GHB1 - Start Here Beginner Guide to Game Hacking

Want to learn game hacking? Great, you're in the right place. GH is the absolute best place to learn game hacking. We have more resources, that are better organized and cater especially to beginners. We go in depth and explain every piece of the game hacking puzzle. 7 years of work have gone into providing you the absolute best learning experience.

Our huge game hacking course, named the Game Hacking Bible has 4 books:

1. GHB1 - Start Here Beginner Guide to Game Hacking
2. GHB2 - Beginners Guide To Reverse Engineering
3. GHB3 - Intermediate Guide to Game Hacking
4. GHB4 - Anti-Debug, AntiCheat & Kernel Mode

But there is something really important that you must understand before you get started:

If you're an intelligent adult, everything you need to get started is right here!

But please do not be naïve and think you can hack games in a few hours or download a source code and be hacking a game a few hours later. Please read this guide and trust me, I have 8 years experience running this forum and helping thousands of people. If you're not committed to learning, you will fail. Do not waste your time, or ours.

If you are not interesting in spending years of your life coding, reverse engineering and debugging then game hacking is not for you.

The biggest problem we see is people focusing on the result (the hack) and not the journey (learning). It's a good life lesson in general, if you focus on the journey the result will come, and often times the reward is less sweet without the

pain and suffering of the journey

LEARN HOW TO HACK FIRST, HACK ANY GAME YOU WANT LATER

Do not try to learn to hack on a complicated game, use something easy such as Assault Cube, CSGO or COD4. If you do a complicated game you won't learn or accomplish anything. Once you learn how to hack, you can hack any game you want with extreme ease. But if you attempt to learn how to hack on a game you want to hack on, you will fail. 99% of users fail because they do not take this advice. Hacking new games is 100x harder than hacking simple old games like Assault Cube. Trying to hack a new game with less than 6 months experience is really stupid and a complete waste of time.

People frequently ask "what games should I learn on?" and we have been saying the same thing for 7 years:

Assault Cube -> CSGO -> COD4 -> Sauerbraten x64 -> Whatever You Want

With this list, you will learn Cube Engine, Source Engine, Quake Engine & x64. They're all native, easy to learn and have tons of resources available so you don't get stuck.

Game Hacking Requirements:

- Intelligence
- Hard Work & Determination
- Learning by yourself
- Lots of time & lots of reading

If you don't meet these requirements, **stop here**, you're not going to be successful.

Still think you have what it takes? Then please keep reading. The above requirements + this forum are all you need.

Basic Idea:

- Learn Cheat Engine
- Learn C++ or C#
- Watch GH YouTube Tutorials
- Practice

Basic Timeline for Learning:

Learning Game Hacking is extremely time consuming, you must practice and get experience using each skill for weeks/months at a time.

1 week	Learn the absolute basics of using Cheat Engine
2 – 3 weeks	Learn enough CE to use all it's features, how to find pointers etc…
2 months	Learn C++/C# well enough to make a hello world and know the absolute basics of coding
3 – 6 months	Experienced enough to make very basic trainers in C++/C# based on stuff you find in Cheat Engine
6 – 12 months	Experienced enough to make hacks without pasting, basic aimbot & ESP
1 - 1.5 Years	Intermediate coding/reversing skills to hack any game you want without anticheat
1.5 - 2 Years	Start reversing anticheat

Serious Warning

If you do not follow these steps you will be missing the necessary knowledge and experience to advance to the next level. I am trying to help you, but if you refuse to listen you are choosing failure. GH is a resource for people who are serious about game hacking, if you're not serious you shouldn't be here.

ALL TUTORIALS USE ASSAULT CUBE v1.2.0.2. DOWNLOAD THE CORRECT VERSION

Legit Step by Step Guide for Beginners:

If you skip any of these tutorials you will be banned. We are not babysitters.

1. How to Solve Problems & Achieve Goals
2. Stop Pasting - Focus on the Basics

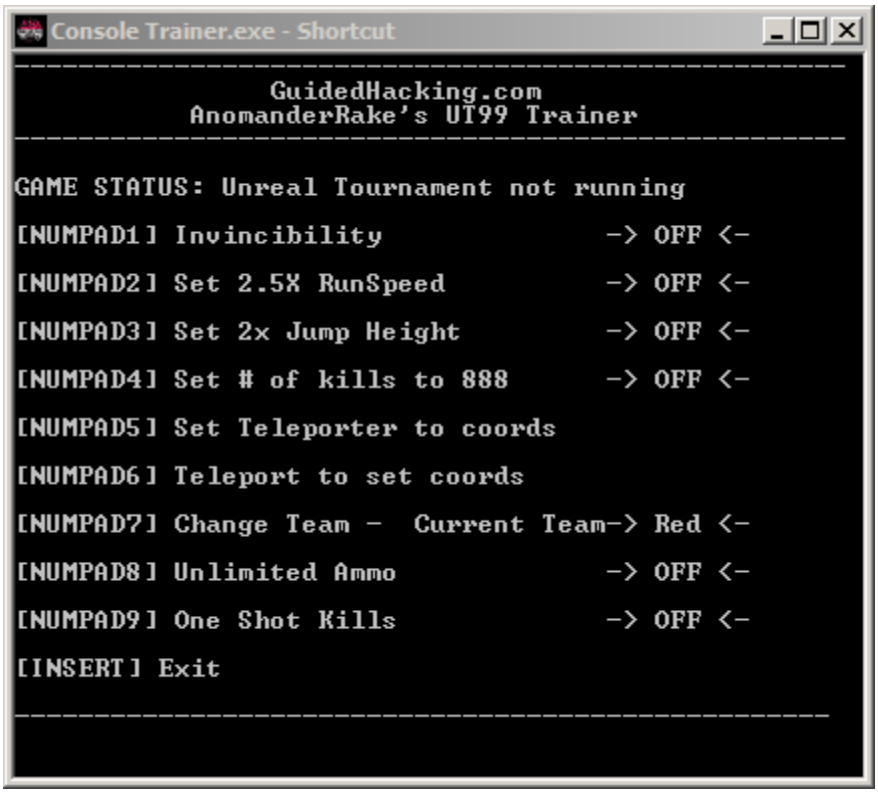
3. Squally CS420 Game Hacking Course
4. Game Hacking FAQ - Frequently Asked Questions
5. Cheat Engine Tutorial Video Guide
6. How To Hack Any Game - Cheat Engine
7. How to Find Position Coordinates with Cheat Engine
8. How to Find View Angles with Cheat Engine
9. Cheat Engine How to Pointer Scan with Pointermaps
10. Game Hacking with Reclass Tutorial Video 1
11. Learn C++ with Step by Step Guide
    1. Windows API Beginner Guide
12. Understanding Strings Unicode, TCHAR, MBCS
13. Get Module Base Address Tutorial dwGetModuleBaseAddress
14. Source Code - FindDMAAddy - C++ Multilevel Pointer Function
    1. Quick Tip: Visual Studio Run as Administrator + Manifest File
15. How to Hack Any Game Tutorial C++ Trainer #1 - External
16. How to Hack Any Game Tutorial C++ Trainer #2 - External v2
17. How to Hack Any Game Tutorial C++ Trainer #3 - First Internal
18. Simple C++ DLL Injector Source Code
19. How to Debug Your Hack or Aimbot with Visual Studio
20. Game Hacking with Reclass Tutorial Video 2
21. C++ External Detour / Hooking Function Tutorial
22. WTF are Multi Level Pointers Tutorial
23. How to find Entity List Pointer in Assault Cube

Practice Time

If you finished all the tutorials in GHB1, you now have all the skills to make simple trainers. Before you proceed, did you really do every video? Or did you just skim over it? If you aren't actually DOING the tutorials, you're not learning anything. You need to actually do the things in the videos, multiple times and gain experience.

Is there something you don't 100% understand? Go back and figure it out. Don't skip ahead until you're ready. Practice what you learn, add a few features to the trainer, go read through a few tutorials from our list if you get bored: MUST DO Game Hacking Tutorials

Here's an example of something I made at this stage in my training:



Might not look like much, but I learned a lot. Challenge yourself to use the skills you have already learned to make something new. But don't try to hack some crazy new game, you're not ready for that. After brushing up on your skills and getting some practice it's time to move on to the next book:

GHB2: Beginner's Guide to Reverse Engineering

Guide - GHB2 - Beginners Guide To Reverse Engineering

Our Most Important Video

Our most important video from this guide is this one, it answers 99% of questions that are asked:

Can't find what you need?

@iPower made this great guide to our best content: MUST DO GH Tutorials!
When you search the forum, use generic terms like "CSGO aimbot" and tick the "search titles only" box

Misc / Resources:

Each game hacking topic has it's own section: Cheat Engine, Aimbot, ESP etc...
Each section of this forum has sticky threads and guides, **READ THEM!**

Two Other "Start Here" guides come after this post:
Solaire's Start Here Thread
Crazywink's Start Here Thread
Erarnitox's Start Here Thread

Guide How to Solve Problems & Achieve Goals

In game hacking, reverse engineering and life in general you will always have problems you need to solve. Learning how to solve problems and achieve goals is something everyone needs to learn how to do. You cannot rely on other people to solve your problems, you need to become independent by learning to solve problems yourself. You need to be able to fix 99% of problems yourself.

This couldn't be more true regarding learning game hacking. This might seem like an odd thread for GH but we get many young people here that have not yet learned these skills. Young people are dependent on their family and other people for everything, which is why they always come to GH begging, that's all they know. Look at every game hacking forum and they're filled with the stupidest posts you will ever see, from the most pathetic, ignorant and belligerent idiots on the planet.

Your growth as an adult comes when the you realize you aren't going to get what you want unless you get it yourself! When you first achieve goals without help from other people, you will learn the importance of independence, causing you to seek it out and become a self sufficient adult. All the best game hackers in the world who you idolize, all realized this and made themselves successful, and so can you!

You cannot expect other people to solve your problems.

How to Solve Problems

Use the scientific method! This is supremely important when reverse engineering.

Scientific Method

1. Question
2. Hypothesis
3. Experiment
4. Observation
5. Analysis
6. Conclusion

The Experiment & Analysis section are the most important for us to touch on. The following is an example (don't worry you don't need to understand it).

Question
How do I access the ammo variable dynamically?

Hypothesis
Perhaps there is ammo variable inside the player class, if I can access the player object I can then access the ammo variable

Experiment
Find the player object, search for ammo variables inside of it. Compare and contrast the offsets and addresses of both the health variable and the ammo variable and how they are accessed.

Observation
We open cheat engine, find the health address and ammo address, use "find what accesses" to see how they are used.

We discover that the offsets are not offset from the same object address for each of these variables. In addition, when we change guns, the ammo address changes.

We reform our hypothesis
Perhaps there is a weapon object which contains the ammo variable, perhaps there is a pointer to the current weapon object in the player class.

How do you form this hypothesis?
Pretend you are the developer and come up with your own logic for managing multiple weapons and weapon related variables. How do you keep track of the current weapon you're using so you decrease from the correct ammo variable? Come up with a couple different logical methods that might make sense, and then test them. This is indeed how I would do it when you consider Object Oriented Programming, I would create dynamic objects and keep track of them by using pointers and as the owner of the weapon is the player, I would put the pointer in the entity object.

Reformed Experiment:
Take the address of the ammo variable, subtract the offset, this will yield the address of the weapon object and then we will reverse engineer this structure from here.

Observation & Analysis
The structure which ammo is in, is indeed the weapon object. We find other variables related to the weapon in it. We also notice it is in an array of other weapon objects which have the same structure and act similarly. When we trace back each ammo address to a dynamic multilevel pointer for the current weapon, and then change this weapon, we notice a pointer in the player class which always points to the current weapon ammo.

Conclusion
The game keeps track of the ammo dynamically by using a pointer to the current weapon object which is inside an array of weapon objects. When you switch weapons, the pointer changes to point at the correct weapon. The ammo variable for this weapon and other weapon variables are accessed from this address by relative offset.

To access the ammo variable dynamically we need only calculate this multilevel pointer at run time and it will always point at the correct ammo address.

See that? We created a hypothesis, tested it and when it was wrong, we formed a new one and tested it. The second hypothesis was proven. But the scientific method holds that, all proven hypotheses require rigorous testing and re-testing as additional knowledge and evidence becomes available.

So next time you have a little problem like "can't find entity list, it's not like assault cube" don't come to GH begging for help, we're not your mommy. You need to use the scientific method to figure it out, form multiple hypotheses. If you can't solve the problem after a few days, then ask your question. If you can't form solid hypotheses to test, the reason is because you don't have enough knowledge and evidence to form the basis of your experiment, in which case you need to search the forum and watch some tutorials.

Constants

Now we need to talk about constant variables and unknown variables.

Our hypothesis tries to solve an equation. The equation in it's simplest form is something like this:

A * B = C

- A and B represent variables which the experiment seeks to define
- C represents the observed behavior.
- The goal of our hypothesis is to define A and B.

Well A * B is the most simplest form. This is not reality. In reality there are hundreds of variables. You need to define as many as possible and every time you change a variable, you must redo the test.

