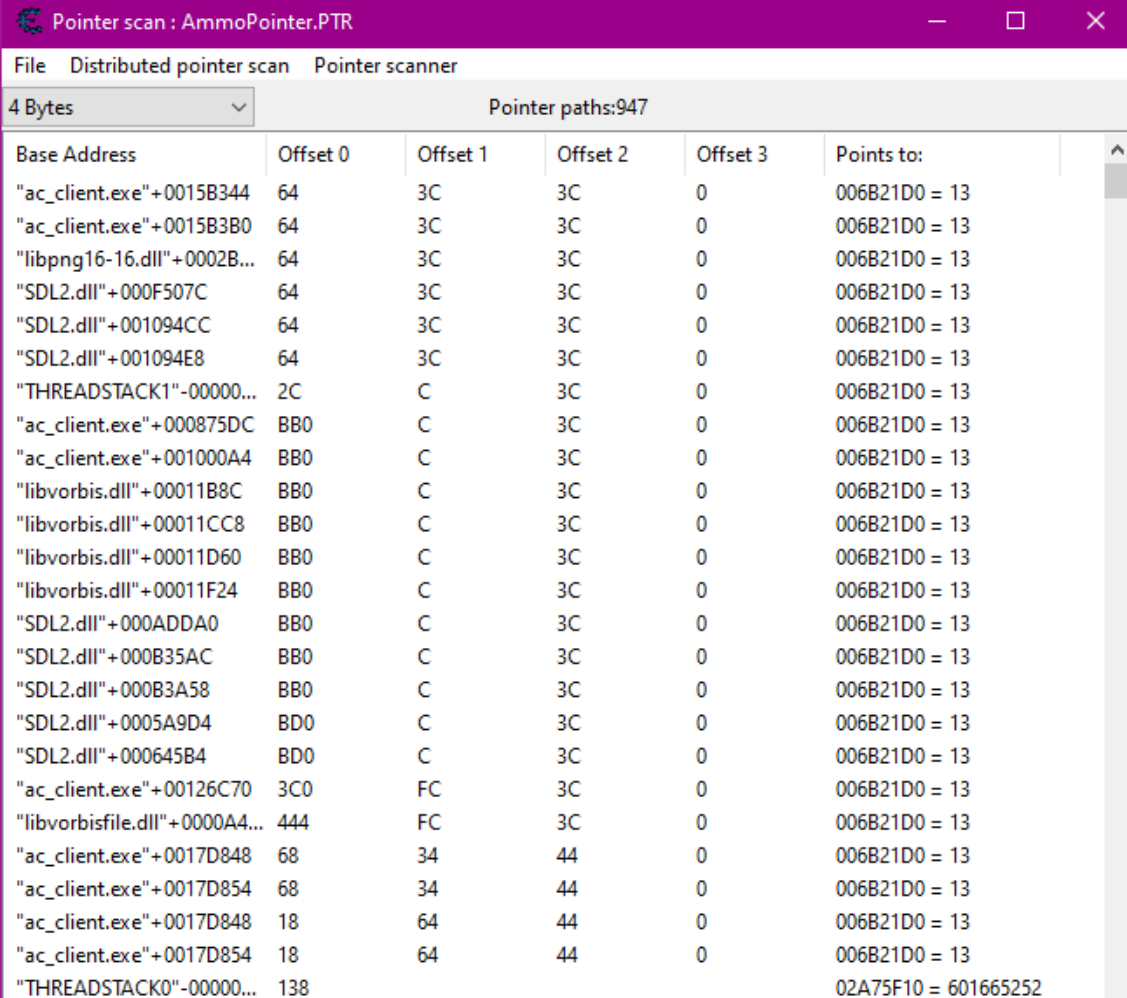
4.1.3 Pointer Scanning: Theory

Games often store values in dynamically allocated memory, before this paper was dealing with static memory, which by itself is not beneficial, since one would need to scan a game for static address after every game restart. To make a working cheating tool, one needs to get a dynamic memory address of the desired game value.

The static address always points to another static address, which points to another static address and so further on.[22] This is called a chain of offsets (or a pointer chain).

4.1.4 Pointer Scanning: Practice in Assault Cube

Cheat Engine has the functionality to scan pointers. To do this, one would need to right-click on the ammo address and then on “Pointer scan for this address”.[23]



Base Address	Offset 0	Offset 1	Offset 2	Offset 3	Points to:
"ac_client.exe"+0015B344	64	3C	3C	0	006B21D0 = 13
"ac_client.exe"+0015B3B0	64	3C	3C	0	006B21D0 = 13
"libpng16-16.dll"+0002B...	64	3C	3C	0	006B21D0 = 13
"SDL2.dll"+000F507C	64	3C	3C	0	006B21D0 = 13
"SDL2.dll"+001094CC	64	3C	3C	0	006B21D0 = 13
"SDL2.dll"+001094E8	64	3C	3C	0	006B21D0 = 13
"THREADSTACK1"-00000...	2C	C	3C	0	006B21D0 = 13
"ac_client.exe"+000875DC	BB0	C	3C	0	006B21D0 = 13
"ac_client.exe"+001000A4	BB0	C	3C	0	006B21D0 = 13
"libvorbis.dll"+00011B8C	BB0	C	3C	0	006B21D0 = 13
"libvorbis.dll"+00011CC8	BB0	C	3C	0	006B21D0 = 13
"libvorbis.dll"+00011D60	BB0	C	3C	0	006B21D0 = 13
"libvorbis.dll"+00011F24	BB0	C	3C	0	006B21D0 = 13
"SDL2.dll"+000ADDA0	BB0	C	3C	0	006B21D0 = 13
"SDL2.dll"+000B35AC	BB0	C	3C	0	006B21D0 = 13
"SDL2.dll"+000B3A58	BB0	C	3C	0	006B21D0 = 13
"SDL2.dll"+0005A9D4	BD0	C	3C	0	006B21D0 = 13
"SDL2.dll"+000645B4	BD0	C	3C	0	006B21D0 = 13
"ac_client.exe"+00126C70	3C0	FC	3C	0	006B21D0 = 13
"libvorbisfile.dll"+0000A4...	444	FC	3C	0	006B21D0 = 13
"ac_client.exe"+0017D848	68	34	44	0	006B21D0 = 13
"ac_client.exe"+0017D854	68	34	44	0	006B21D0 = 13
"ac_client.exe"+0017D848	18	64	44	0	006B21D0 = 13
"ac_client.exe"+0017D854	18	64	44	0	006B21D0 = 13
"THREADSTACK0"-00000...	138				02A75F10 = 601665252

Figure 7 Pointers

A lot of pointers are shown, which point to the ammo memory address. Again, one would need to narrow down the number of results. For this, restart the game and find the ammo memory address once again.

The next step is to rescan the pointers list (Pointer scanner => Rescan pointer list) with the new value, this will throw away wrong pointers. Ideally, this step should be redone a few times, so one would get not more than twenty pointers.

When this step is done, one would want to get some addresses into the *cheating table pane* and reset the game again. Test the pointers like the author did previously with an ammo memory address by clicking the “Active” checkbox.

Active	Description	Address	Type	Value
<input type="checkbox"/>	No description	006B21D0	4 Bytes	??
<input type="checkbox"/>	No description	007A3F70	4 Bytes	??
<input checked="" type="checkbox"/>	pointerscan result	P->00922BC8	4 Bytes	20
<input type="checkbox"/>	pointerscan result	P->00922BC8	4 Bytes	20
<input type="checkbox"/>	pointerscan result	P->00922BC8	4 Bytes	20
<input type="checkbox"/>	pointerscan result	P->726F6381	4 Bytes	??
<input type="checkbox"/>	pointerscan result	P->00922BC8	4 Bytes	20
<input type="checkbox"/>	pointerscan result	P->????????	4 Bytes	??
<input type="checkbox"/>	pointerscan result	P->0052E48C	4 Bytes	6646895
<input type="checkbox"/>	pointerscan result	P->00922BC8	4 Bytes	20
<input type="checkbox"/>	pointerscan result	P->00922BC8	4 Bytes	20

Figure 8 Selected pointers in the cheating table pane

It is necessary to check every pointer until the right one is found.

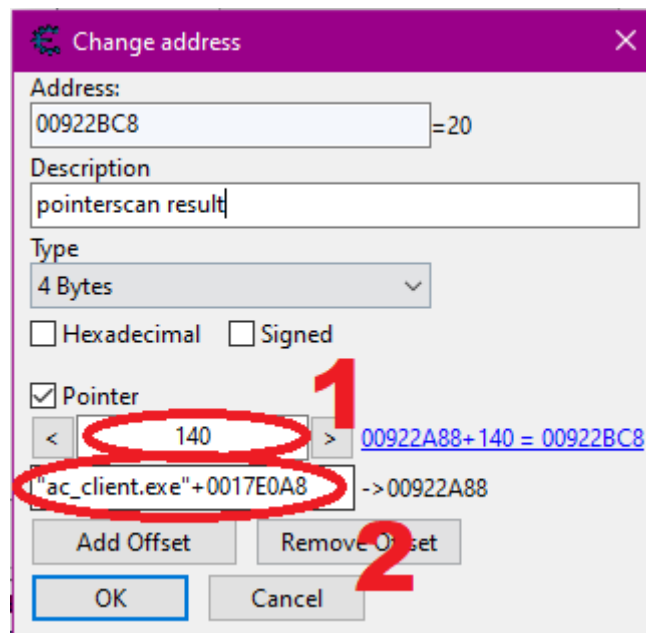


Figure 9 Pointer and its offset

Write down pointer (2) and its offset (1), it is needed to create the exploit.[21]

4.1.5 Creating Assault Cube Exploit using C#

This paper will show an example of creating an exploit application based on .NET Windows Forms.

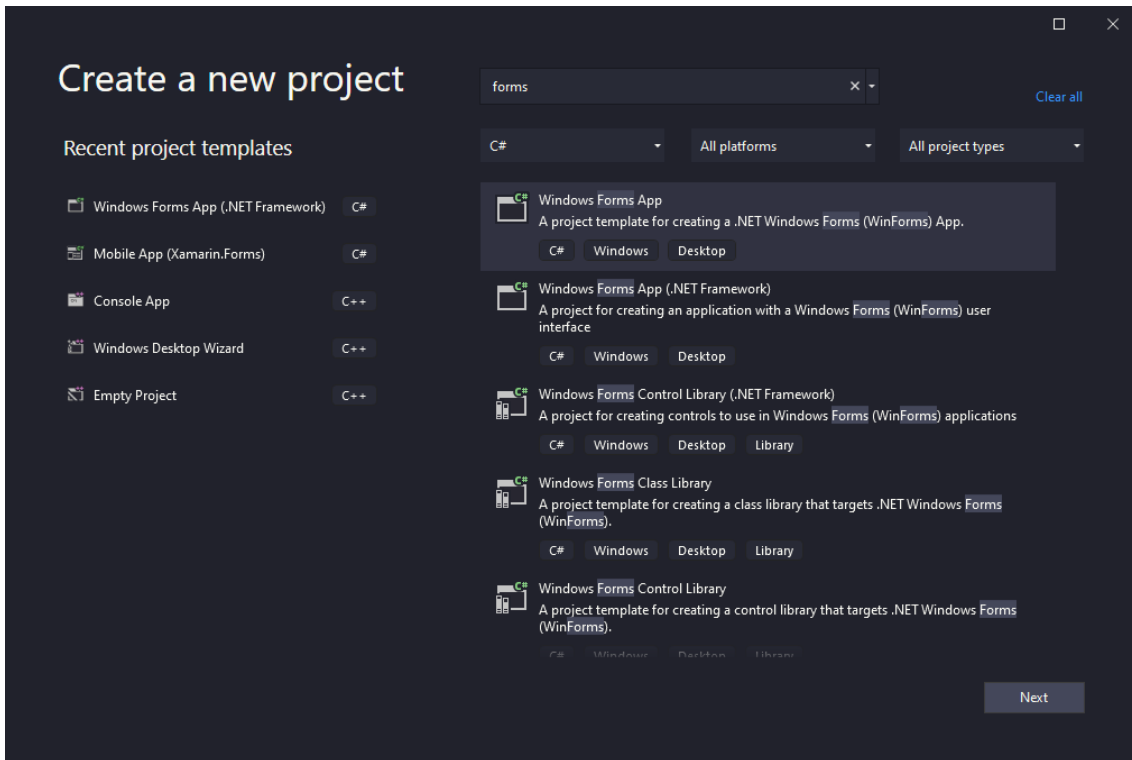


Figure 10 Visual Studio "Create a new project" page

The author has created a window for the exploit, added a checkbox, and named it “Infinite Ammo”.

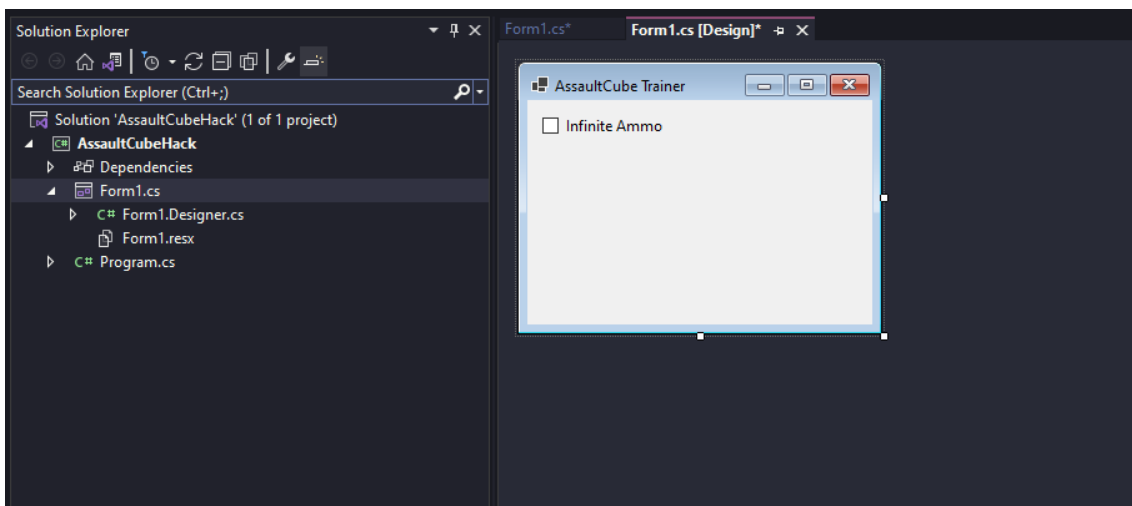


Figure 11 Visual Studio created form

In the load method of the form, the author made a string “Infinite ammo” and gave it a value of the pointer and its offset.

```
public static string InfiniteAmmo = "ac_client.exe+0x0017E0A8,140";
```

Add memory.dll class from the NuGet packages repository. It's needed to write to a memory address.