This is the most important part right here, this is a HUGE problem for people inexperienced with this thought process. When trying to solve a problem they have too many variables and not enough constants. You cannot solve problems like this.

The goal is to define as many variables as constants as possible, this way your experiment only has to test a hypothesis as it related to one unknown variable or a small set of variables. If you only have to solve A and B that's not so bad, maybe you can define both in one experiment. But if you had 20 variables like when you try to find the TraceLine() function you are going to reverse many things before you have enough constants to have a successful experiment.

For example, if you have some weird issue with a lib on one game on one computer, but it works fine on another computer in another game your variables might look like:

- Game
- Operating System
- Architecture
- Game Logic

You need to isolate Game, Operating System & Architecture and define them as constants. This way you are only testing for the definition of "Game Logic". If you only define Game, Architecture as constants, when you perform your observation, you will not be able to draw a conclusion which defines the cause and effect of the other two variables in the experiment.

With too many non-constant variables you cannot attribute the cause and effect to the correct variable.

We see this all too often with questions "dll won't inject, why?"

No one can solve this problem because there are too many unknowns, we need constants so we can hypothesize the potential causes of the problem:

- What OS are you on?
- What Injector are you using?
- What game are you injecting into?
- Does it have anticheat?
- Is the game x86 or x64?
- Is the injector x86 or x64?
- Is the DLL x86 or x64?
- What's the source code for the dll?
- Are you compiling in debug mode or release mode?
- etc...

If you use the scientific method, you can solve almost any problem yourself with time and patience.

How to Be Successful & Achieve Goals

No one becomes a 1337 game hacker by just playing around in Cheat Engine for a couple minutes every month or by watching a couple videos.

To achieve success in anything in life you need to be able to set and achieve goals and give yourself realistic expectations.

Most importantly you need to be serious, if you're not serious and you're just doing this on a whim you will not be successful. Henry Ford and Elon Musk weren't laying in bed going "yeah I guess I'll open a car company maybe... or smth", be realistic! They devoted their entire lives to these projects.

Before you can be successful at anything you must:

1. Define your goal
2. Research what is required to achieve your goal
3. Come up with an action plan & timeline
4. Assess your ability to achieve the goal
5. Commit to achieving the goal
6. Perform the Action Plan
7. Achieve the goal and declare victory

Define your goal
"I want to learn how to hack games."

Research
DuckDuckGo "Learn Game Hacking"
Go to GuidedHacking.com
Read the Game Hacking Bible

Action Plan

- Do not skip any tutorials until I understand it 100%
- Spend 2-8 hours a day, 5 days a week learning game hacking
- For 2 months: Complete the Start Here Guide
- For 2 months: practice what I learned, writing C++ apps and making boring trainers
- For 2 months: Do the Reverse Engineering Guide
- For 2 months: practice reversing various apps and making kewl trainers
- For 2 months: Do the Intermediate Game Hackers Guide

Wow in 1 year I will be able to hack any game I want without anticheat, this is very realistic and I'm not retarded at all. Thanks Rake.

Assess
Can I really do this? Do I really want to do this? Is it this important to me? Am I smart enough and mature enough to do this? This is the biggest thing I've ever tried to do, am I up to the challenge? Do I really want to waste my time doing this? Does this have any long term benefits for me? Are they worth it?

My mom stills dresses me and I can't count to 10, maybe this will take me 2 years instead of 1 year...

Assess and revise your action plan!

Guide Stop Pasting - Focus on the Basics

Pasting is the subtle art of making yourself feel stupid 6 months later. It's the shortsighted and laziness epitomized by the game hacking community. If you're reading this, there is a 99% chance that YOU are a paster.

What Is Pasting?

Pasting is when you use other people's code with slight modifications and then pretend that you made it. Everyone thinks they can hack games by pasting other people's code. Sure you can paste code and you might get a working hack. But you're not a hacker or a coder, you're a paster which is the equivalent of a internet crack head. At some point you will realize you're an idiot for not heeding all the warnings here on GH.

What is a Paster?

A paster is someone who refuses to actually learn because it's "too hard". People can paste for years and never gain a single required skill. They'll post screenshots of "their" hack menu and thanks to the abysmal depths of mental retardation known as the game hacking community, these people actually think they are real hackers.

But what happens when they want to do something that they can't paste?

That's when they realize how stupid they are. They then understand how little they actually know. Something as simple as making an x86 hook from scratch is impossible for these people because they never learned any real skills.

Spending 2 years pasting only to realize you are an idiot? Why suffer this stupidity?

You could have spent 6 months actually learning the skills using our amazing resources.

Pasting is such a ridiculous waste of time, you will spend months failing to fix the most basic C++ errors instead of learning the entire process from the ground up. You will constantly be begging others for help with the dumbest, most pathetic errors and bugs.

Stop pasting IMMEDIATELY. You are making a mockery of god's gift of human intellect.

Do not be naïve and think you can hack games in a few hours or download a source code and be hacking a game a few hours later. Hacking is not easy. Don't waste your time if you're not committed to learning, you will fail.

Stop Pasting & Focus on the Basics!

You will learn nothing by pasting. Learn to hack correctly using our tutorials!

Taking my advice will be the single best thing you do in your game hacking journey.

Everyone that has listened to my advice, is a successful game hacker. Everyone who does not listen to me, still can't hack. They are still begging for help with the stupidest shit. Some of these people have been here for 5 years and still don't even know what a multi level pointer is. It's pathetic.

Have you been to other game hacking forums? Have you been in game hacking discord? Do you want to be one of these pathetic people?

I have devoted 9 years of my life trying to make the learning of game hacking as easy as possible. I have 9 years of experience answering thousands of questions and helping millions of people, I have identified all the problems, formed all the solutions & provided all the resources which you will need in the first 2 years of game hacking.

If there is anyone whose advice you should take, it is mine. I have done all this for YOU because before GH, learning game hacking was a lesson built on suffering. I want you to be successful and have sacrificed my entire life to provide you this learning resource.

Unfortunately, you can lead a horse to water but you can't make it drink.

The Paster's Dilemna

The paster believes a few hours or maybe days of pasting some code and begging for help will result in a working hack.

Many of you are young and have not learned how to learn or how to achieve goals yet. The answer is hard work, practice & patience.

Hard work, practice & patience is the antithesis to the paster's state of mind.

That is why you must discard this state of mind and adopt mine:

- Focus on the basics
- Follow our tutorials
- Practice each step
- Do not skip any steps

If you continue down the road of pasting, your entire experience will be failure and begging for help. You will never learn how to learn, how to solve problems or basic reverse engineering skills. When you need to do something yourself, you will have no idea what you're doing and you will feel like an idiot. You will always rely on other people to solve all your problems. Eventually people will stop helping you because they know it's a waste of time and you don't deserve it.

The best thing you can do is to do the Game Hacking Bible as it was was intended.

I don't write this stuff for fun, I do it because it is important, I am trying to send a very important message because year after year, no one listens.

Game hacking is not for lazy, stupid or immature people.

If you are not interesting in spending years coding, reverse engineering and debugging then game hacking is not for you.

The biggest problem we see is people focusing on the result and not the journey.

Real hackers do it for fun & for the challenge. Most of the pros here just make the hacks and don't even use them.

If you're pasting to avoid the hacking part, you're not even a hacker and you don't belong in the community.

Game Hacking Requirements:

- Intelligence
- Hard Work
- Determination
- Learning by yourself
- Years of coding
- Years of debugging
- Years of reverse engineering
- Years of reading

If you cannot fulfill these requirements, stop here, you're not going to be successful.

Every year hundreds of thousands of people come here trying to paste and fail miserably. They never become real hackers. Do you want to be a failure?

Out of 400,000 members, only 2,000 of them are successful game hackers. You have a 0.005% chance of being successful.

Does that put it in perspective?

Coding is the most important skill

The better coder you are, the better you will be at reversing and the better you mind will be trained for everything regarding game hacking. You should be coding every spare minute you have. 99% of questions are asked because the user simply doesn't know how to code. You can't learn hacking without learning coding. Stop avoiding the hard work. It is 100% necessary, avoiding it is ridiculous and just shows how stupid these people are.

Reverse Engineering skill is required

If you are pasting, you aren't reversing anything. You're not a hacker, you're a poser. I can give you a list of 100 people who have been here for years, who never learned how to hack, only how to paste. You can view their threads and easily identify their problem: they never learned how to reverse engineer because all they did was paste. You can't solve problems if you can't reverse engineer.

Why do pasters fail?

They fail because they're trying to do something beyond the scope of their abilities. You should focus on learning the basics, create a foundation of knowledge and practical experience. Use this foundation to elevate yourself to a position where you can more easily approach your goals which are above your skill level.

If you aren't a real hacker, you can't hack complicated new games. You can't rely on other people to solve all your problems. If you want something, you have to get it yourself. This is reality.

The best way to solve any problem is to discover what skills you are required to learn to solve it 100% yourself. You have to start with learning each and every part until you have the foundation of knowledge & experience required to solve your problem.

Pasters can't do this because they are the problem! The Visual Studio error is just a symptom of their own stupidity.

The Patron Saint of Pasters died in 2020. It's officially time to stop this pathetic behavior.

So what now?

You're just starting game hacking and you have a choice to make:

Do I want to be a paster or a hacker?

A paster depends on others and achieves nothing without begging.

A hacker solves their own problems.

Choose wisely.

p.s. if you still don't get it (please god no), start here:

The Game Hacking Bible

Squally CS420 Game Hacking Course

CS420 is a Game Hacking Course, a series of lectures brought to you by Zac from the Squally team. We have been working with Zac to help promote Squally for the past year, we think it's an awesome tool that has a lot of potential. Zac is now putting a game hacking course together with GH called CS420 which is a growing series of 8 lectures that makes up the first stage of our Game Hacking Bible. These videos will cover many computer science concepts that will form the basis of your game hacking career, and then move forward into practical application of this knowledge by making simple modifications to games. Once you've completed the CS420 tutorials you will be ready to move on to the rest of the GHB where will we jump directly into hacking games.

How to Follow the CS420 as part of the Game Hacking Bible

Watching these videos if a fun an informative way to give you all the background information required to progress to the rest of the GHB. You don't have to practice or follow along with the videos.

CS420 Chapters

- Game Hacking Course Introduction
- Memory Editing 1
- Base Systems - Hex, Decimal & Binary
- How to Hex Edit Games
- Memory Editing 2 & Data Types
- Virtual Memory
- Virtual Memory 2 & Multilevel Pointers Tutorial
- How to Edit Assembly Tutorial

What You Will Learn:

- Basic Computer Science & Game Hacking Concepts
- Simple Data Manipulation
- How To Work With Game Hacking Tools
- How To Read/Write Using Assembly
- Fundamental Reverse Engineering Concepts
- Basics of using Cheat Engine

CS420 Game Hacking Course Playlist

https://youtu.be/hj4rhfnikVs?list=PLt9cUwGw6CYG1b4L76vZ49tvI2mfmRSCl

Additional Information

This series will introduce you to the absolute beginnings of your game hacking journey. Learning about game hacking should be a fun and practical experience. The journey of learning how to hack games is NOT easy. However CS420 aims to create a hacking tutorial series where you can learn the fundamentals of reverse engineering and assembly in the best way possible.

On completion of this game hacking tutorial you will have a fundamental and basic understanding of how game hackers carry out their day to day operations. This Squalr tutorial is the first step in the right direction of becoming an experience hacker. Learning game hacking takes a lot of dedication but if you stick with this course, you will be in a good starting position. We hope you enjoy this tutorial series and with this new found knowledge you continue to grow as a game hacker.

Squally

Squally is a video game designed to teach you hacking, not with boring lessons & lectures but with a variety of minigames. If you want to support Zac directly, please buy our game and give it a nice review after you've tried it. We have stopped development for the time being, but it still provides several hours of fun and engaging gameplay.

Squally Website

Squally Steam Page

Squalr - Memory Editor & Cheat Engine Alternative

You may also know Zac from his previous project Squalr, which is an alternative to Cheat Engine. Squalr is an open source memory editing and game hacking tool made in C#. It has a particularly fast pointer scanner, which you may prefer to CE.

In this second CS420 video, you will learn the details and practical skills of game hacking. Note that practice is crucial in your learning process, thus please get your hands dirty and try to repeat the tutorial on your own machine and games. Only in this way can you have a hands-on experience of game hacking.

This video, CS420 - 2 Memory Editing 1, will teach you the basics of memory editing. Memory edition is the most fundamental skill a game hacker should possess and is the prerequisite of any high-end skills and shenanigans you could think of. Therefore, make sure you understand EVERYTHING in this video and have enough practice; preferably also try this technique on other single-player games you are playing.

What You Will Learn:

Operating System Basic Operating Logic

Introduction to Windows API(Application Programming Interface)

Routine of Memory Editing

Data Storage and Representation Systems

Examples of Memory Editing using CE and Squalr

CS420 - 2 Memory Editing 1

After watching the video, you can keep on reading the rest of the article, which may help solve some problems you encounter in the video. If you feel confident about the content learned, try to finish the homework part trailing this article.

https://youtu.be/xOBE_vWDX_I

The first half of the video is theoretical, and you may not find it helpful immediately. Therefore, here we provide more information to help you memorize and hopefully give you some inspiration on their significance.

When you edit a game with CE, you are editing its copy in the memory instead of editing the program stored on your hard disk. This means most of your editing will disappear if you restart the game. The good news is that you will not possibly crash the executable. However, you can still corrupt your save file, so make sure to create a backup before you make a permanent change to your save data(for example, edit your money/inventory).

Windows API seems far away if you only use CE or Squalr. However, when you become more proficient in game hacking and want to write some independent executables/libraries, you need to handle the work done by CE yourself, such as memory scanning and editing. Then, you need to have knowledge of Windows API. You will learn more about this in the following articles in GHB.

The routine of memory editing, namely, Locate-Edit-Verify, is the core loop of any hacking actions. Keep this routine in your mind all the time, and it will help you keep track of your work.

It is important to know that one value may have various ways to be stored in the memory. When you search for something and cannot find it, change the value type and try again. Popular ways for a game to store a number are to use DWORD(4 Bytes) or float, but sometimes they can be QWORD(8 Bytes), double, or WORD(2 Bytes). Also, there are some specific tutorials if you want to find position coordinates or view angles. If you want to search for strings, it would be more difficult with the involvement of encoding. Here is an article dedicated to this issue. Do not be afraid if you cannot fully understand the content; just come back and read it again when you get confused.

In this CS420 video, you will learn about base systems. Though we started using CE at the end of the last video, now we will go back to the theory part.

Why do I need to know about base systems?

This is a common question regarding theoretical knowledge. That thing seems far away from practice, but they are used almost everywhere in game hacking. You can ignore the theory and only focus on practice, but eventually, you will learn this after suffering and many detours. Therefore, to prevent the waste of time and energy, you should know about base systems before you need to use them.

How do base systems help with game hacking?

By learning about base systems, you will have the idea that there are many ways to represent the same number, and a numerical representation may have different meanings in different contexts. Specifically, you should get used to hexadecimal numbers and use them as fluently as decimal numbers. You will frequently deal with hex numbers in the future, so ensure you can eliminate the confusion with this tutorial.

What You Will Learn:

- Concept of base systems
- New symbols in other base systems
- How to count in different base systems
- How to use a calculator to convert numbers between base systems
- Supplementary resources to help understand base systems

CS420 - 3 Base Systems - Hex, Decimal & Binary

The concept of base systems is not difficult, but it needs some time to digest. After finishing the video, you can check the rest of the article to enhance your understanding of base systems.

https://youtu.be/nA7o5kmH6wg

Mathematical Expressions for an arbitrary base system

Consider an n-based system, and q,w,e,r,t,y are symbols in this system. Then the number qwerty means:

If you change every element of the above expression into its corresponding decimal number, then you can calculate it and get its representation in decimal form.

How to convert an n-based number into a decimal?

You can use the above method, namely, convert every digit into decimal and sum them up.

How to convert a decimal number into n-based? (Optional)

This is more difficult since we are using division and modulus. Suppose we are converting a number x into n-based. For the first(rightmost) digit, we can get it by x mod n, which is the remainder of x/n. Then for the second digit, we can solve it by (x/n) mod n. Note that we are using the quotient of x/n and will discard the remainder instead of resulting in a float number.

Formally, given a decimal number x and a base n, we can get the base-n representation of x by

Where the comma separates two digits.

If you are confused about why this will work, just try to convert it back to decimal:

Where m is a number such that n^m>x. Since we only take the quotient, x/(n^m) = 0. Therefore, the above expression equals x.

Indeed, the two converting methods mentioned above are functionally the same. The difference is only on which number system you want to do the calculation. And we use both of them because we always want to do the calculation in the decimal system, which we are familiar with.

How to convert a number between two arbitrary systems?

Easy, you first convert x-based into decimal, then convert the decimal into y-based. For computers, it uses hexadecimal(binary) as the medium.

Why is it easy to convert hex numbers into binary?

Because 2^4=16, which means that each hexadecimal digit can be exactly described by four binary digits. Therefore, the conversion is just a series of mapping without any other calculations. Similarly, if there exists a 100-based number system, it can be converted to decimal as easily.

Do I need to calculate the conversions by hand?

No, computers can help you with that. The point is to understand what you are dealing with, and you just need to know about concepts and let your computer do the tedious calculation.

Rake is the devoted owner of GuidedHacking.com, an educational platform dedicated to reverse engineering, game hacking & information security. With 10 years of experience building this resource, Rake has grown not only as an expert in the field, but as a trusted authority on the subject. A passionate educator, Rake has meticulously curated a wealth of tutorials, courses & other resources unparalleled in the game hacking community. Learn more: About Rake

Beginning from this CS420 video, we will step into the practice part of game hacking. In this particular video, we will learn to use hex editor to modify game save files.

Why do I need to learn about hex editing?

Because the process of hex editing is similar to memory editing. By learning more about text editor and the editing process, you will be more familiar with Cheat Engine UI and game hacking process.

What You Will Learn:

- Hex editing process
- How to find your save file
- Principals of Process Monitor
- Usage of Process Monitor
- Strings and Character Encoding
- Hex Editor Interface
- Big Endian and Little Endian
- Use HxD to edit save files
- Limitations of Hex Editing

CS420 - 4 - How to Hex Edit Games

As usual, there will be a short Q&A session after the video in case you have any questions.

https://youtu.be/EpcK8uk7lcY

Why do we need hex editing?

If you can directly edit the game memory, then basically you can achieve the goal without hex editing. However, if you want to hack some console games, it is much harder to access the memory and runtime process. Then hex editing will come in handy since you can copy the save file to a computer and edit it here.

How can Process Monitor know the location of save file?

We've introduced the operating system before and mentioned that it provides a series of APIs for user processes. Indeed, if one process wants to make some changes on the disk, its only choice is to use Windows API. Therefore, we can inspect their calls to those APIs and find what file the process tries to access through the parameter. Intercepting windows API is very useful in game hacking, and you can learn more about Windows API here.

Why are there different Character Encodings?

In the beginning, the computer only supports English and some simple symbols; one byte(256 numbers) is enough to map them into different indices. That's the origin of the ASCII code. However, when computers prevailed worldwide, we needed to support more languages and symbols, and UTF was invented. Nowadays, most texts are encoded by UTF-8

since it covers most of the commonly used symbols and is compatible with ASCII (every ASCII symbol has the same encoding in UTF-8).

Is my computer Little or Big Endian?

Generally, Little Endian. All x86 processors use Little Endian, and x86 is the most popular process type nowadays. For more information about x86, you can refer to this post.

Can I try to edit the save file of the game I'm playing?

Honestly, it might be difficult. As is introduced in the last section of the game, most recent games have encrypted their save file, and it is much more difficult to edit them. If you want to make some permanent changes to your save file, a more feasible way is to edit the game memory and let the game save it.

In this CS420 video, we will review the knowledge of data types and talk more about memory editing. The content of this video is highly related to real hacking, so ensure you understand the concepts and have enough practice.

What You Will Learn:

- Review of Integer Data Types and Strings
- Boolean
- Signed Integers
- Floating-Point Numbers
- Unknown Initial Value Scan Techniques

CS420 - 5 - Memory Editing & Data Types

This video contains both theoretical and practical content. Also, this content has some overlapping with the second video. Therefore, you can refer to the Q&A session of that video if you encounter any questions not answered below.

https://youtu.be/6KNNRqjpgGE

Why do integer types not have a variable length like a string?

If an integer may have a variable length, there must be another symbol to work as a stop, like the \x00 in the string. The selection of this symbol is already a problem, not to mention that it will take up a lot of memory space due to the large population of integers. Also, the variable length will strongly affect the efficiency of the CPU since it now needs to handle arbitrary lengths of integers. And the alignment, which can boost the calculation, is also impossible. Moreover, representing a negative number will also become difficult since you need to find a symbol to represent the negative sign.

Overall, variable length integers seem straightforward, but it is indeed very complex and slow.

Why do we use one byte to store a Boolean?

Because byte is the minimal addressing unit in memory. You cannot directly access a bit, but you need to instead first locate the byte containing it and use mathematic expressions to get the content of a particular bit. Therefore, for simplicity, a Boolean is stored as a byte. Also, in programming, 0 means false, and anything else is true.

Why do we use two's complement to represent signed numbers?

This is a common question since two's complement is not the most straightforward way to represent a signed number. Intuitively, we can just use the leftmost bit as a sign: if it is 0, then the number is positive, else it is negative. This representation has a problem: 0 can be expressed in two different ways since +0=-0. Another problem is that the CPU must handle subtraction and addition separately in this representation. However, in two's complement, every number will have a single representation, and the CPU can handle addition and subtraction in a uniform way. That's the beauty of two's complement and why modern computers use it. If you want to learn more about it, you can check Wikipedia. (This topic is more related to computer science and does not help much with game hacking).

Why can floating-point represent very large numbers?

You may have noticed that the max value of a float is much larger than the max value an int can represent. So why do we even use int types? Although a float can represent a very big number, it cannot represent every number from 0 to it. This is natural: float and int have the same size, so they can only store the same number of different values, not to mention that float has to represent fractional values. Indeed, the representation of a float number can be viewed as N*M, where N is an exponent of 2 and M is a fractional number. Therefore, if N is very large, the result can be fairly big. But the values will become sparse as N goes big since M has a limited size. If you want to learn more, you can refer to this page. Again, the details of floating-point numbers are somewhat off-topic, and you can ignore them if you are not interested.

Why can we find the y-position after we know about the x-position of an object?

In most games, the coordinates of an object are represented by a 3-Vector, namely, a collection of three floating-number points. Therefore, those numbers are usually put together, and you can first locate one of them by value searching and find the rest by looking into its neighborhoods. This is indeed a common technique in game hacking since the game will store related information in the same place (usually members of the same class). And by looking into the near memory region of one value you may possibly find something else useful.

In this part of our CS420 game hacking course, we will dive deeper into the operating system and learn more about virtual memory. This video will help you understand more about the underlying mechanism of a running process. This concept is essential if you want to do some static analysis of the game or just write a cheat table that survives game restarts.

What You Will Learn:

Overview of Virtual Memory

Memory Size for a process

EXE File Structure

Virtual Memory Structure

Memory Allocation and Deallocation

Static Address

CS420 6 - What is Virtual Memory?

This video is more related to theory. Make sure to check the Q&A session below after watching the video.

https://youtu.be/aPNcEckD1Qk

What's the virtual memory size of a 64-bit process?

As the video introduces, a 32-bit process will have a virtual memory size of 4GB. This is because the max memory size for a 32-bit system is 4GB. Nowadays, most gaming computers are 64-bit, and the memory size limit is vast. Yet we do not need that much virtual memory, and it is a waste to use a 64-bit address. Commonly, a 64-bit process uses a 36-bit long address, meaning that it supports 16*4=64 GB of virtual memory. Half of the virtual memory is reserved by the system; thus, the process can only use 32GB.

Why can the exe file be loaded into the same place?

Generally, because it is the first object loaded into the memory. There are three classes of items loaded into memory: the exe file, libraries, and generated objects. The last class is generated by codes in either libraries or the exe, while the other two are original files on the disk. At the beginning of those files, there is information about their preferred location to be loaded. The word "preferred" means that this cannot always be realized since other things may have already taken the space. However, when it comes to the exe file, this is not a problem since it is the first thing to be loaded, and it can always hit its preferred location unless the OS intentionally puts it somewhere else by some system options.

Is the address still static if the exe is not loaded into a fixed position?

Fortunately, yes. Though the exe may be loaded into a different location, we can easily find its address by calling some Windows APIs. After that, we can use the relative address to access the content in the exe file (aka the main module), which will never change by game restarts.

Why is there readable content at the beginning of an exe file?

The system needs more information to operate on a file besides its postfix. For an executable, there is a PE header at the beginning to help the system handle itself correctly. There will be some plain text in this header, and the abovementioned preferred location to be loaded is also there. To know more about PE headers, you can check this post. Note that the PE header is complicated and difficult to understand for a beginner, and you are not required to digest it now. This link is just for reference, and you can return to it later when you become more knowledgeable.

In this video, we will explore one of the most crucial concepts in game hacking, reverse engineering, and even programming – Pointers. Pointers are the foundation of dynamic addresses, and mastering them is the foundation of being a game hacker. You are encouraged to comprehensively understand the content of this tutorial and perform substantial practice to enhance your memory.

What You Will Learn:

- Introduction to Pointers
- Memory Leak
- Multilevel Pointers
- Pointer Scan

CS420 7 - Virtual Memory & Multilevel Pointers

Again, the content of this video is essential. Make sure you understand it and finish the homework.

https://youtu.be/_W0xdVO8-j4

What's inside a pointer?