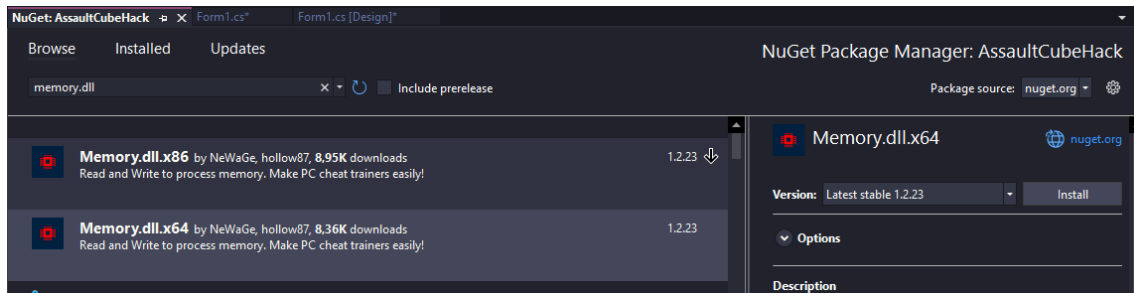


Figure 12 Add memory.dll package

Initialise class:

```
Mem memory = new Mem();
```

In the form load method check if the game is already opened, if it is – start a new thread for the write method and let it run in the background.

```
private void Form1_Load(object sender, EventArgs e)
{
    int ProcessID = memory.GetProcIdFromName("ac_client");
    if (ProcessID > 0)
    {
        memory.OpenProcess(ProcessID);
        Thread WA = new Thread(WriteAmmo) { IsBackground = true };
    };
    WA.Start();
}
```

Write method:

```
private void WriteAmmo()
{
    while (true)
    {
        if (checkBox1.Checked)
        {
            memory.WriteMemory(InfiniteAmmo, "int", "8888");
            Thread.Sleep(100);
        }
        Thread.Sleep(100);
    }
}
```

The last step is to add an app manifest file to the solution:

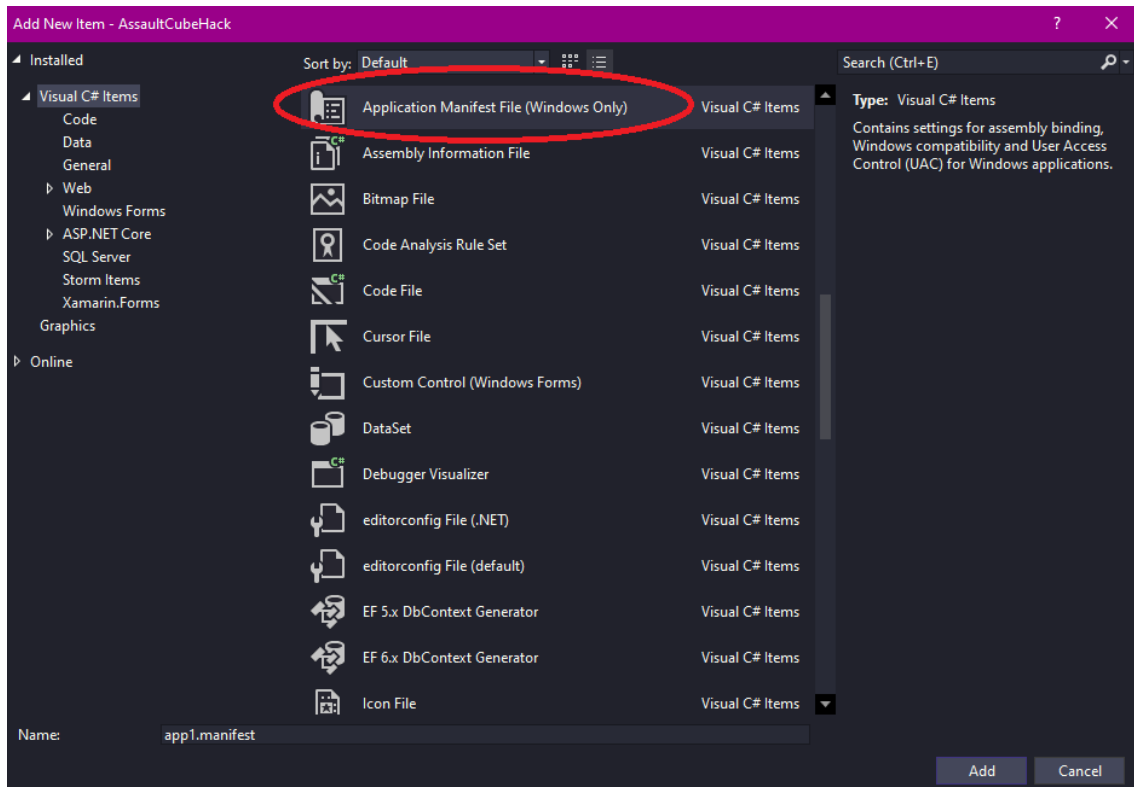


Figure 13 Add application manifest file

Inside the application manifest file, edit the next line from:

```
<requestedExecutionLevel level="asInvoker" uiAccess="false" />
```

To:

```
<requestedExecutionLevel level="requireAdministrator" uiAccess="false" />
```

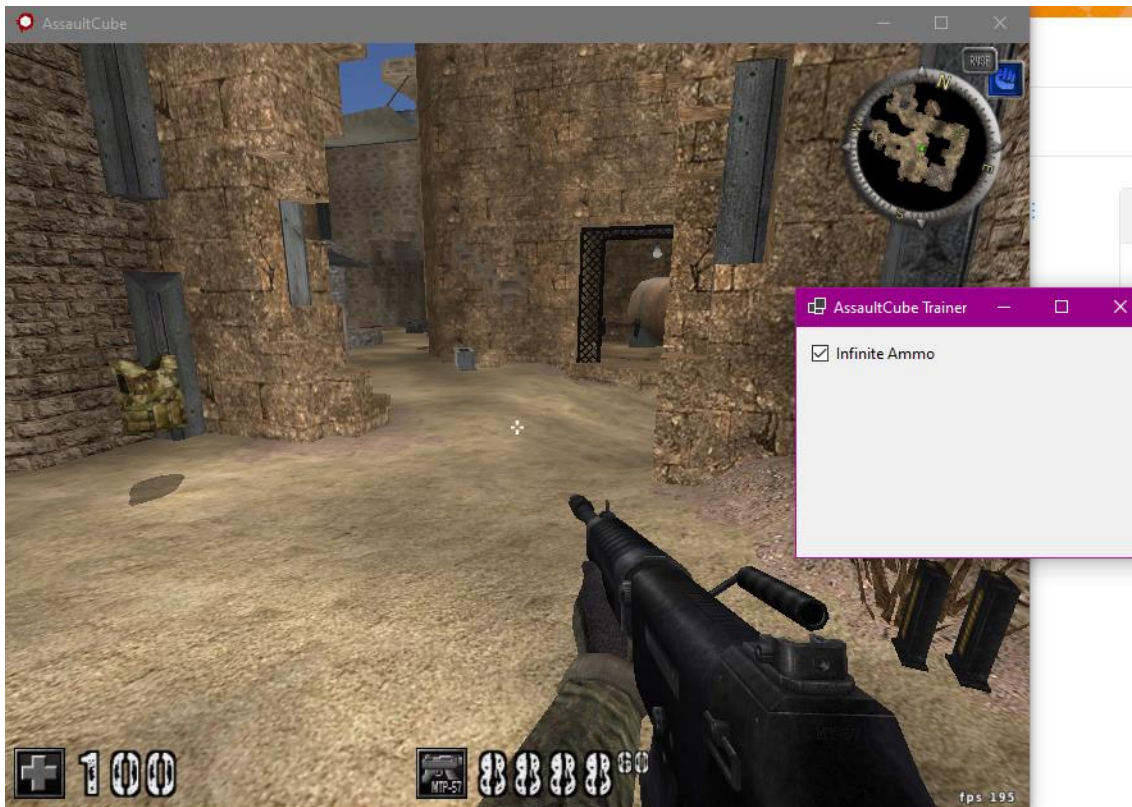



Figure 14 Activate infinite ammo

Exploit now sets ammo at 8888 every 100 milliseconds.[24]

4.2 Counter-Strike: Global Offensive

CS: GO is one of Valve's games that can be hacked. Since the game is running under the Source game engine, the methods are applicable (and sometimes similar) for other Source engine games, like Team Fortress 2, Counter-Strike: Source, Half-Life, etc.

4.2.1 Scanning for Entity Object

Before any scanning/memory manipulation, the game has to be started without the VAC module by providing an -insecure startup setting.[25]

In the game entity object means class, like a class in programming.[26] To start, in CS: GO a hacker needs to find a static entity list memory pointer. They do so by finding entity health dynamic addresses in the Cheat Engine. Then they will create a hypothesis, where they could find static entity object and static health address.

It is obvious that entities start health is 100, thus in Cheat Engine attach the “csgo.exe” process and scan for the exact value 100, 4 Bytes type.

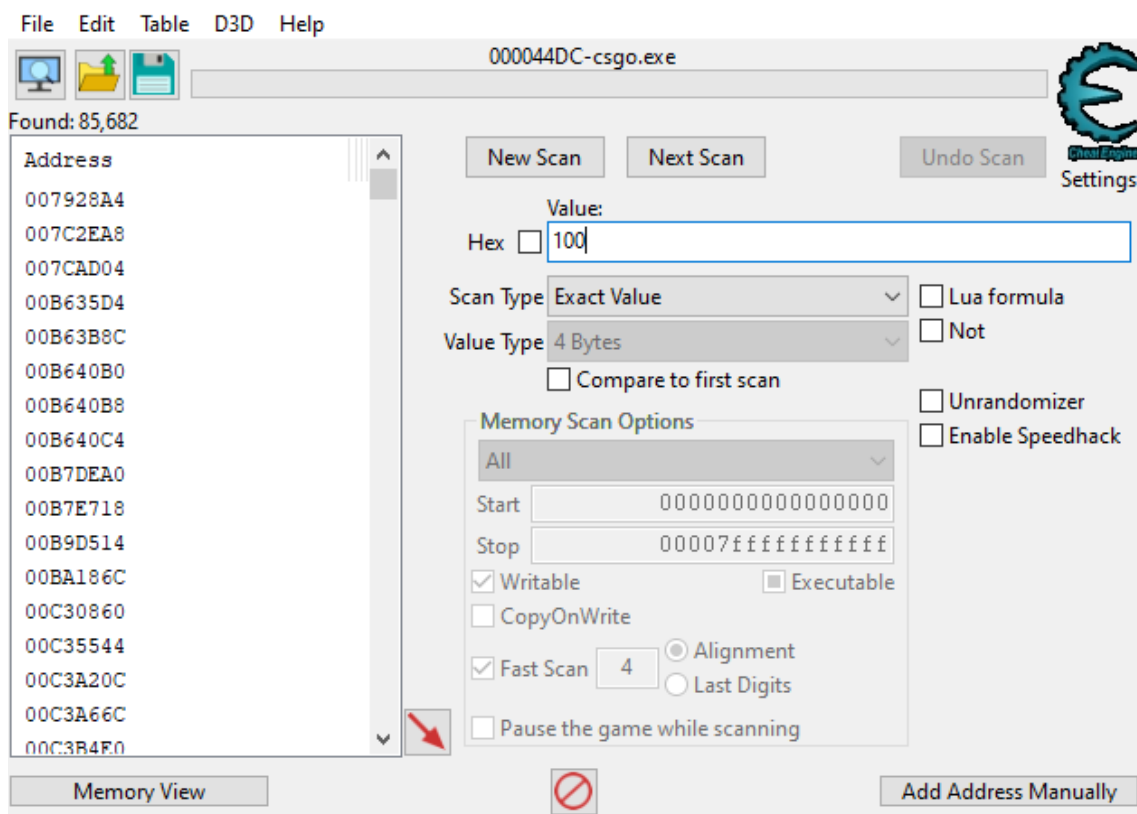


Figure 15 Scan for health value

A quite huge number of results will be shown. To narrow it down, the enemy can be shot at, then the next scan for the exact value is performed. The process should be repeated until the minimum number of dynamic addresses is left.[6] They all are related to health, if modified, some of them (server-side addresses) can change entity health if the server is hosted by a player. Add all remaining addresses to the address list.

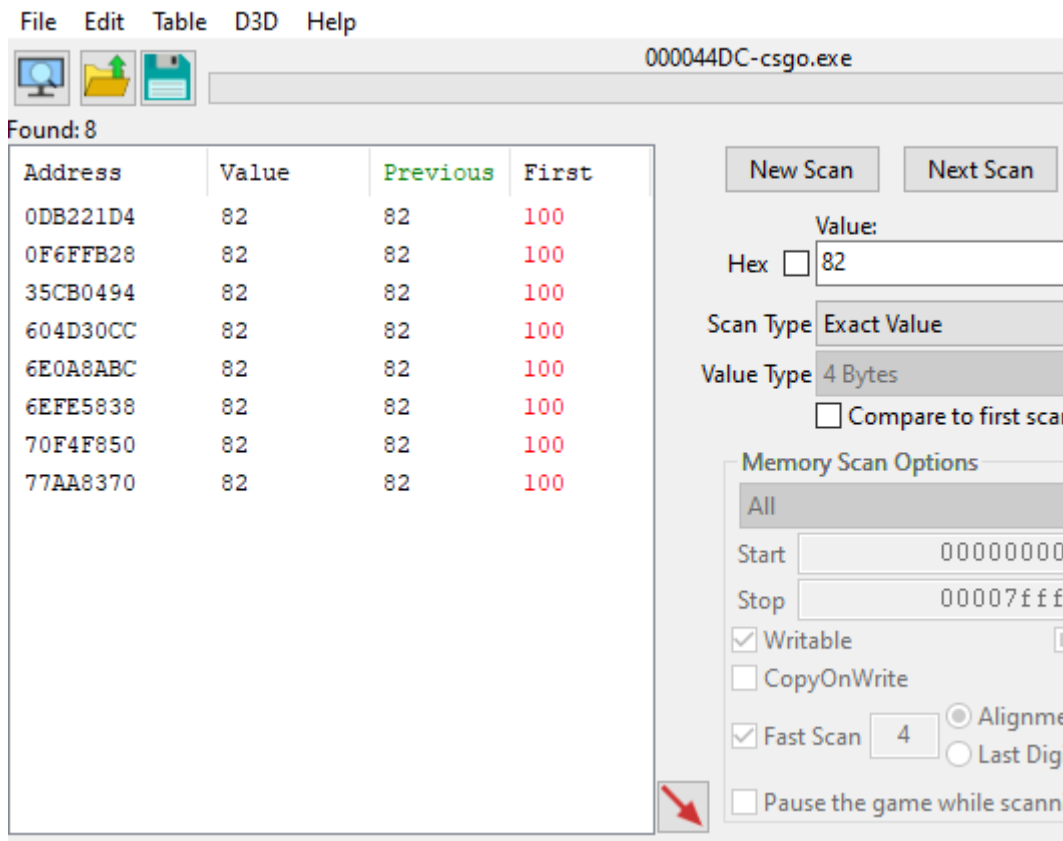


Figure 16 Remaining addresses

The next step is to check what accesses these addresses by right-clicking on them and choosing “Find out what accesses this address”. There is not a straightforward simple way to find the needed address, however more often than not it is intuitive, most likely following game logic.[27]

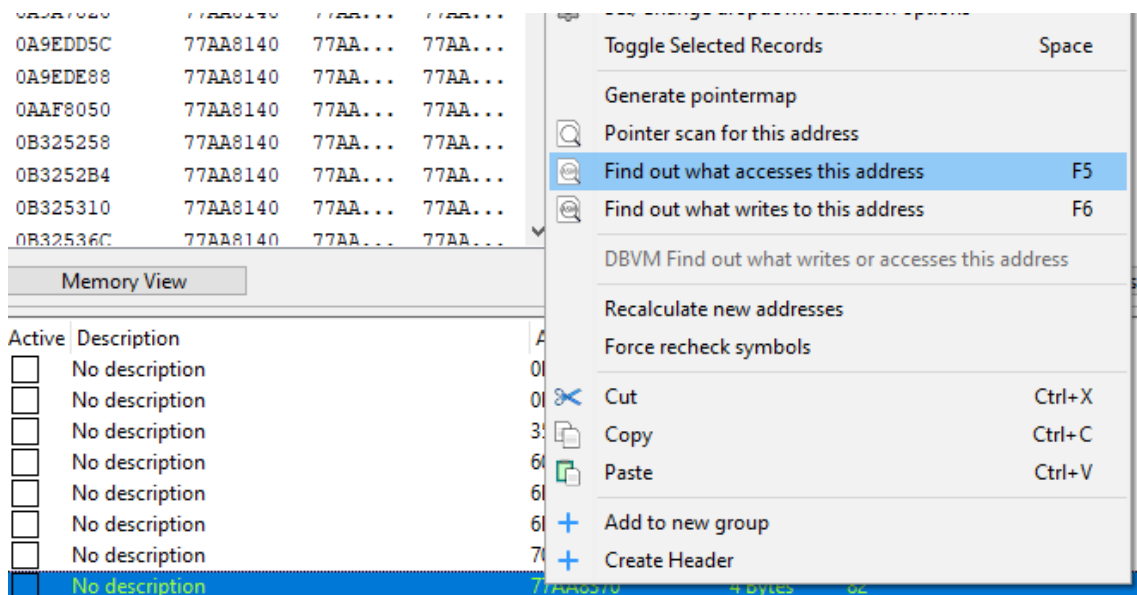


Figure 17 See what accesses the memory address

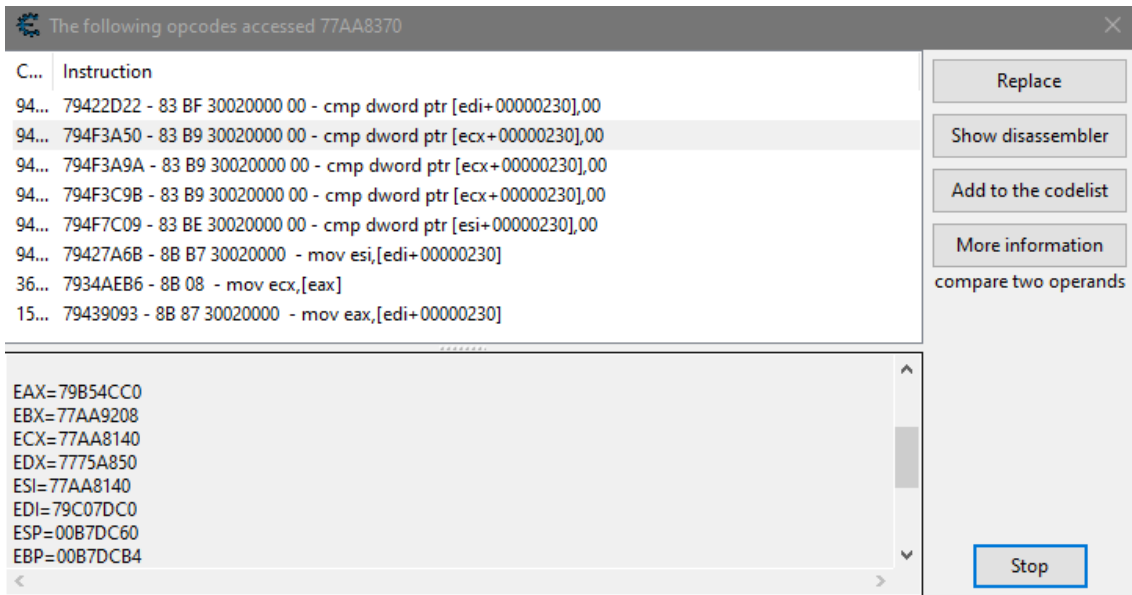


Figure 18 Assembly instructions that access address

These assembly instructions say that something happens at pointers [ecx+00000230], [edi+00000230], [esi+00000230]. Below, are the addresses of these ECX, EDI, and ESI registers, they are all the same. Now, let's scan for the Hex address 77AA8140.

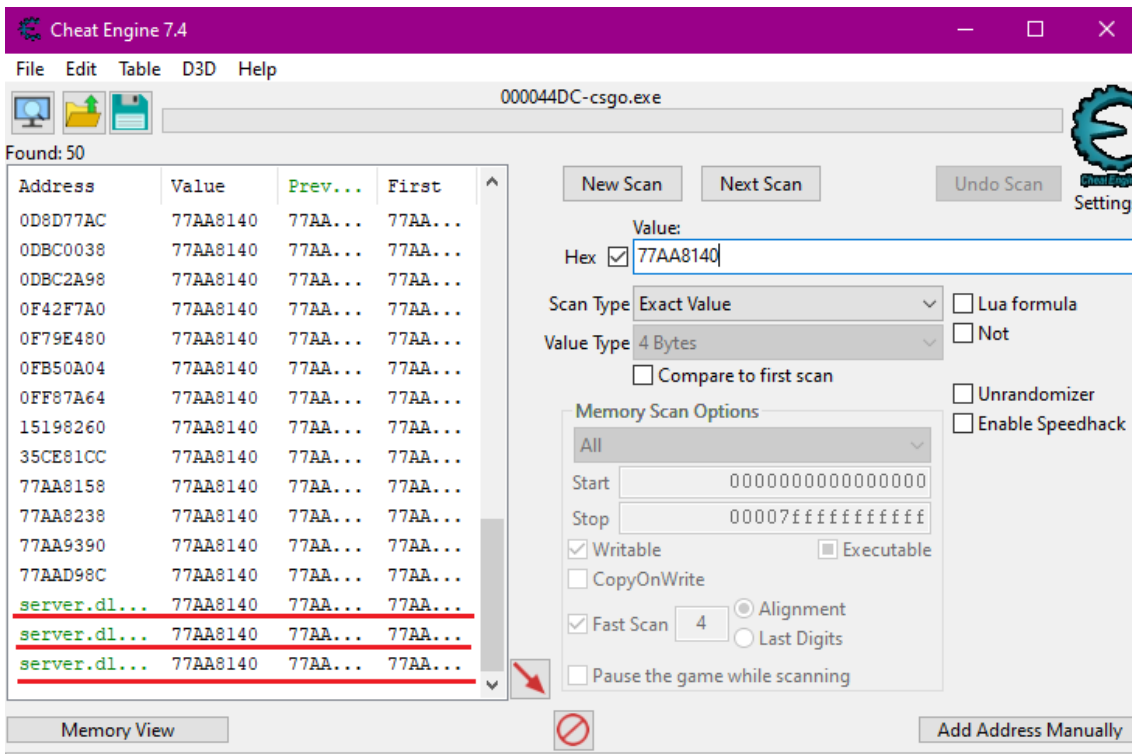


Figure 19 Find out static addresses

Green address in Cheat Engine means static address.[28] The static address will not change after the game restart. However, this static address is from the "server.dll" game

module. This game module will change only server-sided values, which is irrelevant in case one would want a cheat to work online.

So, repeat the process with the next dynamic address (70F4F850).

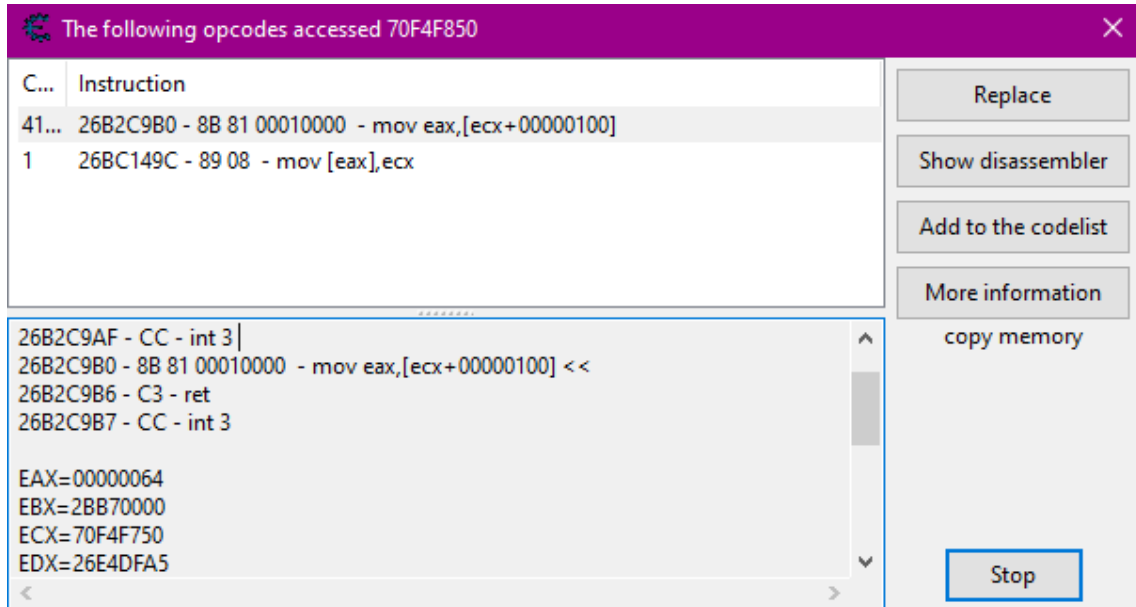


Figure 20 assembly instructions at 70F4F850

Again, it shows, that something happens at [E** + offset] pointer. ECX is 70F4F750. Hex scan this address, and the output is nearly 200 results of dynamic addresses, however, there is one static address “client.dll+4DFCE84”, which is a static address of the bot entity object. Now, from the above figure take offset and add a pointer manually.

It now leads to a static bot health address.

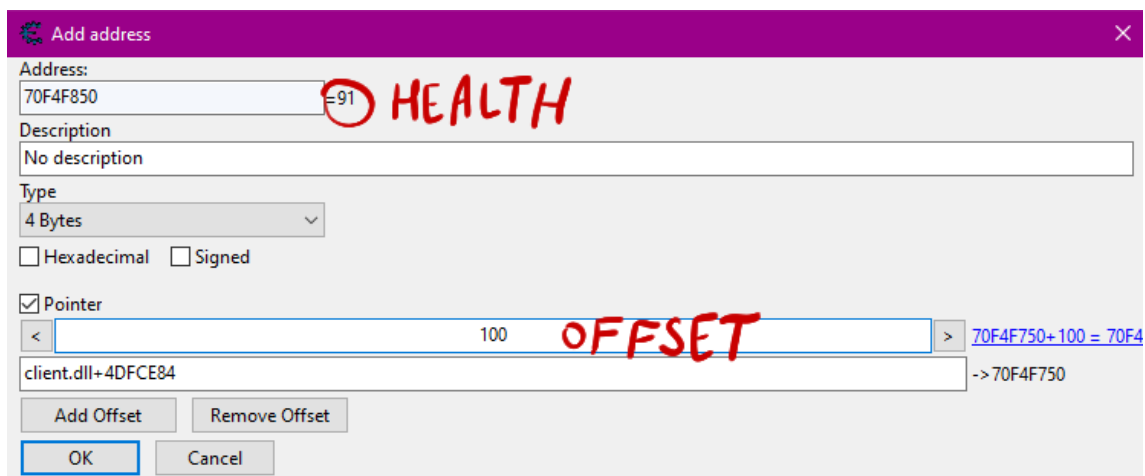


Figure 21 Entity health address

Performing similar manipulations with the local entity (player) will be helpful to find the entity list (see Appendix 2 – Getting Entity List).[29]

4.2.2 Using ReClass.NET to Reverse CS: GO Entity List

ReClass.NET is a powerful game-hacking tool that allows users to easily view and modify variables in video games. It ranks as the third most important tool of its kind, behind Cheat Engine and IDA Pro. One of its key features is the ability to display variables in various formats, such as 32-bit hexadecimal, integer, and float, which makes it easy to identify the variables associated with the player's in-game entity. Additionally, ReClass.NET can generate classes with padding that can be copied and pasted into hack source code, allowing users to access variables using class object pointers rather than offsets.[30]

After attaching ReClass.NET to CS: GO process (File -> Attach to Process...), the entity list static address (<client.dll>+4DFFF04) can be passed in, which will then show a memory region.