To discuss this, first, we need to have a structural understanding of the memory. Memory is just a data block where you can locate an object with its address, and anything can be stored in this object. For example, address 00000000 may store an integer 1, and 14000100 may store a float 0.5. Under these settings, a pointer is a memory object whose content is the address of another memory object. Following the example above, if the content of address 00000000 is now 14000100, we would say that this address points to 14000100. We often use [] to refer to the content of an address, for example, [14000100] is 0.5, and [[00000000]] = [14000100] = 0.5.

In summary, a pointer is a memory record that saves a memory address.

Pointers are unidirectional

A critical property of a pointer is that it is one-sided. But it is not challenging to backtrace it: A pointer is a memory object that stores another object's address, and you can scan through the memory to locate objects with a specific value. However, there might be more than one result, and you need to distinguish which pointer you need. Another approach is to follow the assembly code to backtrace the pointer, which will be illustrated in the next video.

Pointer Scan

The video has shown the usage of a pointer scan. This potent tool generates a possible path from a static address to the place you want to locate. Yet since there might be many results, you may need to filter them, and sometimes you need to redo the pointer scan. In this case, storing the scan result would be helpful and efficient. You can refer to this tutorial for a better way to use the pointer scan function.

Do we always have to find the static pointer path?

Not really. Sometimes the path is too long and difficult to find, and sometimes you cannot find a static path. There is another way to achieve the goal – say, change a specific value – without the path: assembly editing. Details about this technique will be introduced in the next and last video.

In the last video of this series, we will discover the heart and soul of reverse engineering – assembly language. Unlike pointers used everywhere in programming, assembly is basically known to reverse engineers and hardware architects. It is the language of machines, and mastering it means you can change the game's logic. Understanding assembly is the core of reverse engineering – knowing what the code is doing, then changing it as you wish. Upon acquiring the knowledge of assembly, you can call yourself a beginner reverse engineer and game hacker. Now let's dive into the series' last and arguably most critical chapter.

What You Will Learn:

- Assembly Editing Process
- CPU Architecture: x86, ARM and PowerPC
- Assembly Language and Machine Code
- C++ and Compilers
- JIT(Just-In-Time) Compiling and Script Languages
- Determine the Language of a Game
- How to Locate Assembly
- Introduction to Assembly Language
- Registers, RAM, and Hard drive
- X86 and X64 Instruction Sets
- Overview of Registers
- Common Instructions
- How to Write Assembly Scripts

CS420 8 - How to Edit Assembly

This video is longer and requires more time to digest. After finishing this video and the rest of the article, you should be able to make your own hacks and proceed further in the GHB.

https://youtu.be/_Sm84vARhbw

Why ARM Architecture saves the battery?

Because of the design of the instruction set. In x86, you can find that instructions may have different lengths. Thus, the CPU must recognize the instruction and determine how many bytes it needs to read, which takes extra calculations.

However, in the ARM structure, all instructions have the same length, meaning that the CPU no longer needs to care about the variable instruction size. This brings about a more straightforward design and lower power. The downside of the uniform length is some reduction in the performance, so PCs and most laptops are using x86 since they do not have a big battery issue.

Why is Decompiling more difficult than Disassembling?

What we have in the EXE file is the machine code. Assembly and machine code have a one-to-one relationship, meaning that given one (complete) line of machine code, there is only one correct corresponding assembly instruction. Therefore, disassembling is pretty easy. However, compiling is much more complicated, and the result is not one-to-one. Firstly, all variable name is lost, so you cannot determine the purpose of a value by its name. Modern games also hide class information and dynamic types.

Consequently, you do not know the name of nearly anything in the machine code. Secondly, the compiler will try to optimize the user code for efficiency, and the code flow and logic may change. As a result, although decompilers can generate some code, it is not directly readable and digestible. Nevertheless, Decompiling is still essential in reverse engineering, and making the decompiled code readable is one of the critical skills of a game hacker. You will learn about reverse engineering more thoroughly later in GHB2.

Important Feature of Assembly Editing

When you edit the game code, you can only make in-place changes, meaning that you can only overwrite instead of inserting codes. There is one way to work around this limitation: Hook. You can find some empty space in the memory, then edit the game code to jump to this area, execute your assembly script and the game code overwritten by the jump, and then jump back to the original place. This is a common and popular technique used in game hacking.

Homework

Now that you're finished with the CS420 game hacking course, you can continue to work on GHB1 where we will teach you how to use this knowledge in real world scenarios, creating external and internal trainers.

Guide Game Hacking FAQ - Frequently Asked Questions

Game Hacking FAQ

This is a FAQ regarding game hacking only!
Read about our website itself here: GH Forum FAQ

100% of all your questions are answered by doing the Game Hacking Bible
If you're not doing the GHB, then you're an idiot. We made this course specifically so we never had to answer questions ever again.

GHB Considerations
Why are we having you read this now before you even start doing the tutorials?
Because these are the most commonly asked questions in the first 3 months of learning. Rather than waste your time googling for answers, we'll provide them all here for you before you even have the questions.

The Basics
How to Bypass anticheat?
If you're not an advanced coder/hacker, don't waste your time. Focus on learning to hack games without anticheat. Do not post "how to bypass" shit threads please.
How to Bypass Anticheat Guide for Noobs

Should I learn C++ or C#? What language should I learn?
Learn C# if you want to make mainly external trainers. Learn C++ for every other reason. Once you know 1, you can learn the others easily. Starting learning coding immediately, do not waste any time trying to decide. If you need help deciding, just pick C++. The faster you start coding, the faster you become dank, just go go go! Read more here

Game crashes when loading Cheat Engine?
Change debugger in CE settings to VEH Debugger, if that doesn't work, read the first couple paragraphs of Guide - How to Get Started with AntiCheat Bypass

How to use MultiLevel pointers in C++?
Use the good ol' FindDmaAddy function from Fleep that de-references and adds the offsets for you, we fixed it up a bit to be even nicer

How to use client.dll+0xDEADBEEF in C++?
Use the GetModuleBase function which uses the ToolHelp32Snapshot function to walk through the loaded modules and grab the base address on your modules.

How to find multilevel pointers in cheat engine?
Part of 2 of this Cheat Engine Tutorial video will show you an excellent method of finding pointers manually.
PointerScanner Video Tutorial : Learn how to use PointerScanner and pointermaps for finding multilevel pointers

How to find offsets?
Use Cheat Engine "Find what Accesses/Writes", in the assembly the game will access variables using the proper offsets, these are the offsets you will use. This tutorial is very good for beginners.

Why isn't my hack working?
Because you suck. Learn to Debug Your Hack with the Visual Studio Debugger

How to inject a DLL?
Use the GH Injector!

How to update offsets/addresses for a new version of the game?
Anywhere in the project where addresses or offsets are defined you need to update them. Learn to find pointers/offsets here

What's the difference between internal and external?

What game should I learn game hacking on?
Assault Cube first, then either CSGO or COD4. Do not learn game hacking on new games, it's too complicated for beginners. These 3 games have tons of resources and everyone on the forum can help you hack them because we have experience with them. If you pick some random new game, no one can help you and it will be a waste of time and extremely frustrating.

Where do I download IDA Pro?
We do not condone piracy, you want to buy the latest version from the IDA Pro Website. Buy and download version 7.5 or whatever version is newest, but 7.5 will have all the features you need.

What is an Entity?
An entity is the programming term to describe an object which interacts with other objects. In Unreal Engine they call it an *Actor*. Entities or Actors require the most code because they must be able to interact with each other dynamically. Entities typically include the players, bots and things like grenades. Indeed in the game's source code player's are defined as playerents.

You will hear this term used when hacking all games, essentially an entity is a player object.

What is an Entity Object?
playerent is the class used for players. A class defines what a class of objects contains and what they can do. An object is an instance of a class.
So if we have a class named playerent, we create an object of this class named Rake. Now Rake is a dynamic entity which contains all the required variables and can execute all of it's class functions. You will learn more about this later in the Learn C++ part of the GHB.

Essentially, an Entity Object is a player's structure in memory.

You will hear these things being used interchangeably: player object, entity object, player, entity. They all refer to the same thing.

What is an Entity Object's Base Address?
This is simply the first address where the structure begins, with no offsets added. Sometimes we refer to it as offset 0x0.A

What is the Local Player?
This is the player object controlled by your mouse and keyboard. Other players and bots are typically referred to as players or entities in contrast.

What is a green address in Cheat Engine?

A static address.

What is a static address?

An address which is "always" the same, even when you restart the game. In reality it means an address which is always relative to the base address of a module. You will learn more about that later here: GetModuleBase Function. What matters is that you don't need to follow a pointer path to access a static address, meaning FindDMAAddy() is not required, making hacking these variables much easier. You will learn all about this in the next few videos.

What is a pointer?

A pointer is a variable, which contains an address. This seems simple but it will be the most confusing part of the next 3 months of your life. It's something that will "click" in your head one day and make perfect sense, do not kill yourself trying to understand it right now.

What's the difference between "what writes to this address" and "what accesses this address"?

What Writes only returns instructions which overwrite the value at that address.
What Accesses returns both writing and reading instructions, meaning any instruction which touches the address.
We typically just use "What Accesses" because doing it once is better than twice.

What is the structure dissector used for?

This tool lets you easily reverse engineer structures, with the ability to rename each variable. It also makes it easy to find variables which are also in the same structure. If you found the health address, then finding the player position coordinates will typically be as easy as looking inside the same class. While Cheat Engine's implementation is quite good, with it's auto-guessing feature, ReClass.NET is much better, you will be taught how to use it later.

What is a Logical Pointer?

In this video, we find a pointer that is not logically defined but it is still a valid pointer in memory. Within a large game process, there can be hundreds of thousands of pointers which point to the correct address. You cannot depend on most of them because they just temporarily hold the correct address. A logical pointer will always point to the correct address, and they're easy to find when you follow the game logic using find what accesses. You want to use pointers which are used by game logic you have identified, these are most likely to work forever because they utilize a logical pointer chain. You can discover the logical pointer path by finding out what writes and what accesses.

Common Visual Studio Problems

Linker Errors:

C++:

```
Error LNK2019    unresolved external symbol
```

This error means the compiler cannot find the "external symbol". The "external symbol" is usually a function. You're calling a function but it wasn't included properly, like you probably forgot to include the header file or link the lib/dll. The compiler will even tell you what line of code the error is on. You need to correctly link the header file and any LIB/DLL that's needed.

For instance this error:

C++:

```
Erro LNK2019    unresolved external symbol glBegin referenced in function "int __stdcall
hwglSwapBuffers()
```

Was fixed by adding:

C++:

```
#pragma comment(lib, "lib\\OpenGL32.Lib")
#include <gl\GL.h>
```

Unicode/MultiByte Character Set
Seeing error "cannot convert char* to LPWSTR?" or something similar?
This happens because projects can be be set to use a certain type of string literal, Unicode or "regular" Multibyte Character Set (MBCS)
Fleep made his tutorials using Multibyte Character Set but the industry standard is Unicode, Visual Studio 2017 has the default set to Unicode.
If you want char* string literals to be single byte chars, set the project to Multibyte Character Set, especially if you're doing a Fleep tutorial.
Project Properties -> General -> Character Set

error C2664 cannot convert argument 1 from 'const char [11]' to 'LPSTR'

error C2664 cannot convert argument 1 from 'const char [11]' to 'LPSTR' - Guided Hacking

How to install Windows Forms for Fleep Tutorials on Visual Studio 2017?
Visual Studio 2017 - Windows Forms

Why don't we teach GetWindowThreadProcessId to get the process id?
Because this function has notoriously been the cause of thousands of noob problems.
This function will cause you grief when multiple windows have the same title.
If you use our ToolHelp32 method, you will never have any of these problems.

Precompiled Header: stdafx.h vs pch.h
Visual Studio keeps changing the default name of the precompiled headers. It doesn't matter which you use as long as you're consistent in your project. Use whatever Visual Studio generates. You can always change it in the project settings if you want.

Why use Precompiled Headers?
It makes compilation of large projects much faster, which is very helpful when you're debugging and solving errors in your code. Sometimes you will be hitting the compile button 5 times per minute and saving 5 seconds per compile is huge.

Confused by the windows API?
Everyone is during this tutorial. Just do the GHB and you will learn 1 piece of it at a time. Make sure you read What is the Windows API?

What's up with _wcsicmp?
```
if (!_wcsicmp(procEntry.szExeFile, procName))
```

_wcsicmp = wide character string insensitive compare

This means it compares two strings where each character is a wide char (2 byte char instead of a regular 1 byte char), wide characters strings can accommodate UNICODE which requires 2 bytes for char. We use Unicode in almost all our tutorials because it's the industry standard now.

Insensitive means it's case insensitive, it ignores the capitalization, which is very helpful, if you forget to capitalize something, it still works, so you don't have to hunt down rogue bugs because of a simple typo in the process name.

_wcsicmp is one of many string comparison functions, they're basically all the same with slight variations. The variations are in the type of strings they accept as arguments, you just use the one that works on the string types you already are using. This might be super confusing now but it's really simple once you have more experience.

_wcsicmp return value checking with the not operator (!) ??

```
if (!_wcsicmp(procEntry.szExeFile, procName))
```

using the not operator (!) before a function call, is essentially checking if the return value of the function is zero

When the return value of this function is zero, it means the strings are identical and we found the correct process:

What is a trainer?