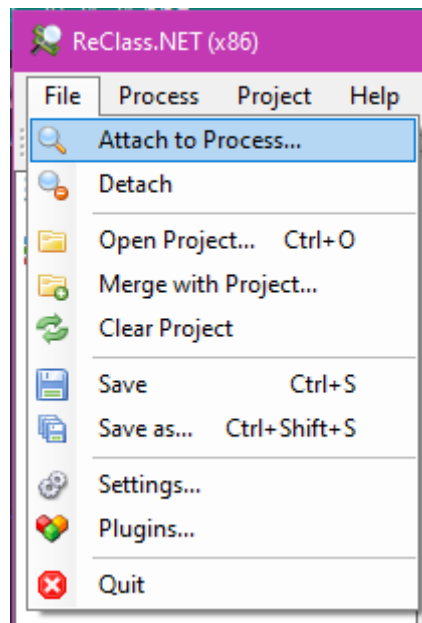


Figure 22 Attach process in ReClass.Net

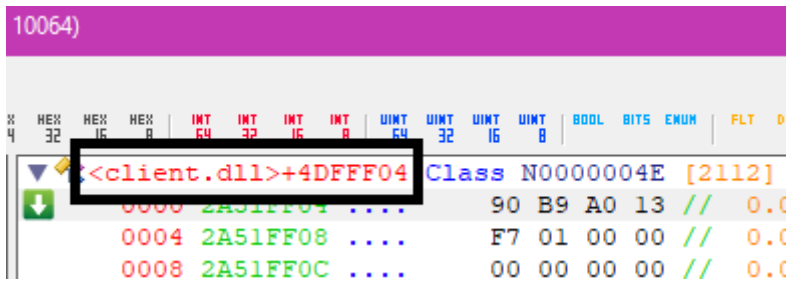


Figure 23 Insert correct memory address

Investigating Source Engine public code, it is obvious that the entity list is a linked list.[31][32] Entity info object has the following structure:

```
class CEntInfo{
    IHandleEntity *entityPtr // 0x0
    int serial_number // 0x4
    CEntInfo *previousEntityPtr // 0x8
    CEntInfo *nextEntityPtr // 0xC
}; // 0x10
```

It is also obvious, that some pointers point to in-game entities, which can be proven by adding or kicking players (bots) from the session since the pointers will also appear and disappear.

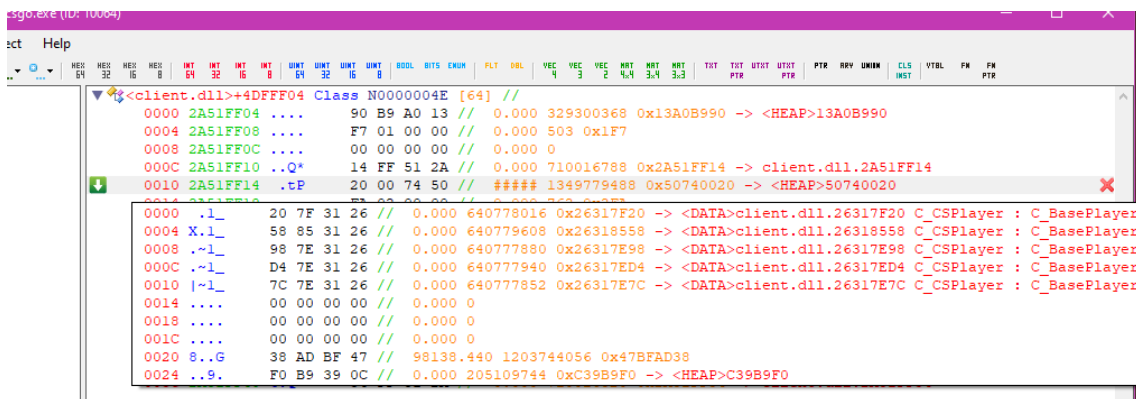


Figure 24 Pointer to the player object

Changing values according to previously acknowledged entity info object structure,

```
0010 2A51FF14 Instance LocalEntity <CEntInfo> //
  400000 Class CEntInfo [16] //
    0000 2A51FF14 Ptr EntityPtr -> 0x50740020 //
    0004 2A51FF18 Int32 SerialNumber = 762 0x2FA //
    0008 2A51FF1C Ptr PreviousEntity -> 0x2A51FF04 //
    000C 2A51FF20 Ptr NextEntity -> 0x2A520314 //
```

Figure 25 Entity Information object

it is now possible to dissect into entity class, where 0x100 offset will show entity health.

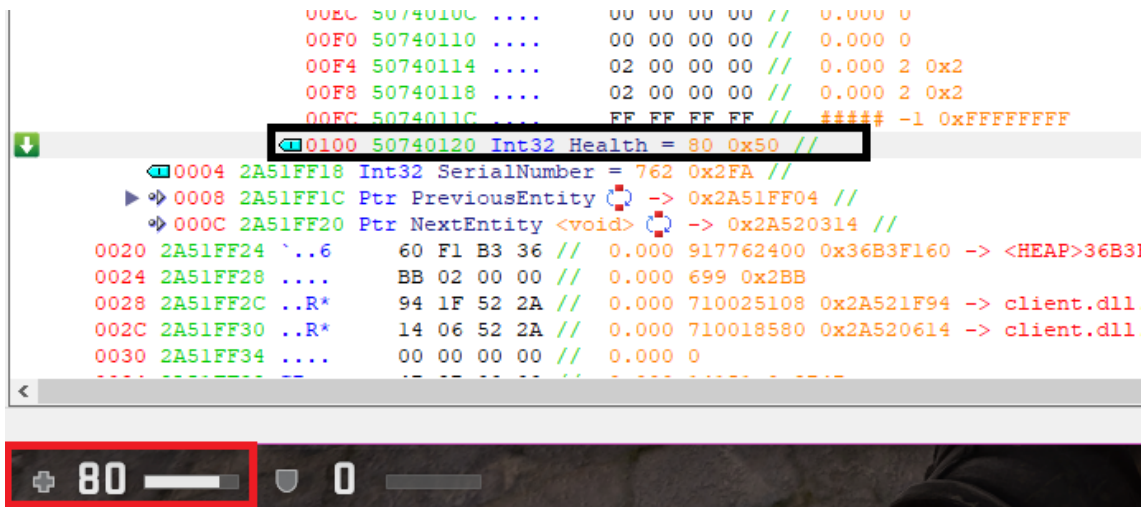


Figure 26 Offset 0x100 health

Now, changing the instance type to an array of entity info objects will allow going back and forth in that linked list.

The reversed structure looks like this:

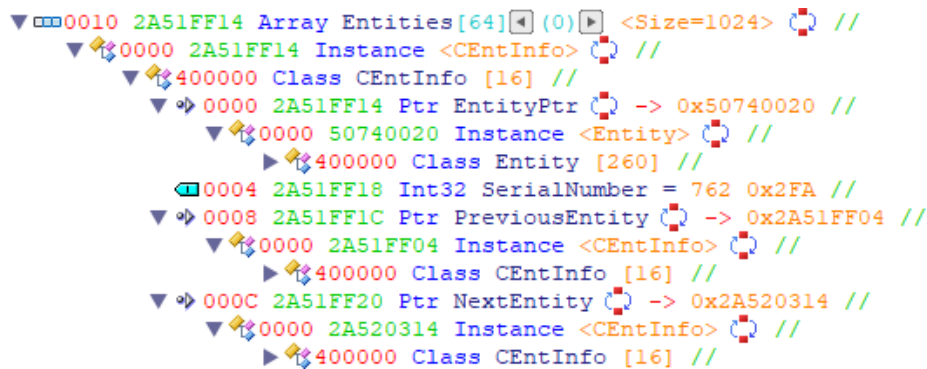


Figure 27 Fully reversed structure

And the generated C++ code:


```

// Created with ReClass.NET 1.2 by KN4CK3R

class CEntInfo
{
public:
    class Entity *EntityPtr; //0x0000
    int32_t SerialNumber; //0x0004
    class CEntInfo *PreviousEntity; //0x0008
    class CEntInfo *NextEntity; //0x000C
}; //Size: 0x0010

class CBaseEntityList
{
public:
    char pad_0000[16]; //0x0000
    class CEntInfo Entities[64]; //0x0010
}; //Size: 0x0410

class Entity
{
public:
    char pad_0000[256]; //0x0000
    int32_t Health; //0x0100
}; //Size: 0x0104

```

Figure 28 Generated C++ code

This code could be pasted into Visual Studio IDE (see Appendix 3 – Show Entity Health DLL Hack), and after some coding and injecting, the following result will be shown:

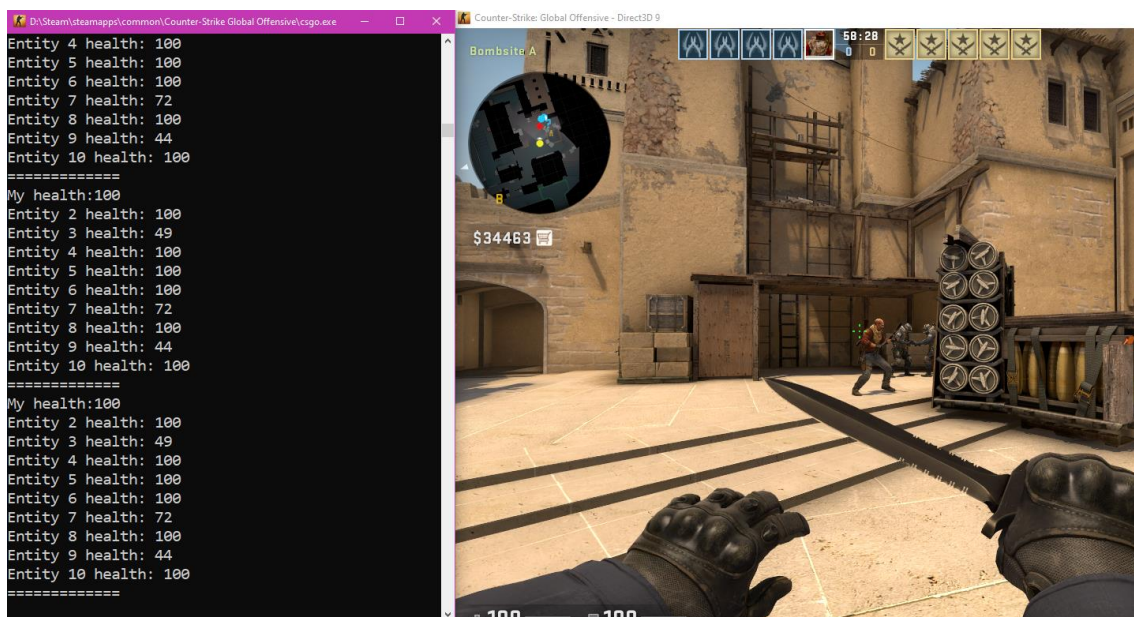


Figure 29 Show a health hack

This is a simple hack that shows entities' health.

4.2.3 Disassembling Binaries and Finding Network Variables Offsets with IDA Pro

To properly display an entity on a client, it is necessary to have certain variables, such as position, angle, or health, transmitted over a network. These variables, known as networked variables, represent the essential properties of the entity.[33]

IDA Pro provides functionality to find offsets of these variables by attaching the “client.dll” game module (see Appendix 4 – Setting Up IDA Pro) and searching for strings.

In the IDA Pro window, the “SHIFT + F12” button combination will generate a list of strings, which can be used to find network variables by pressing “CTRL+F”.

List of CS: GO network variables is available online; as a proof of concept the author has decided to find health and armor variables.[34]

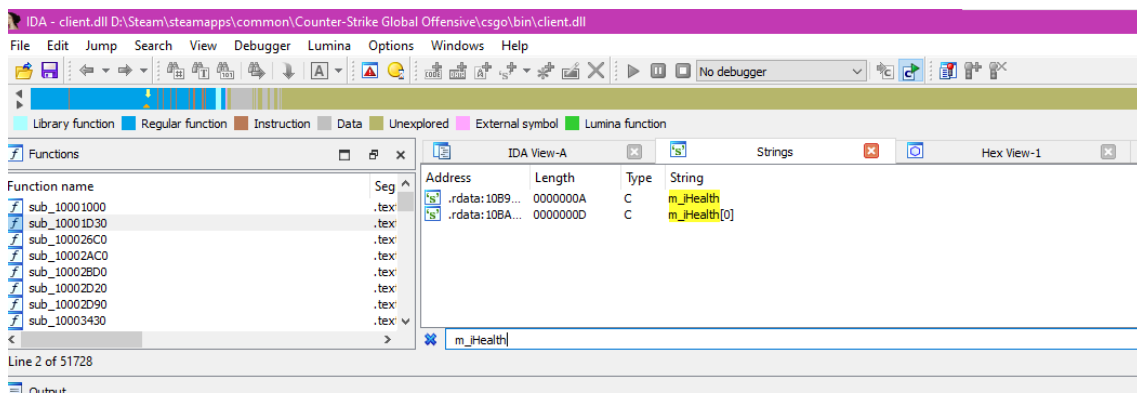


Figure 30 Search in IDA Pro

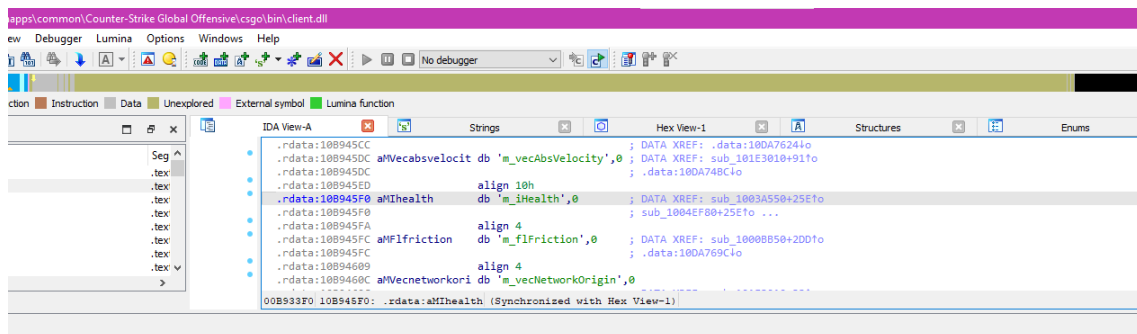


Figure 31 Network variables

Now, jumping to cross-reference to operand will lead to exact instruction and variable offset.

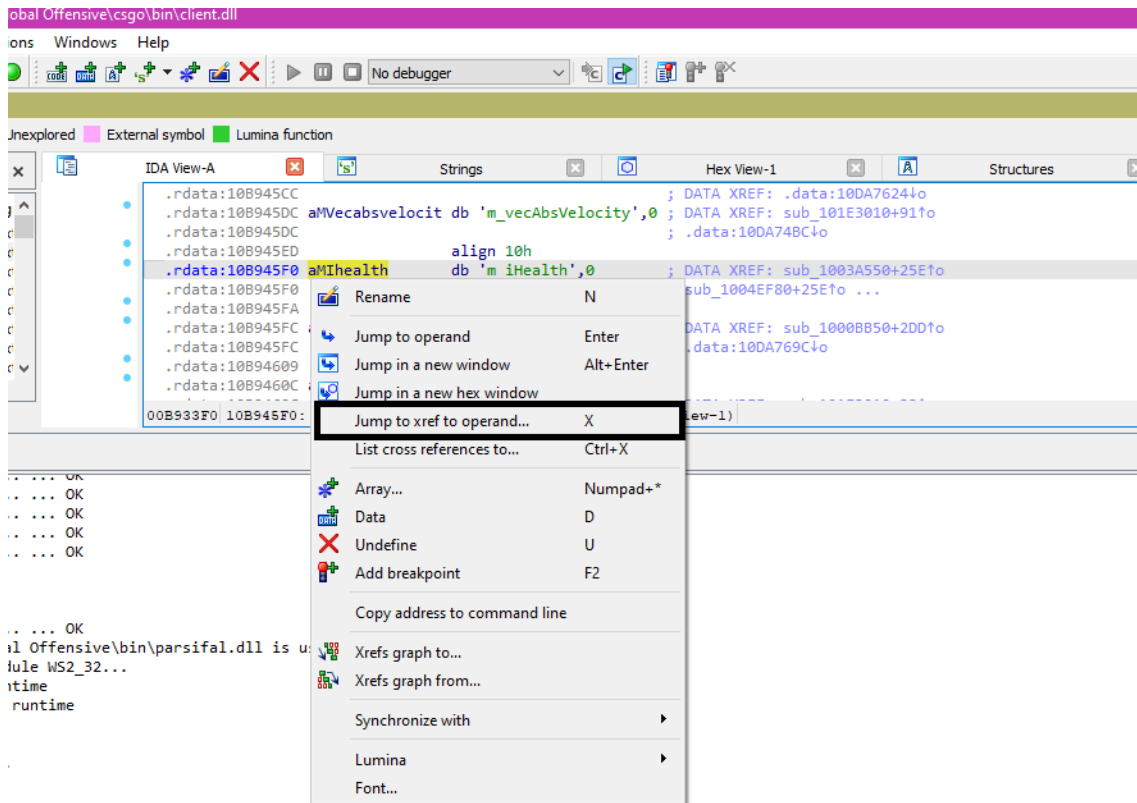


Figure 32 Jump to cross-reference operand



Figure 33 Offset of “m_iHealth” variable

The same can be done with other variables:

```

Strings
Hex View-1
Structures
Enums
mov     dword_15360138, offset aMIclass ; "m_iClass"
mov     dword_15360164, 117C8h
mov     dword_1536013C, 0
mov     dword_15360140, 0
mov     dword_15360158, offset sub_102A2AF0
call    sub_102A2890
mov     ecx, offset dword_153601B0
mov     dword_15360174, offset aMArmorvalue ; "m_ArmorValue"
mov     dword_153601A0, 117CCh <-OFFSET
mov     dword_15360178, 0
mov     dword_1536017C, 0
mov     dword_15360194, offset sub_102A2AF0
call    sub_102A2890
mov     ecx, offset dword_153601EC
mov     dword_153601B0, offset aMangeyeangles ; "m_angEyeAngles"
mov     dword_153601DC, 117D0h
mov     dword_153601B4, 2

```

Figure 34 Offset of "m_ArmorValue" variable

Offsets can be proven by adding them to the local entity address base (client.dll+4DFFF14 + 117CC), it will show armor value.

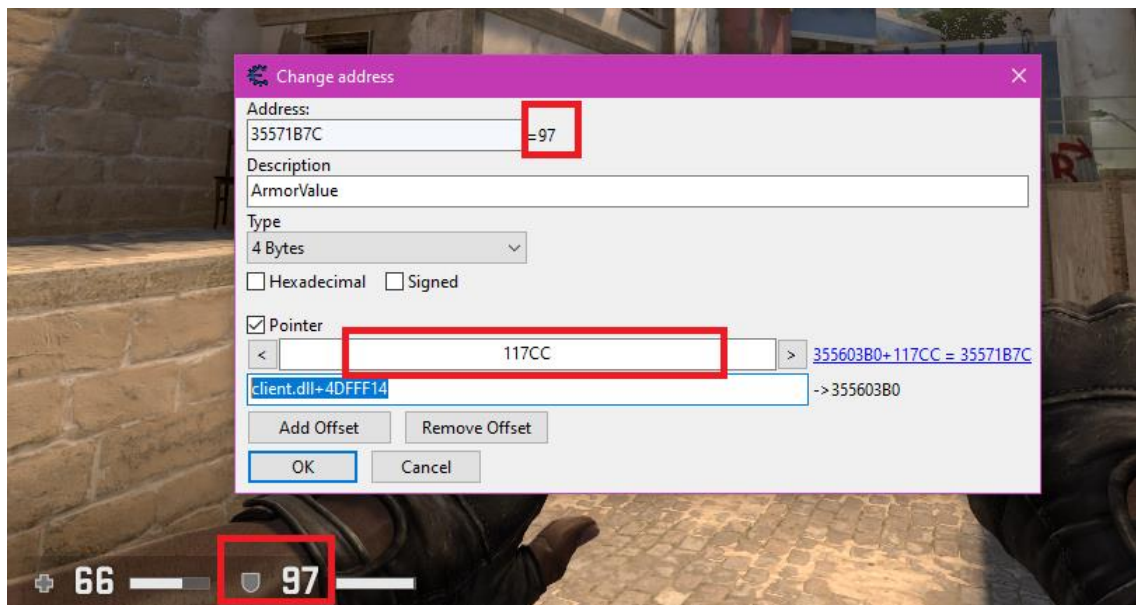


Figure 35 Added armor offset to local player entity

5 Solution

Valve Software made the source code for the Source Engine and other information publicly available, which allows hackers to use special tools and techniques such as reverse engineering to gain an unfair advantage in the game by accessing and manipulating the game's memory structure.[38]

5.1 External Exploit

C++ programming language provides all the necessary features to access the game's memory and change its values. External hacks interact with the game process's memory by using WriteProcessMemory (WPM) and ReadProcessMemory (RPM). To use these functions, hackers must obtain a handle to the process by requesting access through the kernel with OpenProcess and the necessary Process Access Rights, usually PROCESS_ALL_ACCESS. This handle is necessary for RPM and WPM. See “Appendix 5 – Memory Read and Write”.

Since CS: GO has an entity glowing effect inside the game, to create a “wallhack” cheat, a hacker could use in-game variables like team number, glow object manager, glow index, local player, and previously reversed entity list.

After getting the handle to the process, the author assigned memory addresses to the abovementioned variables changed the Boolean value of the glowing effect turned it on, and set RGB values depending on the entity team (red or blue).[43]

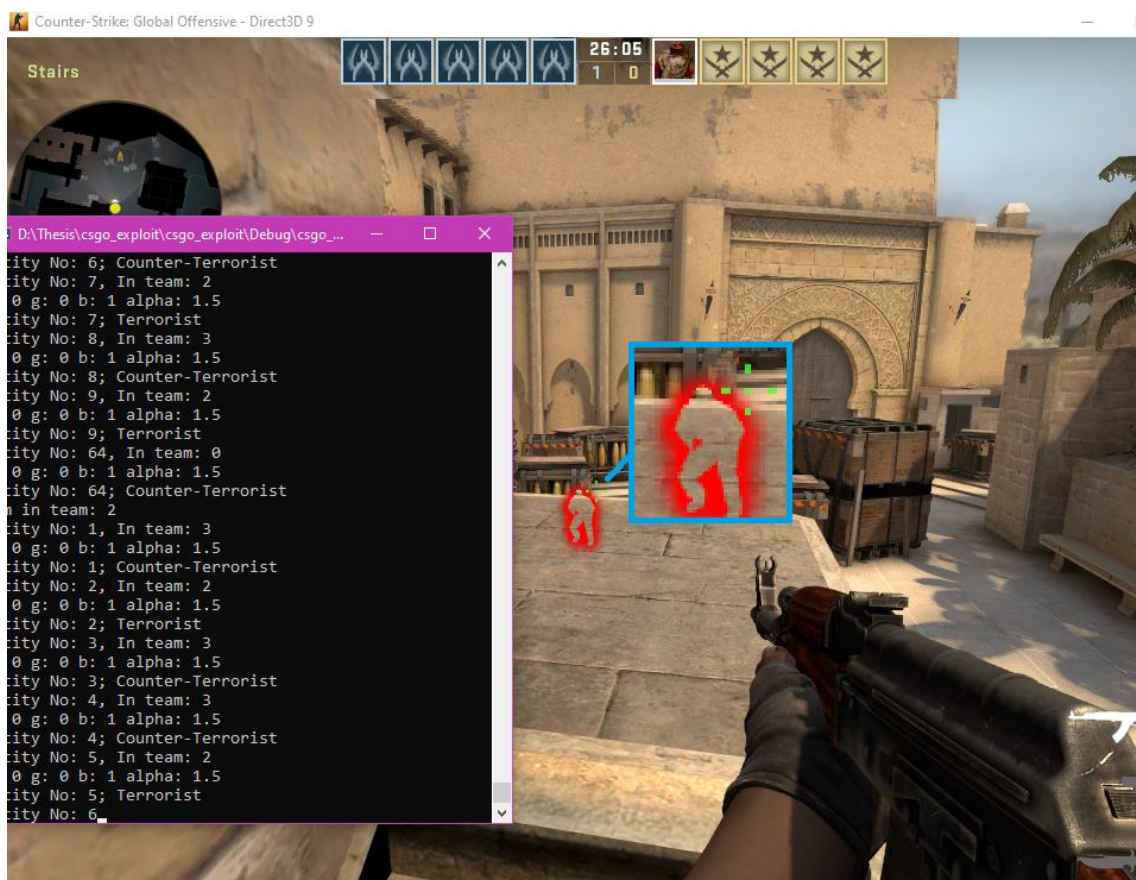


Figure 36 Glow hack

5.2 Internal Exploit

Internal hacks involve injecting a DLL into the game process, which gives direct access to the process's memory for fast and easy manipulation. To make these injected DLLs harder to detect by VAC, hackers can use various injection methods, such as manual mapping.[39]

Although everything done by making an internal hack is achievable by external hacks, an internal hack's main advantage is its performance.

The author's internal exploit, besides the glow hack, has a “bunny hop” hack, which forces a player to jump at the exact moment of hitting the ground. This allows a player to gain more moving speed than usual. “Bunny hop” hack required such variables as `m_fFlags` to check whether the player is staying on the ground, ducking, jumping out of the water, etc., and `dwForceJump` which is a force jump variable specifically.

These flags can be seen in the “Source” Engine's public code [40]:

```
FL_ONGROUND          (1<<0) // At rest / on the ground

FL_DUCKING           (1<<1) // Player is fully crouched

FL_WATERJUMP        (1<<2) // Player jumping out of water

FL_ONTRAIN           (1<<3) // Player is _controlling_ a train,
so movement commands should be ignored by the client during prediction.

FL_INRAIN            (1<<4) // Indicates the entity is standing
in rain

FL_FROZEN            (1<<5) // Player is frozen for 3rd person
camera
```

The “bunny hop” code:

```
val.flag = *(BYTE*)(val.localP + offsets.m_fFlags);

    if (GetAsyncKeyState(VK_SPACE) && val.flag & (1 << 0))
    {
        *(DWORD*)(val.csgoModule + offsets.dwForceJump) = 6; //
force jump is 6 (0110 in binary)
    }
```

The “bunny hop” is activated when the “space” button is being held, and as it can be seen from the abovementioned flags, “val.flag & (1 << 0)” checks whether bit 0 is set, which means player touches the ground, and force jumps if so.[41]

The solution can be found in the official V. Sidorenko GitLab repository.[44]

5.3 DLL Injector

DLL injection is a method of running code within the memory space of another process by forcing it to load a dynamic library. This technique is often used by external programs to alter the behavior of the target program in unexpected ways, such as by intercepting function calls or copying data variables. A program that injects code into processes is called a DLL injector. DLL injectors are used to inject internal hacks into the target process.