The word trainer typically means it's a simple external cheat.

WTF does this mean: "OG for a fee, stay sippin' fam"

This is just a stupid line of text I came up with to test console output. Fleep (the **O**riginal **G**angster of GH) used to drink alot of tea and juice and chocolate milk in his videos and you could always hear him sipping. He even ate ice cream during tutorials, so we're just making fun of that.

Video Tutorial How To Hack Any Game - Cheat Engine

This Cheat Engine tutorial and the next video, will bridge the gap between our previous teachings on Cheat Engine and *real game hacking*. It will cover the basics of finding addresses, pointers & offsets, and introduce you to key terms such as base address, entity, local player, entity object, static pointer etc...

What will you learn?

- How to find the local entity object
- How to find pointers
- How to find offsets
- How to use Structure Dissector

This is a remake of an old video made by Solaire which should be much more clear and is a good example of the quality of videos you will find later in the course. This is the first video introducing you to Assault Cube, which is an open source, free, native, x86 game coded in C++ that does not use a game engine. This makes it ideal for game hacking tutorials and we will use it for the majority of the GHB.

The next video will teach you how to write a simple C++ trainer using the pointers we obtain in this video.

Considerations

Even if you did CS420 or Game Hacking Shenanigans first, do NOT skip this video, this is the foundation of all future tutorials.
Do not try this on other games yet, they will be much more difficult and only confuse you.
You must be in a real bot match for all tutorials, the demo map will not work.
You must use the correct version of Assault Cube: 1.2.0.2

Requirements

- Cheat Engine
- Assault Cube 1.2.0.2

How To Find Offsets, Entity Addresses & Pointers

When you're done watching the video, continue reading down below where we will clarify some things from the video that may be confusing. Then practice this on Assault Cube, keep repeating the steps from the video until you can do it without referencing the video.

https://youtu.be/YaFlh2pIKAg

What is an Entity?

An entity is the programming term to describe an object which interacts with other objects. In Unreal Engine they call it an *Actor*. Entities or Actors require the most code because they must be able to interact with each other dynamically. Entities typically include the players, bots and things like grenades. Indeed in the game's source code player's are defined as playerents.

You will hear this term used when hacking all games, essentially an entity is a player object.

What is an Entity Object?

playerent is the class used for players. A class defines what a class of objects contains and what they can do. An object is an instance of a class.
So if we have a class named playerent, we create an object of this class named Rake. Now Rake is a dynamic entity which contains all the required variables and can execute all of it's class functions. You will learn more about this later in the Learn C++ part of the GHB.

Essentially, an Entity Object is a player's structure in memory.

You will hear these things being used interchangeably: player object, entity object, player, entity. They all refer to the same thing.

What is an Entity Object's Base Address?

This is simply the first address where the structure begins, with no offsets added. Sometimes we refer to it as offset 0x0.

What is the Local Player?

This is the player object controlled by your mouse and keyboard. Other players and bots are typically referred to as players or entities in contrast.

What is a green address in Cheat Engine?

A static address.

What is a static address?

An address which is "always" the same, even when you restart the game. In reality it means an address which is always relative to the base address of a module. You will learn more about that later here: GetModuleBase Function. What matters is that you don't need to follow a pointer path to access a static address, meaning FindDMAAddy() is not required, making hacking these variables much easier. You will learn all about this in the next few videos.

What is a pointer?

A pointer is a variable, which contains an address. This seems simple but it will be the most confusing part of the next 3 months of your life. It's something that will "click" in your head one day and make perfect sense, do not kill yourself trying to understand it right now.

What's the difference between "what writes to this address" and "what accesses this address"?

What Writes only returns instructions which overwrite the value at that address.
What Accesses returns both writing and reading instructions, meaning any instruction which touches the address.
We typically just use "What Accesses" because doing it once is better than doing it twice.

What is the structure dissector used for?

This tool lets you easily reverse engineer structures, with the ability to rename and change the type of each variable. It also makes it easy to find variables which are also in the same structure. If you found the health address, then finding the player position coordinates will typically be as easy as looking inside the same class. While Cheat Engine's implementation is quite good, with it's auto-guessing feature, ReClass.NET is much better, you will be taught how to use it later.

What is a Logical Pointer?

In this video, we find a pointer that is not logically defined but it is still a valid pointer in memory. Within a large game process, there can be hundreds of thousands of pointers which point to the correct address. You cannot depend on most of them because they just temporarily hold the correct address. A logical pointer will always point to the correct address, and they're easy to find when you follow the game logic using `find what accesses`. You want to use

pointers which are used by game logic you have identified, these are most likely to work forever because they utilize a logical pointer chain. You can discover the logical pointer path by finding out what writes and what accesses.

Having trouble?

Having trouble understanding offsets, entity addresses, and pointers is normal when you first start. It will take months to have a solid understanding of it. Watch the video again, evaluate all the details slowly and practice each step. If you're still stuck, sometimes skipping it and going through the next few tutorials, will solve your frustration. But if you get 2-3 videos ahead and are still off-track, come back to this tutorial.

If your scans find more addresses than the video, just continue to filter them down by hurting yourself and doing additional scans for the new health value.

Homework

Repeat the steps from the video 2-3 times until you can complete them without referencing the video.

GHB Navigation

Prev: GHB105 - Beginner Cheat Engine Tutorial Video Guide
Curr: GHB106 - How To Hack Any Game with Cheat Engine
Next: GHB107 - How to Find Position Coordinates with Cheat Engine

Tutorial How to Find Position Coordinates with Cheat Engine

What are coordinates?

In 3D video games, position coordinates define the location of an object in 3D space. We define 3D space by using the Cartesian coordinate system.

From Wikipedia:

--

A **Cartesian coordinate system** is a coordinate system that specifies each point uniquely in a plane by a set of numerical **coordinates**, which are the signed distances to the point from two fixed perpendicular oriented lines, measured in the same unit of length. Each reference line is called a *coordinate axis* or just *axis* (plural *axes*) of the system, and the point where they meet is its *origin*, at ordered pair (0, 0). The coordinates can also be defined as the positions of the perpendicular projections of the point onto the two axes, expressed as signed distances from the origin.

One can use the same principle to specify the position of any point in three-dimensional space by three Cartesian coordinates, its signed distances to three mutually perpendicular planes (or, equivalently, by its perpendicular projection onto three mutually perpendicular lines). In general, *n* Cartesian coordinates specify the point in an *n*-dimensional Euclidean space for any dimension *n*.

--

It looks like this:

GHB Considerations

We have instructed you to read this article at this point in the GHB but you do not need to understand it 100%. We are making you read it now, because people always ask us "how to find position coordinates" and this will ensure you know where the information is before you get to that point. Try playing around with Cheat Engine and finding your local player's coordinates and then move on to the rest of the tutorial series. You can come back to this post again later.

3D Position coordinates are typically defined as a vec3, a structure containing three floating point values: X, Y & Z which represent 3 axes:

- Forward and backwards
- Left to right
- Up and Down

Z is almost always the vertical axis, X and Y can be interchanged, it doesn't matter.

A vec3 is defined like so:

C++:

```
struct vec3
{
    float x, y, z;
}
```

X, Y, Z just represents a naming convention that has stuck over the years, they normally defined like so:

Z = up and down, vertical axis

X and Y can be left and right or forward and backwards, it's hard to tell unless you know the original orientation. For the purpose of game hacking it doesn't matter.

But there is no guarantee, the developer can use any order they want. 99% of games use Z as the up and down axis.

So coordinates represent the distance across a plane, from the origin, on each axis. But what values can they be?

Again the developer can do whatever they want. They will almost always using floating point decimal values, but sometimes double precision floating points as well. The bigger the game and the more precision required, the larger the range of values can be.

1.0f can represent 1 foot, 1 meter or 1 mile. This is all defined by the developers. It doesn't matter for our purposes, we only need to be able to find them and read their value for most hacks.

How to find Position Coordinates with ReClass the easy way

Your coordinates are usually in the same class as your health. So find your health address, find the health offset and find the entity base address. Learn how in this tutorial.

1. Put your entity base address into Reclass
2. Right click and add 4096 bytes.
3. Now stare at ReClass and jump
4. Look in the float value column for values that increase when you jump but don't change when you stand still
5. Keep scrolling down until you find it
6. Sometimes there will be multiple variables, you have to use trial and error to find out which one is correct.
7. Once you find the Z value, you should see X and Y right above them
8. Your position coordinates should be writeable, so try changing them once you find them.

Always treat your position and angles as a vec3, so the position address/offset should start with the address of X. Do not separate them into 3 different variables , if you find any code that treats them separately, adapt it to use vec3, it's 1000% easier this way.

How to find Position with Cheat Engine the long way

1. Start by looking for Z
2. Stand somewhere on a hill, stairs or near a ladder
3. Scan for a float value using "unknown initial value"
4. Go Up, scan for increased
5. Go Down, scan for decreased
6. Repeat this again and again until you have less than 50 results

This is very easy to do with cheat engine hotkeys, set your numpad plus and minus signs to "scan increased" and "scan decreased".

If this doesn't work, the game is weird. Maybe it uses integers or doubles, try scanning for those types

If that doesn't work then you're either an idiot (LIKELY) or the values are obfuscated (VERY UNLIKELY).

Try this guide to try and find obfuscated values:
How to Find Encrypted or Obfuscated Variables in Cheat Engine

Once you have reduced your scan to less than 50 results:

- Keep re-scanning until the number of results does not change
- Now add them all to your cheat table

Your "correct" position address *SHOULD* be writable.

Select all the addresses in your Cheat Table and hit enter. Try changing all Z variables at once to a value that is plus or minus 5.0f from the original. If your position has changed then the address is in your table. Now to find the correct address, use the divide and conquer method. Select half of the addresses, change them. If your view doesn't change, delete this half. If your view does change, delete the un-selected half. You will eventually only have one correct pitch address. Check the X and Y and make sure they also change when you move around.

You now have your writable position address. Your local writable position and the coordinates of your character, which are sent to all the other players on the server, are not necessarily the same address. Sometimes they use 2 different structures for networking and local information.

How to find Position Offset

Right click on the the X position variable and select "Find what Accesses" and then "Find what Writes", this will open up 2 dialog windows. Then move your mouse.

In both windows you will see stuff like:
mov eax, [**esi** + **0xC**]

In this case, 0xC should be your X position offset and **esi** should be the player base address. But this isn't a guarantee. You have to use trial and error and check all the instructions and have a decent understanding of the game you're working on. The more experience you get, the easier this will be to understand. Once you're a pro, it's really easy, takes 5 minutes for each game.

Once you find the position coordinates, you will want to find a pointer to them. If they are offset from your entity base address, then that is easy, just find a pointer to your local player entity's base address, and then add the last offset for the position and you have your pointer.

At this point you can play around in Cheat Engine in Assault Cube or another very simple game. You don't have to understand this perfectly, but play around a bit. Then move on to the rest of the GHB. Come back to this post later when you need to.

Tutorial How to Find View Angles with Cheat Engine

What are View Angles?

In 3D video games, view angles define the direction a player is facing. They are typically defined as a vec3, a structure containing Pitch, Yaw and Roll as float values which represent angle of degrees around an axis.

In a 2D game it is only 1 value. In a 3D game it is typically represented by 3 floating point decimal values, Pitch, Yaw and Roll.

1574305645683.png

Pitch is the up and down rotation, Yaw is the left to right. But when you're defining the rotation of an object, there is a third variable called Roll. Look straight ahead and while staring in the same direction move your head down toward your left shoulder as if you're using your shoulder as a pillow and do the same on your right shoulder. This axis of rotation is called Roll.

In first person shooters where you do not move on the roll axis, only Pitch and Yaw are important. But view angles are typically represented as a vector or a vec3 with the z variable set to 0. In a flight simulator or a space flight game roll will be used.

A vec3 is defined like so:

C++:

```
struct vec3
{
   float x, y, z;
}
```

X, Y, Z just represents a naming convention that has stuck over the years, they normally defined like so: X = Pitch, Y = Yaw, Z = Roll. But there is no guarantee, the developer can use any order they want. Yaw, Pitch, Roll is also a common order.

These types of view angles are typically represented as Eular Angles. But they are sometimes seen as radians, and often will be converted to radians for certain calculations.

The developer can use any method they want of representing rotation.

There are many ways to represent rotation, a vec3 of Eular Angles is one of the most straightforward, but there are also rotation matrices, quaternions & more. You can read more about them, but be prepared to be confused Rotation formalisms in three dimensions - Wikipedia

So View Angles typically represent the angles in degrees of rotation around each axis. But what values can they be?

Again the developer can do whatever they want. If you look at a circle, normally you refer it to having 360 degrees.

Here is what we normally see with view angles, 99% of games use this methodology:

Pitch = 180 total degrees of rotation, from -90 to +90

Yaw = 360 total degrees of rotation, from -180 to +180

But again, the developer can do whatever they want. Sometimes it's 0 to +90 or 0 to +360.

How to find View Angles with ReClass the easy way

Your view angles are usually in the same class as your health. So find your health address, find the health offset and find the entity base address. Learn how in this tutorial.

Put your entity base address into Reclass, right click and add 4096 bytes. Now stare at ReClass and move your mouse around, look for changes. Look in the float value column for values that change when you move your mouse and don't change when you stand still. Keep scrolling down until you find it. Sometimes there will be multiple variables, you have to use trial and error to find out which one is correct. Your view angles should be writeable, so try changing them once you find them.

How to find View Angles with Cheat Engine the long way

- Start by looking for pitch
- Look straight up into the sky as far as you can
- Scan for a float value between 88 and 92

Look straight down and scan for a value between -92 and -88

Repeat this again and again until you have less than 50 results

If this doesn't work

Start by looking for pitch

Look straight up into the sky as far as you can

Scan for a float value between 178 and 181

Look straight down and scan for a value between -182 and -179

Repeat this again and again until you have less than 50 results

If that doesn't work

Start by looking for pitch

Look straight forward then arc upwards 45 degrees so you're pointing between straight ahead and straight up

Scan for a float value between -362 and +362

Aim down, scan for reduced value

Aim up, scan for increased value

Repeat this again and again until you have less than 50 results

If this doesn't work, the game is weird. Maybe it uses integers or bytes to represent angles

Start by looking for pitch

Look straight forward then arc upwards 45 degrees so you're pointing between straight ahead and straight up

Scan using "all data types"

Scan for a value between -362 and +362

Aim down, scan for reduced value

Aim up, scan for increased value

Repeat this again and again until you have less than 50 results

If this doesn't work, maybe your angles do not exist as Eular Angles and they are only represented as Radians

Radians are between -2 and +2 but are typically between -1 and +1. For the sake of including all values and reducing the amount of scanning you need to do:

Start by looking for pitch

Look straight forward then arc upwards 45 degrees so you're pointing between straight ahead and straight up

Scan for a float value between -2 and +2

Aim down, scan for reduced value

Aim up, scan for increased value

Repeat this again and again until you have less than 50 results

Once you have reduced your scan to less than 50 results:

Keep re-scanning until the number of results does not change

Now add them all to your cheat table

Your "correct" view angles address SHOULD be writable

Select all the addresses in your Cheat Table. Try changing all variables at once to a value within the correct range (-90 and +90 for instance). If your angles have changed then the address is in your table. Now to find the correct address, use the divide and conquer method. Select half of the addresses, change them. If your view doesn't change, delete this half. If your view does change, delete the un-selected half. You will eventually only had one correct pitch address. Check the yaw and make sure it changes when you move your mouse and check the roll, it should be zero.

You now have your writable view angles. Your local writable angles and the angles of your character which are sent to all the other players on the server, are not necessarily the same address. Sometimes they use different structures for networking and local information.

How to find View Angle Offset

Write click on the pitch variable and select "Find what Accesses" and then "Find what Writes", then move your mouse. In both windows you will see stuff like:

mov eax, [esi + 0xC]

In this case, 0xC should be your angle offset and esi should be the player base address. But this isn't a guarantee. You have to use trial and error and check all the instructions and have a decent understanding of the game you're working on. The more experience you get, the easier this will be to understand. Once you're a pro, it's really easy.

Once you find the view angles, you will want to find a pointer to them. If they are offset from your entity base address, then that is easy, just find a pointer to your entity base address, and then add the last offset for the view angles and you have your pointer.

Video Tutorial Cheat Engine How to Pointer Scan with Pointermaps

This Cheat Engine tutorial will teach you how to pointer scan. It will show you a more efficient way of pointer scanning using pointer maps which will save you a lot of time. The video is broken into 2 parts: Traditional Pointer Scanning and Advanced Pointer Scanning. You will learn many tips to make this process incredibly fast.

Why do we use pointer maps?

Finding pointers manually in old games like Assault Cube is simple, but in new games it's insane, there will be million of pointers. Using pointermaps will really speed up the process.

What will you learn?

- How to Pointer Scan
- How to use pointer maps
- How to speed up pointer scanning

Pointer scanning is used to find static pointers to dynamic values such as a health or ammo.

A multilevel pointer which is static is one which will always work because the first address in the chain is always the same (or it's always relative to a module like client.dll). Later you will take these pointer chains and use the FindDMAAddy function in C++ to parse them in your applications.

Considerations

Knowing how to pointer scan is a mandatory skill which you will need to acquire early on in your journey. But it can be one of the hardest things to learn because you have to do it perfectly. If you make 1 little mistake, your entire pointer scan is ruined, so you will have try it many, many times. In this tutorial, we are using the game Divinity Original Sin but it will work on any game. We apologize for using a different game than Assault Cube which unfortunately will make it more difficult to follow.

Requirements

- Cheat Engine - always use the latest version

How To Pointer Scan With Pointermaps

After watching the video, proceed to read below where we will answer some questions you might have. Then try to use this method to find the health and ammo pointers in Assault Cube.

https://youtu.be/nQ2F2iW80Fk

Pointermap Tips

Restarting the game is the fastest way to narrow down your results.
Use my variable naming convention from the video to stay organized.
Use the Cheat Engine scripting functions from the video:

```
getAddress('EoCapp.exe') + getModuleSize('EoCapp.exe')
```

What is a pointer?

A pointer is a variable, which contains an address. This seems simple but it will be the most confusing part of the next 3 months of your life. It's something that will "click" in your head one day and make perfect sense, do not kill yourself trying to understand it right now.

What is the manual method?

We taught this to you earlier in the GHB, remember this?

This method typically yields a logical pointer, but it takes way too long. New games will have pointers with 5+ offsets so it's really not ideal.

What is Traditional pointer scanning?

Traditional pointer scanning is an outdated method of pointer scanning that utilizes only one pointer map, it's what happens by default when you use the pointer scanner. It's not as efficient as Advanced pointer scanning because of the multiple steps you have to do to filter down your results manually.

Why are pointer maps faster?

Instead of scanning the game memory dynamically, it's scanning a list of pointers that was previously saved.

What is Advanced pointer scanning?

Advanced pointer scanning uses multiple pointer maps, it is taught in the second part of this video.
Advanced pointer scanning requires less work and provides faster results.

What is a Multi-level pointer?

A multi level pointer is an address and a list of offsets. At each level, a pointer is de-referenced and an offset is added. This is also called a pointer chain, which ultimately leads to the address of the value you want. Using them enables you to dynamically access variables without using hard coded addresses.

These are all multilevel pointers:

What is a Base address?

Base addresses don't store values like health & ammo. A base address is just a regular pointer which starts your pointer chain. A pointer is just a variable which contains another address. You should always use the module.dll+0xDEADC0DE format for these, not hard coded addresses, because the modules aren't always loaded in the same address. Even if hardcoded addresses work, you should get in the habit of doing it the correct way to future proof your code.

How do you know what the correct pointer is?

The correct pointer follow the game logic. The pointer scanner itself just brute forces them, but if you can narrow down the results to less than 100, the majority of them will be logical. Pointer scanning is great for making personal cheats rapidly, but for genuine game hacking, you will want to reverse the game yourself and understand the actual logic of the pointer chain. Pointers are for ghetto cheats, real hacks use pattern scanning to avoid problems with game updates.

What's better less offsets or more offsets?

This really depends on the game's code but the rule of thumb is: less is better.
More offsets typically results in more chances for the pointer to become invalid due to the increased complexity.
The sweet spot is often 3-4 offsets, I get very suspicious when I find 1-2 level pointers that work.
Save your pointer scans, if your pointer goes bad, you can just grab another one later.

Is it normal to get 9 million results?

Yes it is. After restarting the game 5 times and using all of our tips, you should have it down to less than 3000 on a new game and less than 500 on an old game. If you're getting diminishing returns each time you re-load the game, then stop there, you're just wasting your time.

Having trouble with pointer scanning?

Pointers are the most confusing thing you will have to deal with in the first 3 months of learning. But you will eventually figure it all out. It's recommended to practice each step until you fully understand the process.

If you're having trouble, check out this similar video from the Game Hacking Shenanigans series: Cheat Engine Pointer Scanning Tutorial | GHS105

Pointer Scan Homework

Follow this video using the Assault Cube health address and ammo address. You will fail multiple times because you have to do everything perfectly, so pay attention and follow the video exactly. Getting frustrated is normal, get used to it. Just imagine if you had to learn this without our tutorials?

GHB Navigation

Prev: GHB108 - How to Find View Angles with Cheat Engine
Curr: GHB109 - How to Pointer Scan With Pointer Maps
Next: GHB110 - Game Hacking with Reclass

Video Tutorial ReClass.NET Tutorial - How to Reverse Structures with ReClass

What is Reclass.NET?

ReClass is the premier structure reverse engineering tool and our How to Use ReClass tutorial will teach you everything you need to know about it. ReClass has two main purposes, first you use it to reverse engineer classes, then you export these classes to be used in your coding projects.

ReClass gives you the ability to quickly analyze memory, define data types and define variable names.
It is incredibly useful when reversing player classes and other objects.

GHB Considerations

ReClass.NET is the 3rd most important game hacking tool, behind Cheat Engine and IDA Pro. Therefore it's very important to learn how to use it effectively. You will get plenty of practice using it, as we use it in many chapters of the GHB. Download the complete Visual Studio project from the attachments if you have trouble.

ReClass.NET Tutorial Requirements

- Reclass.NET - always use GH's updated download
- Cheat Engine - always use the latest version
- Assault Cube 1.2.0.2 - use this version for all our tutorials
- "Auto-Padding" for class recreation - You need this for following the video

ReClass Tutorial - How To Reverse Structures

This tutorial has two video parts
Video one you just learn the basics
Video two you learn how to use ReClass classes in your C++

ReClass Video One - How to Reverse Structures

Part one is for noobs that have no idea what they're doing and uses ReclassEX.
This just teaches how to use ReClass as a basic reverse engineering tool similar to Cheat Engine's Data Dissector.

ReClassEx is just an extended version of the original ReClass, but it's still very old. You do not need to download this old version. You should only be using our ReClass.NET download.

We apologize this video was poorly made, but please watch it for a few minutes just to get the idea of how it works.
We will try to replace this video soon.

https://youtu.be/DyqnhSkcVIw

ReClass Video Two - How to Use Generated Classes in C++

Part two is for people who have already finished our internal hacking tutorial and uses ReClass.NET.
You will need the code from the internal hacking tutorial to follow along.
Completed project can be downloaded in the attachments

https://youtu.be/vQb21RM9-5M

Links Referenced in Video Two

- GH's ReClass.net Download - best version to download, it's updated with all commits
- GHB1 - First Internal Trainer - code needed for second video
- ReClassNET/ReClass.NET - official GitHub (outdated)
- C++ "Auto-Padding" for Class Re-Creation

ReClass.NET Tutorial - Generated Classes

Here is an example of the types of classes you can generate using ReClass.NET, this is what we made in the video tutorial

How do we use the exported classes?

For simplicity's sake, here's an example of how you typecast a class pointer to the address at runtime:

C++:

```
Ammo* currAmmo = *(Ammo*)0xDEADBEEF;
```

But the video example is a bit more complicated:

C++:

```
ent* localPlayer = *(ent**)(moduleBase + 0x10F4F4);
```

We're actually doing 2 de-references here so it might be a little confusing. `moduleBase + 0x10F4F4` is a static pointer, not an actual structure.
`*(ent**)` = cast to an ent pointer, de-reference that pointer
Think of this way, `*(ent**)` has 3 stars.
The first one is a de-reference, the last one is a de-reference but the one in the middle is not a de-reference it's just the data type you're casting to which is `ent*`.

Don't worry, this will make more sense as you continue learning. Add and subtract de-reference operators (*) and then run your debugger and see what happens, that's the best way to learn this.

How do we access member variables?

C++:

```
localPlayer->health = 1337;
localPlayer->currWeapon->ammoptr->ammo = 1337;
```

This is what our main function looks like:

See how simple it is when using ReClass exported classes?

ReClass.NET vs. Cheat Engine's Struct Dissector

ReClass.NET is a better tool for 2 main reasons: faster acquisition of data type and C++/C# class exporting.

Default View Types

By default it visually shows every address as 4 different representations:

- ASCII
- float
- 32 bit integer
- hex pointer

This makes it incredibly easy to discover a variable's type. Cheat Engine only shows you 1 value type at a time, so it's much more labor intensive to reverse. On the flip side, Cheat Engine does have the auto-guessing feature but I honestly prefer to define them myself.

If you see words in the ASCII section, then it's a string. If the float representation is a rounded number like 7.1f then it's probably a float. If the integer representation is a rounded number like 100, then it's probably an integer. If it's a pointer, you just hover over the pointer and it'll show you what it points at.

ReClass's Export Class Feature

Import these files directly into your hack and start interacting with them within seconds. We will show you why this is so amazing later.

ReClass Versions

Let's talk about the different versions. First off, you should only be using our latest version of ReClass.NET, don't use anything else. Whenever we mention the word ReClass, we are referring to ReClass.NET.