Manual mapping is a technique used to load and execute a dynamic-link library (DLL) within the memory of another process. It involves emulating the LoadLibrary function and performing a series of steps:

- Map sections into the target process
- Inject shellcode
- Do relocations
- Fix imports
- Do TLS callbacks
- Call Dll main function

In this thesis, the paper author has created a DLL injector using manual mapping injecting. Manual mapping is often used for bypassing anti-cheat measures because it allows the DLL to be loaded without being visible to ToolHelp32Snapshot or other methods for enumerating loaded modules, such as walking the module linked list in the process environment block (PEB) or using NtQueryVirtualMemory.

The author's manual mapping injector can be found on the project’s GitLab page.[42]

5.4 How Long Can Hackers Stay Undetected?

In CS: GO, when a hacker is coding their hack, and they have done everything the right way, meaning a hacker is not executing memory scanners and disassemblers while VAC is running in the background, they are very safe. 100% safety is never achievable since VAC can update anytime. Game updates will not lead to a ban; however, a hacker would have to freshen up their memory address offsets.

Using publicly available hacks will probably lead to a ban as soon as Valve adds exploit signatures to their database.

Another point is that CS: GO has the “Overwatch” system, where peers check each other's games and can report suspicious players. Thus, hackers should moderately use their hacks. Doing so and updating exploit each time game updates will result in very high safety rates.

5.5 Observation

The author has performed a cheat test by himself, and also cheat was given to another person.

The author has run his cheat with both “bunny hop” and “glow” hacks. Aside from the time for cheat development, total online game time was rough ~10 hours, during this time and about 10 days while offline, no VAC ban was received.

The person that received a copy of the exploit, has played about ~3 hours using it, a week after no VAC ban was received.

5.6 White Hat Point of View

Taking a side of a white hat hacker, the author would say that solution Valve Software should come to is to port Valve online games like Counter-Strike: Global Offensive to their new Source 2 game engine, which would increase game security. The public code of the Source engine is one of the reasons it is possible to develop game hacks that would bypass the Valve Anti-Cheat system.

6 Summary

Cheating in video games has a long-lasting history, firstly cheats were invented by game developers, and only then hackers began to develop their own cheats. Cheating in online games is possible because of the weaknesses in the game code, or its publicity. Due to the publicity of the Source Engine, such games as TF2, Half-Life 2, Left 4 Dead, Counter-Strike: Global Offensive, etc. can be hacked, VAC Anti-Cheat system is hard to stop the kind of cheating shown in this paper, at least at the moment of writing this paper.

This paper was intended to show methods to develop hacks that could get past the Valve Anti-Cheat system. The author of this thesis has performed manipulations with Counter-Strike: Global Offensive game memory utilizing public Source Engine code created both external and internal versions of the hack and coded a dynamic link library injector for the internal hack. The author has shown the use of such hacker tools as Cheat Engine, ReClass.NET, and IDA Pro. The goal to create cheats that work online was accomplished, the cheats successfully bypass the Valve Anti-Cheat system and can be safely run. The author and one more person had played the game for a total of ~13 hours, no VAC ban was received by either of them.

References

- [1] Wolf, Mark J. P. & Perron, Bernard. 2014. The Routledge Companion to Video Game Studies. Routledge, New York. 518pp. [https://books.google.ee/books?redir_esc=y&hl=ru&id=P43HBQAAQBAJ&q=cheating#v=snippet&q=cheating&f=false] [last visited: 23/12/2022]
- [2] Maher, Jimmy. 2012. The Wizardry Phenomenon. [<https://www.filfre.net/2012/03/the-wizardry-phenomenon/>] [last visited: 19/12/2022]
- [3] Computer Gaming World. Vol. 4 no. 1. February 1984. p. 15. [http://www.cgwmuseum.org/galleries/issues/cgw_4.1.pdf] [last visited: 31/12/2022]
- [4] Stafford, Brent & Malone, Michael. Cheat! Pringles Gamers Guide: Brand Integration Case Study. 9pp. [https://brentstafford.com/wp-content/uploads/2015/10/Pringles-Case-Study_vertical_BS_FIN.pdf] [last visited: 16/12/2022]
- [5] Masood, Dawood Khan. 2019. Hacking Online Games using Cheat Engine. [<https://hackhex.com/how-to/hacking-online-games/>] [last visited: 21/12/2022]
- [6] N. Cano, Game Hacking. No Starch Press, 2016. 304pp. [<https://learning.oreilly.com/library/view/game-hacking/9781492017462/>] [last visited: 31/12/2022]
- [7] Wallhax, ESP, 2022. [<https://wallhax.com/what-are-esp-cheats/mem>] [last visited: 02/01/2023]
- [8] Kovidomi, GitHub repository, 2020. [<https://github.com/kovidomi/game-reversing>] [last visited: 23/12/2022]

- [9] GamingSection, Are game hacks illegal? 2019. [<https://gamingsection.net/news/are-game-hacks-illegal/#:~:text=It%20is%20illegal,as%20COD%20etc%20is%20copyrighted>] [last visited: 22/12/2022]
- [10] Elizabeth Wolfe and Brian Ries, 2019. A Fortnite superstar has been banned for life for cheating. [<https://edition.cnn.com/2019/11/06/entertainment/faze-jarvis-fortnite-ban-trnd/index.html>] [last visited: 31/12/2022]
- [11] Wallhax, Aimbots, 2022. [<https://wallhax.com/aimbots/>] [last visited: 16/12/2022]
- [12] Video from YouTube. 2017. Girl live streaming catch cheating in CS: GO. Available at: [<https://www.youtube.com/watch?v=mA5NUOTuRFo>] [last visited: 29/12/2022]
- [13] Gaming Section, 2021. What is a Triggerbot? [<https://gamingsection.net/news/what-is-a-triggerbot/>] [last visited: 21/12/2022]
- [14] Josse Van Dessel, 2021. These are the most infamous VAC-banned CSGO pros [<https://win.gg/news/these-are-the-most-infamous-vac-banned-csgo-pros/>] [last visited: 21/12/2022]
- [15] Ron, GameNews24, 2021 [<https://game-news24.com/2021/09/25/what-is-aim-assist-and-is-it-cheating/>] [last visited: 28/12/2022]
- [16] Steam Docs. Valve Anti-Cheat (VAC) System [<https://help.steampowered.com/en/faqs/view/571A-97DA-70E9-FF74>] [last visited: 31/12/2022]
- [17] Sa1fu, 2022. How to bypass EAC [<https://hackvshack.net/threads/how-to-bypass-eac.733/>] [last visited: 19/12/2022]

- [18] AssaultCube, Official Website [<https://assault.cubers.net>] [last visited: 19/12/2022]
- [19] Timothy Edward Downs and Ron White, How Computers Work, Ninth Edition. Que, 2007. 464pp. [<https://learning.oreilly.com/library/view/how-computers-work/9780789736130/?ar=>] [last visited: 25/12/2022]
- [20] Cheat Engine, Official Website [<https://www.cheatengine.org>] [last visited: 01/01/2023] [last visited: 16/12/2022]
- [21] Rake, Beginner Cheat Engine Tutorial, GuidedHacking, 2017. [<https://guidedhacking.com/threads/beginner-cheat-engine-tutorial-video-guide.9690/>] [last visited: 23/12/2022]
- [22] N. Toppo and H. Dewan, Pointers in C. Apress, 2013. [<https://learning.oreilly.com/library/view/pointers-in-c/9781430259114/?ar=>] [last visited: 26/12/2022]
- [23] Rake, Cheat Engine How To Pointer Scan with Pointermaps, GuidedHacking, 2017. [<https://guidedhacking.com/threads/cheat-engine-how-to-pointer-scan-with-pointermaps.9739/>] [last visited: 17/12/2022]
- [24] V. Sidorenko, GitHub repository, 2022. [<https://gitlab.com/vlsido/thesistrainer>] [last visited: 02/01/2023]
- [25] Rake, How to Bypass VAC, GuidedHacking, 2016. [<https://guidedhacking.com/threads/how-to-bypass-vac-valve-anti-cheat-info.8125/>] [last visited: 19/12/2022]
- [26] M. Lee, C++ programming for the absolute beginner, 2nd ed. Boston (Mass.) [etc.]: Course Technology/Cengage Learning, 2009. [https://www.ester.ee/record=b2754408*eng] [last visited: 25/12/2022]

- [27] Game Hacking Academy, A Beginner's Guide to Understanding Game Hacking Techniques, 2021, 511pp. [<https://gamehacking.academy/GameHackingAcademy.pdf>] [last visited: 19/12/2022]
- [28] G. Balakrishnan and T. Reps, "Analyzing Memory Accesses in x86 Executables," in Compiler Construction, 2004, pp. 5–23. [https://link.springer.com/chapter/10.1007/978-3-540-24723-4_2] [last visited: 31/12/2022]
- [29] Rake, Reverse Engineering, GuidedHacking, 2019. [<https://guidedhacking.com/threads/reverse-engineering-how-to-find-the-csgo-entity-list.13313/>] [last visited: 21/12/2022]
- [30] KN4CK3R, ReClass.NET GitHub repository, 2019. [<https://github.com/ReClassNET/ReClass.NET>] [last visited: 29/12/2022]
- [31] Valve Corporation, Source Engine SDK Entity List, 2013. [https://github.com/ValveSoftware/source-sdk-2013/blob/master/mp/src/game/shared/entitylist_base.h] [last visited: 22/12/2022]
- [32] Microsoft, Using Singly Linked Lists, 2021. [<https://learn.microsoft.com/en-us/windows/win32/sync/using-singly-linked-lists>] [last visited: 17/12/2022]
- [33] Valve Corporation, Networking Entities, 2019. [https://developer.valvesoftware.com/wiki/Networking_Entities] [last visited: 28/12/2022]
- [34] Namazso & zbe, NetVar & DataProp dump, 2017. [<https://www.unknowncheats.me/forum/counterstrike-global-offensive/211333-current-netvar-dataprop-dump-classes-header-format.html>] [last visited: 28/12/2022]

- [35] Datadome, What is a botnet attack and how does it work? 2022
[<https://datadome.co/learning-center/what-is-botnet-how-does-botnet-attack-work/>] [last visited: 28/12/2022]
- [36] Josse Van Dessel, 2021. These are the most infamous VAC-banned CSGO pros
[<https://win.gg/news/these-are-the-most-infamous-vac-banned-csgo-pros/>] [last visited: 23/12/2022]
- [37] Wallhax, Triggerbot, 2022. [<https://wallhax.com/hacks/csgo/triggerbot/>] [last visited: 26/12/2022]
- [38] Valve Corporation, Source Engine SDK 2013 edition, Github, 2013.
[<https://github.com/ValveSoftware/source-sdk-2013>] [last visited: 23/12/2022]
- [39] UnKnoWnCheaTs, “Best resources to learn manual mapping injection?” thread, 2020.
[<https://www.unknowncheats.me/forum/general-programming-and-reversing/404055-resources-learn-manual-mapping-injection.html>] [last visited: 16/12/2022]
- [40] Valve Corporation, Source Engine SDK Constants, Github, 2013.
[<https://github.com/ValveSoftware/source-sdk-2013/blob/master/mp/src/public/const.h>] [last visited: 03/01/2023]
- [41] Microsoft, C Bitwise Operators, 2022. [<https://learn.microsoft.com/en-us/cpp/c-language/c-bitwise-operators?view=msvc-170>] [last visited: 25/12/2022]
- [42] V. Sidorenko, Manual Mapping Injector, GitLab, 2022.
[<https://gitlab.com/vlsido/manual-mapping-injector>] [last visited: 05/01/2023]
- [43] V. Sidorenko, External CS: GO Exploit, GitLab, 2022.
[<https://gitlab.com/vlsido/external-csgo-exploit>] [last visited: 05/01/2023]

- [44] V. Sidorenko, Internal CS: GO Exploit, GitLab, 2022.
[<https://gitlab.com/vlsido/internal-csgo-exploit>] [last visited: 05/01/2023]
- [45] V. Sidorenko, External CS: GO Show Health Hack, GitLab, 2022.
[<https://gitlab.com/vlsido/csgo-show-health>] [last visited: 05/01/2023]
- [46] UnKnoWnCheaTs, Game Hacking Web-Forum
[<https://www.unknowncheats.me>] [last visited: 05/01/2023]
- [47] D. Georgiev, “What Is a White Hat Hacker?”, 2022.
[<https://techjury.net/blog/what-is-a-white-hat-hacker>] [last visited: 05/01/2023]

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis¹

I Vladislav Sidorenko

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "A Method for Bypassing the Valve Anti-Cheat System in Video Games" , supervised by Kaido Kikkas
 - 1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
 - 1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

24.12.2022

¹ The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – Getting Entity List

Firstly, Cheat Engine scans the local player's 100 health, then proceed to damage the local player and scan for new values until the minimum amount of addresses is left.

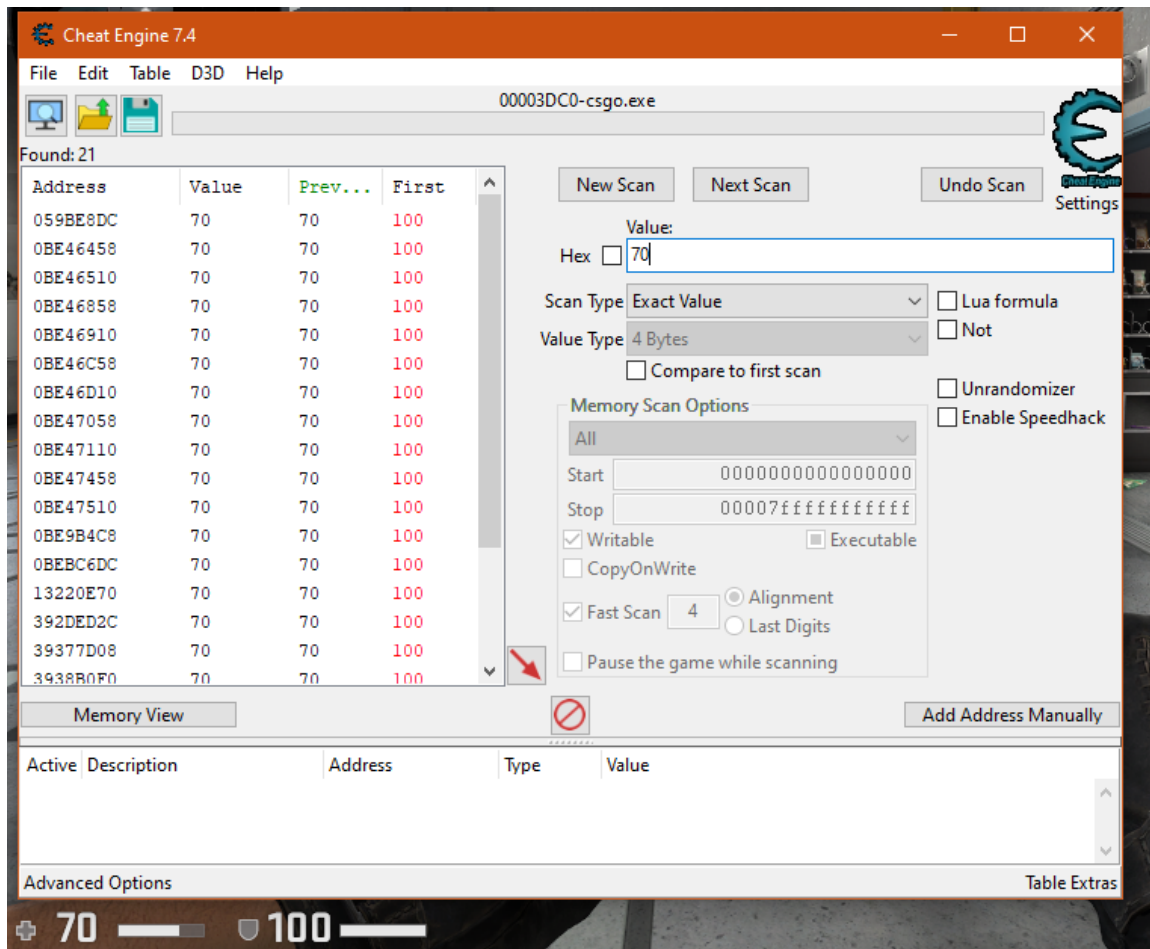


Figure 2-1 Cheat Engine health addresses

Find the most appropriate address and check what accesses it.

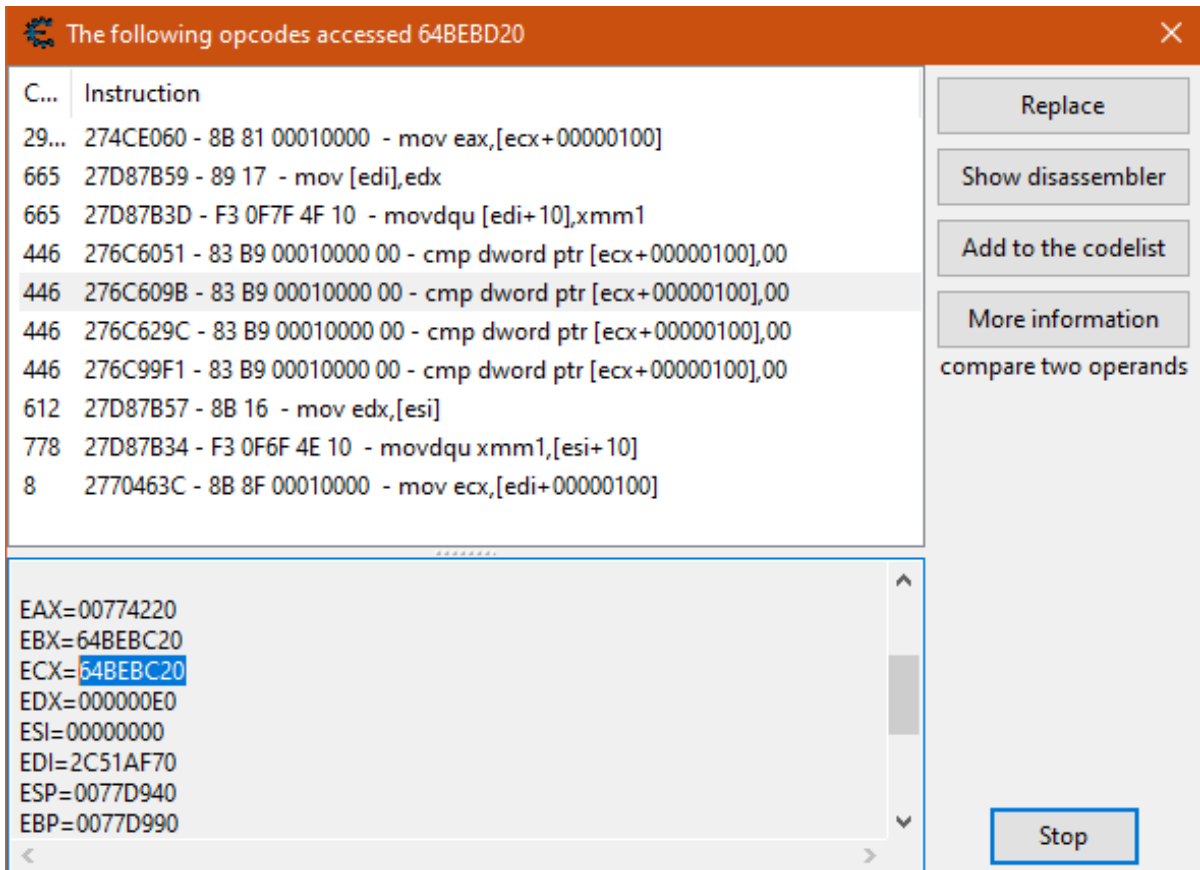


Figure 2-2 Opcodes that access address

Scan the registered HEX address, and find a static address, the right one should be almost similar to that of the entity object, thus if the entity object is **4DFFF24**, the local entity object is most likely **4DFFF14**¹.

¹ The addresses have changed since the last game update. Now, at the moment of writing this paper, the entity object is 4DFFF24, and the local entity object is 4DFFF14.

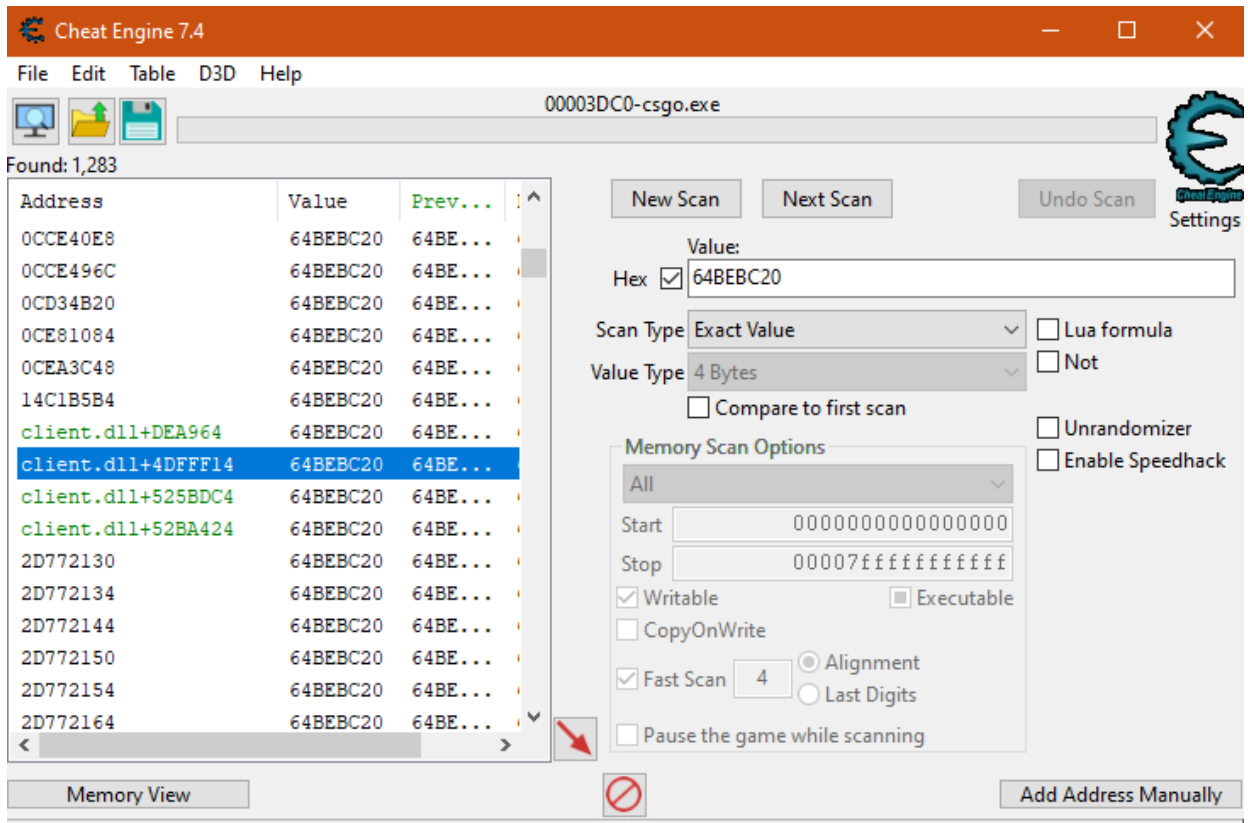


Figure 2-3 Static addresses

Choosing “Find out what accesses this address” will show the entity list static address, which is “**client.dll+4DFFF04**”.

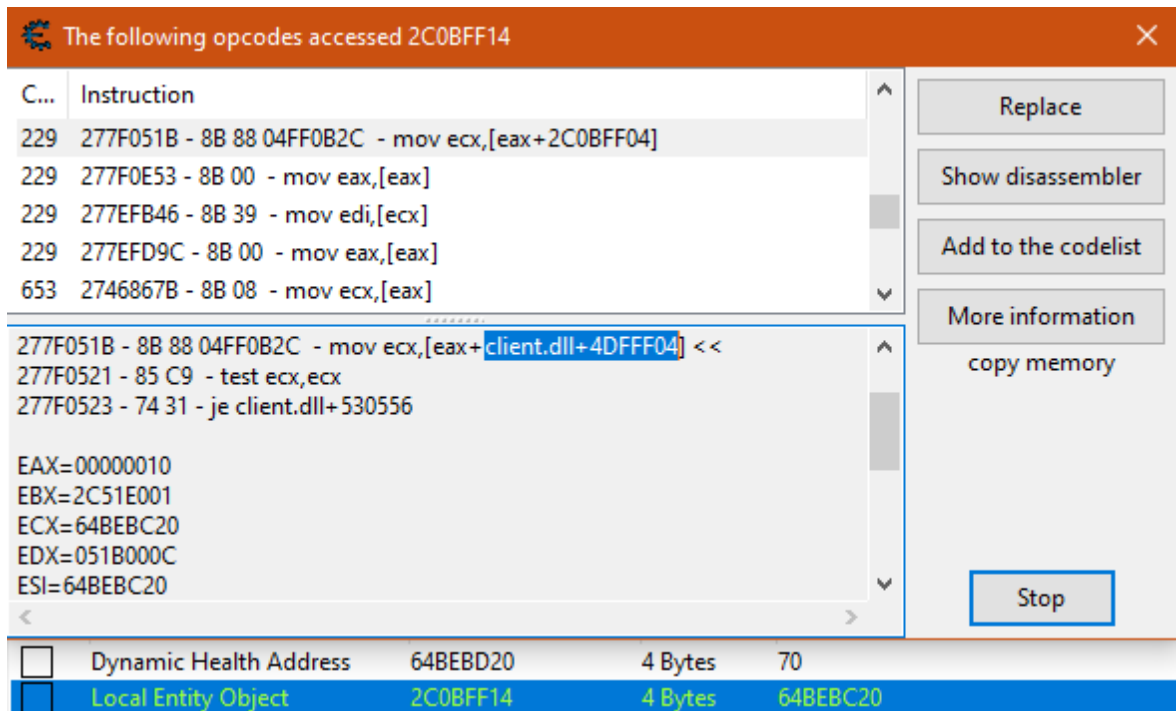


Figure 37 Entity list static address

Appendix 3 – Show Entity Health DLL Hack

After creating a DLL project in Visual Studio 2022 and pasting previously found code as an “ent.h” header, under project properties following options have to be set:

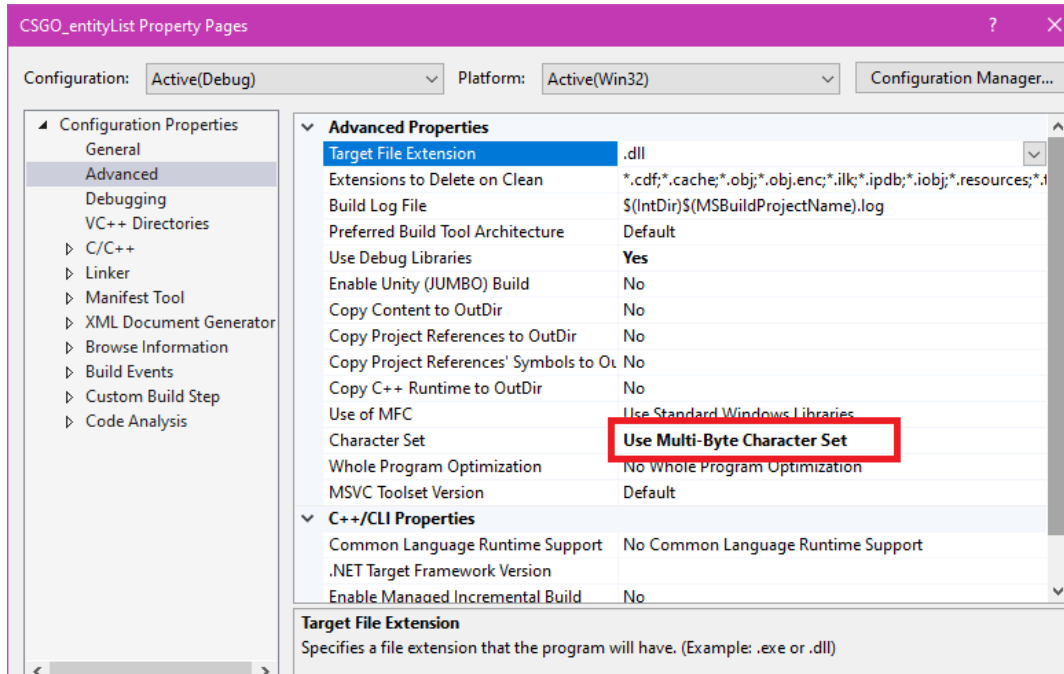


Figure 3-1 DLL project settings part I

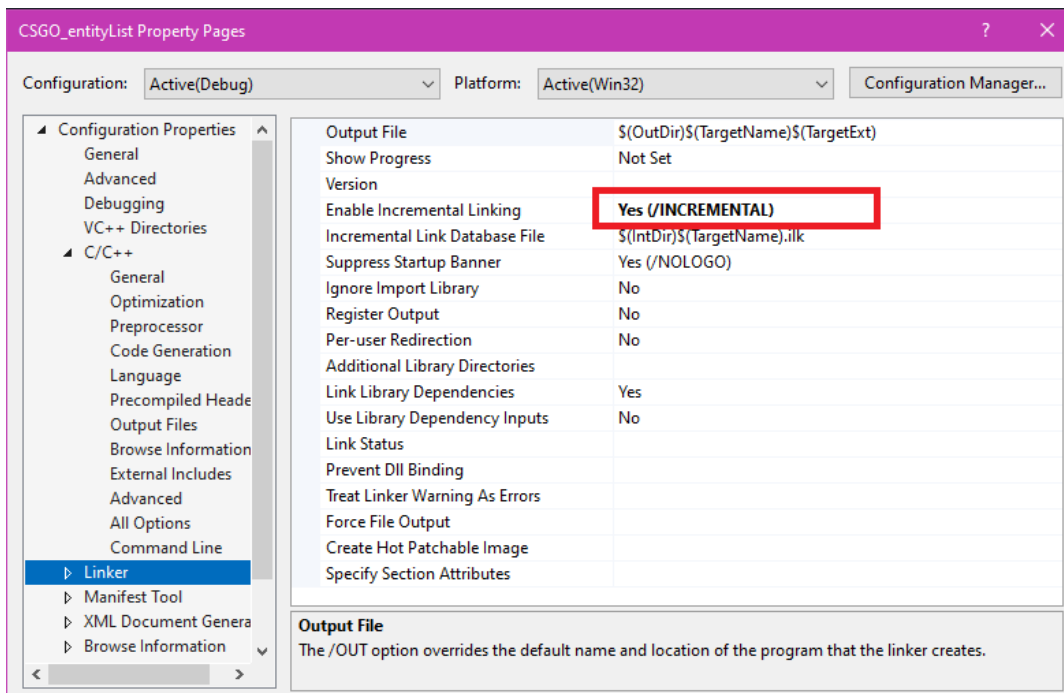


Figure 3-2 DLL project settings part II

Besides, the project has to be built in x86 mode, since the game is x32 bit.

```
#include "stdafx.h"
#include <iostream>
#include "ent.h"
#include <Windows.h>

struct vars
{
    DWORD csgoModule;
} vars;

void hack(HMODULE hModule)
{
    //Create Console
    AllocConsole();
    FILE* f;
    freopen_s(&f, "CONOUT$", "w", stdout);

    // Assign module HEX value to a variable
    vars.csgoModule = (DWORD)GetModuleHandle("client.dll");
    // Add entity list offset to csgo module
    CBaseEntityList* entityList = (CBaseEntityList*)(vars.csgoModule
+ 0x4DFFF04);

    while (true)
    {
        int x = 1;
        for (auto entity : entityList->Entities)
        {
            if (entity.EntityPtr)
            {
                if (x == 1)
                {
                    std::cout << "My health:" <<
entity.EntityPtr->Health << std::endl;
                }
                else {
                    std::cout << "Entity " << x << " health:
" << entity.EntityPtr->Health << std::endl;
                }

                x += 1;
            }
        }

        std::cout << "======" << "\n";

        Sleep(1000);
    }
}
```

```

        fclose(f);
        FreeConsole();
        FreeLibraryAndExitThread(hModule, 0);
    }

    BOOL APIENTRY DllMain(HMODULE hModule,
        DWORD ul_reason_for_call,
        LPVOID lpReserved
    )
    {
        switch (ul_reason_for_call)
        {
            case DLL_PROCESS_ATTACH:
            {
                HANDLE hThread = nullptr;
                hThread = CreateThread(nullptr, 0,
                    (LPTHREAD_START_ROUTINE)hack, hModule, 0, nullptr);
                if (hThread)
                {
                    CloseHandle(hThread);
                }
            }

            case DLL_THREAD_ATTACH:
            case DLL_THREAD_DETACH:
            case DLL_PROCESS_DETACH:
                break;
        }
        return TRUE;
    }
}

```

The project is available in the GitLab repository.[45]

Appendix 4 – Setting Up IDA Pro

In IDA, select “New” to attach the “client.dll” file, which is located under **D:\Steam\steamapps\common\Counter-Strike Global Offensive\csgo\bin\client.dll** (path may be different).

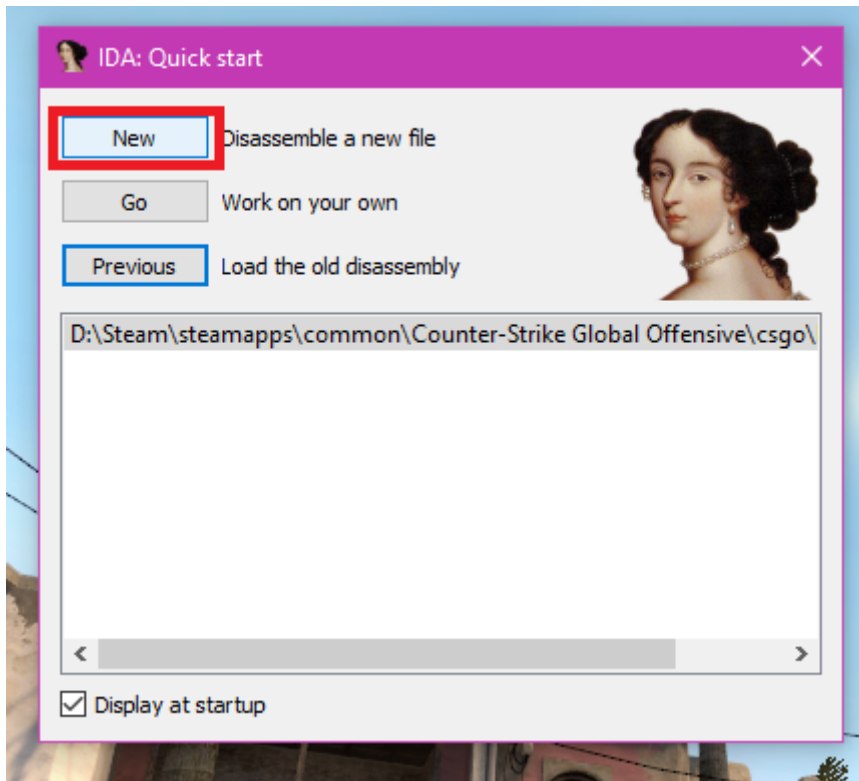


Figure 4-1 IDA Pro Quick start screen

IDA will ask for the **parsifal.dll** file, which is located under **D:\Steam\steamapps\common\Counter-Strike Global Offensive\bin** (the path may be different).

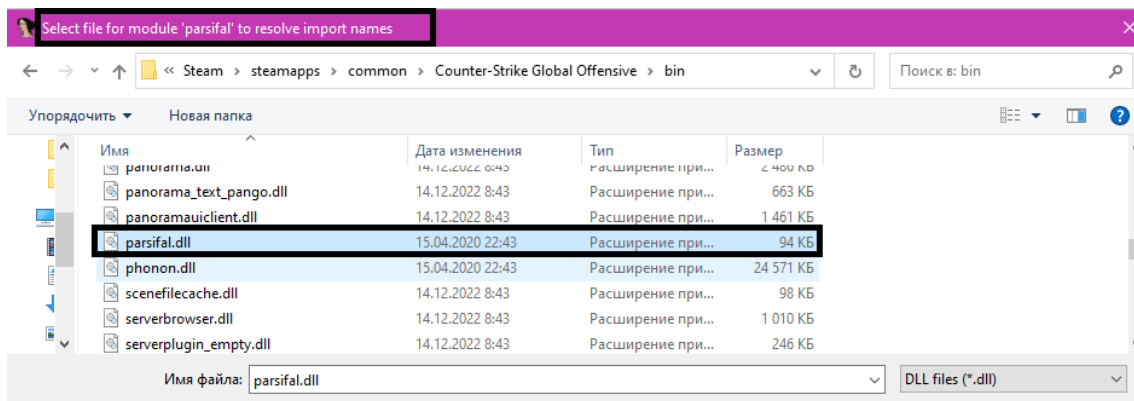


Figure 4-2 Select the "parsifal" module

After that, IDA will ask to look for another file and dismiss this confirmation window.

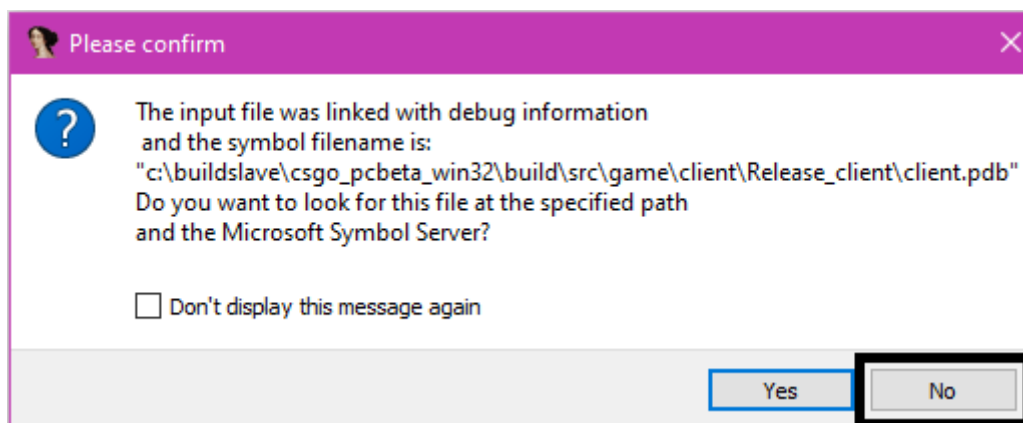


Figure 4-3 Dismiss this window

The preparation of IDA Pro is done.

Appendix 5 – Memory Read and Write

Contents of “ReadWriteMem.h” class:

```
#pragma once
#include <Windows.h>
#include <vector>
class ReadWriteMem
{
public:
    ReadWriteMem();
    ~ReadWriteMem();
    template <class val>
    val readMemory(uintptr_t addr)
    {
        val x;
        ReadProcessMemory(handle, (LPBYTE*)addr, &x, sizeof(x),
NULL);
        return x;
    }
    template <class val>
    val writeMemory(uintptr_t addr, val x)
    {
        WriteProcessMemory(handle, (LPBYTE*)addr, &x, sizeof(x),
NULL);
        return x;
    }

    uintptr_t getModule(uintptr_t, const wchar_t*);
    uintptr_t getProcess(const wchar_t*);
    uintptr_t getAddress(uintptr_t, std::vector<uintptr_t>);

private:
    HANDLE handle;
};
```

Contents of “ReadWriteMem.cpp”:

```
#include "ReadWriteMem.h"
#include <TLHelp32.h>
#include <iostream>
#include <iomanip>
ReadWriteMem::ReadWriteMem(){handle = NULL;}
ReadWriteMem::~ReadWriteMem()
{
    CloseHandle(handle);
}
uintptr_t ReadWriteMem::getProcess(const wchar_t* proc)
{
    HANDLE hProcessId = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS,
0);
```

```

uintptr_t process;
PROCESSENTRY32 pEntry;
pEntry.dwSize = sizeof(pEntry);
do
{
    if (!_wcsicmp(pEntry.szExeFile, proc))
    {
        process = pEntry.th32ProcessID;
        CloseHandle(hProcessId);
        handle = OpenProcess(PROCESS_ALL_ACCESS, false,
process);
    }
} while (Process32Next(hProcessId, &pEntry));
return process;
}
uintptr_t ReadWriteMem::getModule(uintptr_t procId, const wchar_t*
modName)
{
    HANDLE hModule = CreateToolhelp32Snapshot(TH32CS_SNAPMODULE |
TH32CS_SNAPMODULE32, procId);
    MODULEENTRY32 mEntry;
    mEntry.dwSize = sizeof(mEntry);
    do
    {
        if (!_wcsicmp(mEntry.szModule, modName))
        {
            CloseHandle(hModule);
            return (uintptr_t)mEntry.hModule;
        }
    } while (Module32Next(hModule, &mEntry));
    return 0;
}
uintptr_t ReadWriteMem::getAddress(uintptr_t addr,
std::vector<uintptr_t> vect)
{
    for (int i = 0; i < vect.size(); i++)
    {
        ReadProcessMemory(handle, (BYTE*)addr, &addr,
sizeof(addr), 0);
        addr += vect[i];
    }
    return addr;
}

```