2011 - DrunkenCheetah released the first version of **ReClass** (C++)
2011 - IChooseYou released **ReClassx64** (C++)
2012 - ajkhoury released **ReClassEx** (Extended) (C++)
2016 - KN4CK3R released **ReClass.NET** (C#)

The main difference between ReClassEx and ReClass.NET is this:
ReClass.NET requires all pointers to point to classes. For example, you cannot define an int pointer.
But this is very easy to solve, just make it a class pointer and define the first variable in that class to be an integer.
Keep this in mind if you're using any ReClassEx tutorials so you don't get confused.

ReClass Bugs

ReClass has had many bugs over the years and the Github release is very outdated, so make sure you use our latest updated version of ReClass.NET which we compiled with all the latest commits. Almost all the bugs have been fixed in this version.

Almost all the bugs are related to the improper padding that gets created when you frequently go back and change the data types of variables. If you reverse a big class, and then go back in and change variables in the middle of the classes, this will often mess up the padding.

Advice on this offset bug

1. Always reverse the classes from top to bottom
2. Avoid going back and change things in between defined offsets

If it gets messed up, you can try to fix it by inserting and deleting bytes, but honestly 8/10 times it just messes it up even worse, which is why I typically just delete the class and start over. You might have thought you fixed it manually and then export it into your hack. If your class is still bugged, you will introduce some insane and confusing errors in your code. This has happened to me a dozen times, and often it takes many hours of debugging to find out the class I exported 3 weeks ago was bugged. 5 minutes restarting the class is better than a frustrating 3 hour debugging session. Save your sanity.

How do you dereference module + offset inside ReClass's address field?

Inside this blue box you can use some ReClass scripting to make your life easier.

To get a module base address do this:
<CSGO.exe> + 0x1337 - this will get the base address of the executable and add 0x1337

To de-reference a pointer do this:
[0x4012ABDE] + 0x1337 - this will de-reference the pointer and then add 0x1337

Combine them like this:
[<CSGO.exe> + 0x1337]

You can use nested operations also:
[<Program.exe> + offset + [<Program.exe> + offset2]]

You can also use subtraction, multiplication & division -> + - * /

Having trouble?

Using these exported classes in an internal DLL is a huge jump from doing basic memory manipulation, this tutorials adds a lot of confusing pointers and de-references. Don't be surprised if this one confuses the hell out of you, you may need to spend a few days working on it.

When you're confused about the pointers/de-references, try adding and removing the stars (de-reference operations) and then running the Visual Studio Debugger, as you step over these instructions, view the data in VS and in ReClass at the same time and compare them. This should help you figure it out, but don't be surprised if it takes you 3 months to really have a solid grasp of what's going on. I still have to add and subtract (*)'s a couple times to get it working even today.

Download the complete Visual Studio project from the attachments if you have trouble, it works perfectly and you can compare your code against it to find your errors. Use a program like WinMerge to compare them.

Homework

Keep repeating the tutorial until you can complete it without watching the video.

GHB Navigation

Prev: GHB109 - How to Pointer Scan with Pointermaps
Curr: GHB110 - ReClass Tutorial - How to Use ReClass.NET
Next: GHB111 - Learn C++ with Step by Step Guide

Tutorial How to Learn C++ Programming Where to Learn C++

The Who, What, Why and Where of Learning C++

We get these 2 questions all the time:
How much C++ do I need to learn for game hacking? Where to learn C++?

Today we will exterminate this line of questioning with a thorough guide

What is C++?

C++ is C with the addition of object oriented programming and a bunch of libraries that make your life easier. C/C++ is a native language, it compiles to assembly and then the linker combines that into an executable. An interpreted language is easier to use than a native language but how it works is abstracted away from you, making some lower level stuff more difficult. Game hacking is low level, so in this case C++ is better overall.

C++ is a high-level, general-purpose programming language created by Bjarne Stroustrup as an extension of the C programming language, or "C with Classes". The language has expanded significantly over time, and modern C++ has object-oriented, generic, and functional features in addition to facilities for low-level memory manipulation

The C++ standard consists of two parts: the **core language** and the **standard library**.

The standard library includes:

- containers such as vectors, queues, stacks
- common algorithms (for_each, binary_search, etc.)
- input/output facilities (iostream, for reading from and writing to the console and files)
- filesystem library
- localisation support
- multi-threading library and more

A large part of the C++ library is based on the Standard Template Library (STL). Useful tools provided by the STL include containers as the collections of objects (such as vectors and lists), iterators that provide array-like access to containers, and algorithms that perform operations such as searching and sorting.

So it's just a danker version of C.

Should I learn C++ or C#? What language should I learn?

Learn C# if you want to make mainly external trainers. Learn C++ for every other reason.

Once you know 1, you can learn the others easily. Everything you learn in one language is transferable to other languages, if you disagree then you're just an idiot.

Start learning how to code immediately, do not waste any time trying to decide. If you need help deciding, just pick C++. The faster you start coding, the faster you will become proficient. Every second that you are not coding, is a waste of time. I'm not joking, several people on GH have wasted years trying to decide which language to pick because they're

unable to commit, this is a severe mental illness (which is very prevalent amongst the game hacking community). I'll take the burden of choice away from you, just pick C++ because I told you to and you'll be better for it.

Who should I learn C++?

People who want to hack games, because it's the best language for hacking games. Anyone who is a real man.

Why learn C++?

Three reasons:

1) Almost every language is referred to as C-like or follows basic C syntax. C is the parent language of 99% of the other languages. If you know C++ you can easily learn any other language, therefore by becoming proficient in C++ you have built the foundation for a long future career in programming, wherever that may take you.

2) C/C++ is very low level, it's as close to assembly as you can get. By learning C/C++ you will have a closer relationship to assembly than if you learned an interpreted language.

3) If you want to bypass kernel anticheat you will need to learn to make kernel drivers, which are made in C/C++.

Game Hacking Bible

When following our Game Hacking Bible the first section is the Start Here Guide, you jump right into the Cheat Engine part of learning game hacking but halfway through the Guide you are instructed to learn the basics of C++. The Bible is specifically made for you to learn coding before reversing engineering.

Why learn how to code before learning more reverse engineering?

Learning how to code is probably the most important part of game hacking. Firstly, if you can't code, then you probably can't reverse engineer either. Reverse engineering is doing the opposite of coding which is much more difficult. If you know how to code, you will understand why the assembly is what it is. There is a direct correlation between the code and the assembly, so the more you understand this relationship the easier it will be to discern the high level logic by looking at the lower level assembly.

Learning how to code and getting good at it is going to change the way your brain works, it will get hard wired for logic and problem solving. The things you learn in coding, you will be able to apply to reverse engineering as well as your REAL life outside of computers. Approaching everything with a perspective based in logic will always give you a good starting point. This is why it's common to hear people say "everyone should learn how to code", not because they think garbage men should make iPhone apps, but because it gives you good like logic skills.

Do I need a book to learn coding?

No. Using the guide on this page is better than any book. Don't ask us what books we recommend, we don't recommend any. Trust me.

The basic flow to the Bible is:

1. Learn the basics of Cheat Engine
2. Learn the basics of C++
3. Practice and get proficient with the basics of C++
4. Make some basic external game trainers
5. Gain expertise in reverse engineering and coding
6. Combine the two skills and hack any game you want

In this way, you are starting off having some fun with Cheat Engine, you learn some C++, combine those skills to create a trainer, then you increase your skills so you don't rely on pasting and offset dumps. The idea is to give you all the skills to be independent and successful, while not boring you to death. I'd much rather force you to become an expert in C++ before you ever touch Cheat Engine but that's just not realistic, no one would do the GHB if that was the case.

Step by Step How to Learn C++

You must practice. Watching a video or pasting some code is not learning. You have to get experience by doing, there is no other way to get good at something. People say all the time "I did the tutorials" but they have no skills, these people are just watching them and pasting the code. You actually have to develop programs by yourself to gain experience.

You don't have to know everything right away, you just need to get your feet wet, you'll learn how to swim later. It will be confusing and complicated and you will feel stupid for the first couple weeks. You MUST suffer in order to get good, it's all part of the experience. Each problem you solve, is a learning experience. If you never have problems, then you aren't learning anything. Suffering is mandatory. Just get in there and start learning, there will be resources to help you fill in the blanks once you get the basics down.

1) Watch The Cherno's tutorial series and do the first 20 videos. It really is the best, trying to make a better video series is impossible.

https://youtu.be/18c3MTX0PK0?list=PLlrATfBNZ98dudnM48yfGUldqGD0S4FFb

2) Practice what you learned

3) Go to learncpp.com and do the first 12 chapters. This will solidify your knowledge from the videos and fill in any blanks. The first 12 chapters are the minimum you need to know to make hacks. You don't need to be an expert, you just need to understand what things are, you can always look them up again later.

4) Practice what you learned

5) Finish the Start Here Guide

Main Resources You Will Use In The Future

There are 4 main resource that contain everything you need:

- The Cherno C++ Video Series
- learncpp.com
- MSDN
- cplusplus.com / cppreference.com

Topics after you've learned C++ (not required during GHB1, come back to this later)

So now you can code, awesome. But you have only scratched the surface, C++ Windows Development is a bottomless pit.

C Headers & C++ Standard Library

You may not be aware of all the things that are available, find out more cplusplus.com & cppreference.com

Modern C++

There have been many versions C/C++. C++98, C++11, C++14. C++17 are the most notable but C++20 is coming soon too. We often use regular C code in game hacking and there is nothing wrong with that, but modern C++ offers soooo much more.

MSDN Guide to Modern C++ - MSDN is a hugely underrated resource

Compiler and Linker Flags
MSVC Compiler Options

C++ Security
Security Best Practices for C++

Windows Development and Windows API
Overview of Windows Development
Windows API Index

Windows GUI
You have many options when it comes to Windows GUI: Win32Api, MFC, GDI, Direct2D, QT and more

After 6 months of learning C++ and game hacking

Video Tutorial What is the Windows API? - Beginner Overview

The Windows API is the programming interface provided by Microsoft for interaction with the Windows operating system. It's a series of header files which expose exported DLL functions to the programmer. These header files are called the Windows SDK, and it's huge!

https://youtu.be/S4lQwJawOzI

The following article is supplementary to the above video but does feature some additional information.

If you're new to programming, the Windows API will be the first thing to really confuse you. You can learn how to use it, but not necessarily understand what it is as a whole. Until now there was no beginner guide describing what it actually is, that's our goal here.

Windows API Beginner Introduction

I'm going to walk you through the Windows API as a concept. Not explicitly *how to use* any part of it, because that's the purpose of the Game Hacking Bible. I aim to address some of the confusion that those of you getting started have concerning the whole concept. I know a lot of beginners see the Windows API as a black box, not really understanding what they're doing. Especially those of us with no programming experience.

This article and video are intended to be a pre-requisite to the first programming sections of the GHB, but feel free to follow along at any stage of learning. It'll help you to build the "bigger picture" in your mind.

Let's answer the most popular questions up front.

How much of the Windows APIs do I need to learn?

Your goal should not be to learn the entire API or to focus on learning the API first. Just learn the specific functions that you need as you begin coding, you'll naturally start off with the most basic ones. As the months go on, you'll learn more and more functions. You don't focus on learning the API because you don't need 90% of it. You should learn more parts of the API only when you need to accomplish a specific goal. Don't know what API to use? Just ask Search.Brave.com "how do I write to process memory?" and it'll tell you what API to use. That's how you really learn it. After 5 years, you'll be familiar with all the common functions and learning a new one is super easy after that.

What is the best resource to learn the WinAPI?

The only resource needed is MSDN and now we also have this tutorial. It's not just documentation, they have thousands of tutorials. There is no Learn the Windows API Tutorial, because a single tutorial that just taught you 10,000 functions wouldn't make sense, MSDN uses a reference/documentation format for this reason. Focus on becoming a software developer who develops for the Windows desktop platform and you will learn what is necessary piece meal (1 thing at a time).

If MSDN doesn't give an example of how to use the API, then search GitHub for it and you'll find real world code examples in the language you define, that's always very helpful. Your goal should not be to learn every API, it should be

"learn the functions necessary for each task I want to accomplish" and as your programming career continues, you will slowly build up your knowledge of the functions.

But what IS the Windows API?

The Windows API is a set of functions which are exported by DLL files authored by Microsoft and installed on your PC. The .h header files in the Windows SDK define for us how to call those functions. When we include a .h and link to the associated DLL file, we can use those Windows API functions.

What is an API

Before we can discuss the Windows APIs, we need to know what an API is. Put simply, an API is an interface that allows two pieces of software to interact with one another. In our case, the API is provided by the Windows operating system. So the Windows API is a means by which you can programmatically interact with your operating system. Thinking of it this way, it starts to make sense that the Windows API is quite large!

You may have heard of common DLLs like `user32.dll / kernel32.dll / ntdll.dll`, but you may not know how they relate to the code that you're writing. In your code, you include header (`.h`) files, but these files don't contain the code that makes the Windows API work. These header files contain only the definitions of functions and types you're able to call in your program.

The actual functionality exists in pre-compiled form on your system. When you call `ReadProcessMemory()` for example, the DLL (*dynamic link library*) which implements this function (`kernel32.dll`) is *dynamically linked* to your program. When your program is executed it will call the function that already exists within the DLL.

Using the ReadProcessMemory documentation as an example, you can see at the top of the page the WinAPI topography which should help you understand it's organization.

If you scroll down to the bottom you will see that the header that provides this functionality is `memoryapi.h` and the function is exported by `Kernel32.dll`.

What is kernel32.dll?

It's a Windows library that represents the core of the Windows kernel, it's one of the lowest level libraries and it provides many functions that the OS needs to get up and running. It provides the basic management of memory, processes and input/output required. As a developer, it's just a library of functions that we may use.

What is ntdll.dll?

NTDLL is not just an API library, it's a platform on top of which the rest of the system operates. It's also your interface with the kernel, which you can't speak to directly from user applications. You interact with it via the ntdll.dll exports.

What is user32.dll?

This library provides the basic functionality for managing your Desktop, Windows and menus. It gives access to the APIs needed to create the standard user interfaces we're all familiar with.

What is gdi32.dll?

GDI or Graphics Device Interface is historically the method used to do low level drawing of the Windows user interface. Much of the user32.dll functions use gdi32.dll to carry out it's tasks.

So how can we actually use the Windows API in our projects? Well, we need the Windows SDK (*software development kit*). You can download it here, but if you use Visual Studio it is already included and properly linked with your IDE.

Win32 vs WinAPI

The WinAPI (just another word for Windows API) is huge and it includes additional things like DirectX, but when referring to the Windows API we're typically talking about the basic set of APIS from ntdll.dll, kernel32.dll and user32.dll specifically.

Win32 which is still just part of the WinAPI, on the other hand is typically used to refer to the Win32 GUI API which is made up of user32.dll, gdi32.dll and others. When you see the word Win32, think old school dialog boxes, MessageBoxA, WinProc & message handlers.

A Closer Look at the Windows API

Now armed with a basic understanding, let's dive into the details of the Windows API.

The Structure of the API

The API has no inherent structure on your file system, but is categorically organized in Microsoft's official documentation. Most of this will not be applicable to hacking, so there's no need to dive too deep into this. I just want to illustrate that the Windows API is not some arcane hacking utility, but rather an integral part of developing applications for Windows. The following image shows the core categories that the API can be broken down into.

WinAPI Data Types

This section and the next could be considered the most important parts of this article. This is what throws *everybody* off when they first start using the Windows API. Have you ever wondered what a `DWORD` is supposed to be, or what a `HANDLE` is?

The Windows API is littered with typedefs, which is pretty much a `#DEFINE` but for types. The programmers have decided that 'HANDLE' can be used to represent a void pointer, and the pre-processor will handle changing it over before compiling. But why? It seems like a useless layer of complication, and today that's an accurate assessment.

However, these abstractions have two purposes. The first of which is backwards-compatability, this is a natural outcome of supporting the same API through so many generations of hardware. Back in the day of 16-bit Windows there was a distinction between a near and a far pointer, hence the two types `LPCWSTR` and `PCWSTR` (*long?* pointer to c wide-string). Both of these are now identical.

The second reason is far less technical. In the days before online documentation and modern development tools, these names served as helpful abstractions. It's easier to refer to something as a HANDLE than a void pointer. These days C primitives are far more readable and portable, but it hasn't always been the case. You have to remember that this API is *old.* Older than a lot of you reading this, and the one writing this!

What is a DWORD?

Open Visual Studio, type in DWORD and then right click it and select "Go To Definition". This gives you a list of all the common Windows Typedefs. As you can, a DWORD is just an unsigned long which is a standard C++ data type. Use this same technique to learn what everything is in the Windows API. It's really that simple, which is why we show this in our GHB videos.

WinAPI Naming Conventions

The strange function names stem from some of the same reasons as the types, but are a bit less straight-forward. To start with, some functions have prefixes like Cm, Nt, and Zw. These correspond to the kernel components to which the function belongs. For example, a function that begins with 'Cm' belongs to the configuration manager.

Function Name Suffixes

Sometimes you'll come across functions with three different variations. One generic, one with the suffix 'A' and one with the suffix 'W'. These relate to encodings, the 'A' meaning ANSI and the 'W' means WIDECHAR but better is better described in 2022 as UNICODE. The generic version (aka. the one without any suffix) can be used with both encodings. Please note that standard procedure in 2022 is to always use Unicode, but hackers often use the regular ANSI functions.

Finally, you may come across a function that has an additional variation with the suffix 'Ex'. This stands for *extended* and generally takes more arguments, giving you *extended control* of how said function executes.

This can be extremely confusing for noobs, but it's very simple with our explanation below:

- CreateWindow - This is just a macro
- CreateWindowA - ANSI
- CreateWindowW - Unicode
- CreateWindowEx - Extended Macro
- CreateWindowExA - Extended ANSI
- CreateWindowExW - Extended Unicode

CreateWindow Macro:

The macro says: use the Unicode function if Unicode is enabled

The problem is with this many options, you don't know which to use. I'll make it easy for you: always use the Unicode versions. Why? So your software will work with all languages, which is why Unicode is now standard.

Conclusion

The above information is very useful to know, but it's all theory. You will not need to remember all of the quirks of the API when you're writing code. The only thing you will need to understand is what your own code is doing. You are not casting magic spells in your code! You are using the API as a tool to accomplish your goals, it's just the easiest means to an end.

When you're stuck on something, here are some tips to get you up and running. You should first refer to Microsoft's documentation. It's excellently laid out and code examples are provided everywhere. This is where you will find the answers to almost every question you have.

If you need to quickly see what kind of arguments a function takes, or what some weird type really is, in Visual Studio you can right click on anything and press "Go To Document". This will let you view the actual source that you've imported it from. The same can be accomplished by holding 'CTRL' and clicking.

Did you know that Microsoft doesn't actually give us the full WinAPI? Learn about the Undocumented Windows Functions & Structures

To learn more about the Windows API we recommend these excellent resources

- Geoff Chappell: NTDLL
- MSDN Windows API Index
- Wikipedia: Windows Architecture
- Wikipedia: Windows Library Files
- Microsoft Fandom: Windows API

Tutorial Understanding Strings Unicode, TCHAR, MBCS

Everyone gets confused by strings in the beginning of their coding career, I was certainly not an exception.

Read these articles to understand them thoroughly.

You do not need to be an expert at this time, you just need to understand the basic stuff and then you can use these articles as reference later after you get more experience, when you come back to these articles in 6 months they will make much more sense.

We are making you read them now, because unicode/multibyte character set is gonna cause you grief early on, if you have no conception of what they are. If we don't force you to read them, you will be asking stupid questions about them by the time you touch the first C++ tutorial.

Just read through these quickly and bookmark this page for later

- The Private Lives of Strings — Cunning Planning
- Unicode & Windows - CunningPlanning
- Unicode — Cunning Planning
- The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)
- The Complete Guide to C++ Strings, Part I - Win32 Character Encodings - CodeProject
- String and character literals (C++)
- UTF-8 Everywhere

Tutorial Get Module Base Address Tutorial dwGetModuleBaseAddress

How do you get the base address of a process or module in C++? dwGetModuleBaseAddress is the function which does this. It has been used for many years, but we will improve upon it in this tutorial by creating a new function simply named GetModuleBaseAddress. This module base address tutorial will teach you how to get the address of any module and how to use simple C++ and C# versions of the function.

If you just want to paste the function, scroll down.

ac_client.exe+0x109B74

In Cheat Engine this syntax isn't a problem because CE will always get the address of client.dll at runtime. But when you transition to making real hacks, you have to resolve the module's address yourself using our function. That's what this tutorial is all about.

GHB Considerations

We are making you read this tutorial before we show you how to do this in a video because it's the 2nd most common question when learning game hacking. We want you to know this tutorial is here, so you can come back to it later if necessary. This is officially taught in video form in our first C++ External Trainer Tutorial. Reading through this article first, will set you up for success before you start the video. So don't focus too hard on understanding this perfectly right now, it should all click when you watch the video later, if it doesn't, come back and read this.

GetModuleBaseAddress Tutorial

If you're completely unfamiliar with this, we will explain everything in an easy to understand way. We use this function in all future tutorials, so it's important that you learn it now.

What will you learn?

- How to find the module base address in Cheat Engine
- How to find the base address of a module using C++
- How to navigate through the module list
- How to obtain a list of modules for a specified process

Why do you need these skills?

Why would you need to know the base address of a module? Everything you do in game hacking will involve relative offsets, the module will not always be loaded into the same base address. To compensate for this you are required to find the base address of the module at run time using the Windows API. The ToolHelp32 library provides us the functionality needed to loop through the modules and get their base addresses.

You should never use hard coded addresses, this is a bad behavior that will result in problems down the road. That's why we always teach you to use relative addresses in the form of binary.exe+0xBADF00D, when resolved at runtime using our function you will never have a problem.

What is a module?

An .EXE or .DLL file (both are Portable Executable files) are referred to as ***binaries*** when they are on disk but they are called ***modules*** when loaded into memory.

Why don't modules have the same address every time?

In the PE header for each binary, there is a field called ***ImageBase*** which defines the address it should be loaded. But what happens when another module is already in that location? Then the Windows loader will find a different address for it, this is called DLL re-location. That's why it's referred to as the ***preferred*** image base.

What about ASLR?

You can imagine that exploiting vulnerabilities is quite simple when the things you're trying to exploit are in the same address on every computer, every time the program runs, right? That's why Microsoft added Address Space Layout Randomization as a security feature. With ASLR enabled, the address where a module is loaded is randomized. But in reality this is a very poor security feature, getting the address at runtime is quite simple as we will show.

.EXE vs .DLL Preferred Image Base

When an EXE is loaded, the virtual address space of the process is empty, so it can always load into it's preferred image base address.
The only time an .exe isn't loaded into the ***PreferredImageBase*** is when ASLR is enabled on the OS and the DynamicBase flag is set to enable the OS to randomize the virtual address of the module.

But none of this really matters, ASLR is standard procedure now so we just assume it's enabled and use our function to ensure we get the right address.

Visual Example Using Cheat Engine

To view all the modules loaded by a process using Cheat Engine:

1. Click Memory View
2. Click Tools
3. Click Dissect PE Headers

   In the resulting window click on any DLL or .EXE and then expand *PE Header* to show this:

#3 is the preferred image base from the PE header and #4 is the current base address. They are the same in this example because ASLR is disabled and it's an EXE.

Common mistake: Make sure you click "info" each time you select a new module or it won't load the new results.

Bonus Article: Why is 0x00400000 the default imagebase?

Our GetModuleBaseAddress Function

There are actually multiple different ways to get the base address. The Windows API function named **CreateToolhelp32Snapshot** is the easiest and learning it will enable you to use this entire library as most of the functions work in the same way.

Why not just use the old dwGetModuleBaseAddress?

The old function uses DWORD so it only supports x86. Our function uses uintptr_t which will resolve to a x86 or x64 address depending what architecture you build for. This makes it work on both, so you only need one function.

C++ GetModuleBaseAddress

C++:

```
#include <windows.h>
#include <TlHelp32.h>
```

```
uintptr_t GetModuleBaseAddress(DWORD procId, wchar_t* modName)
{
    //initialize to zero for error checking
    uintptr_t modBaseAddr = 0;

    //get a handle to a snapshot of all modules
    HANDLE hSnap = CreateToolhelp32Snapshot(TH32CS_SNAPMODULE | TH32CS_SNAPMODULE32,
procId);

    //check if it's valid
    if (hSnap != INVALID_HANDLE_VALUE)
    {
        //this struct holds the actual module information
        MODULEENTRY32 modEntry;

        //this is required for the function to work
        modEntry.dwSize = sizeof(modEntry);

        //If a module exists, get it's entry
        if (Module32First(hSnap, &modEntry))
        {
            //loop through the modules
            do
            {
                //compare the module name against ours
                if (!_wcsicmp(modEntry.szModule, modName))
                {
                    //copy the base address and break out of the loop
                    modBaseAddr = (uintptr_t)modEntry.modBaseAddr;
                    break;
                }

            //each iteration we grab the next module entry
            } while (Module32Next(hSnap, &modEntry));
        }
    }

    //free the handle
    CloseHandle(hSnap);
    return modBaseAddr;
}
```

Function Call

To call the function it looks like this:

C++:

```
uintptr_t clientDLLBaseAddr = GetModuleBaseAddress(ProcId, L"client.dll");
```

How does it work?

CreateToolhelp32Snapshot returns a handle to a snapshot, which contains all the information for all the loaded modules in the designated process. You then iterate through all the loaded modules using Module32First and Module32Next. During iterations, it compares the process name to find the correct module. It then returns the correct module address, if it's not found, then it returns zero.

You input the ProcessID and the name of the module and it outputs the address of the module. In order for your project

to function it must be set to UNICODE, if you're are experienced you can easily change this to be compatible with Multibyte Character Sets

But wait, we're missing ProcId?

This is just the Process ID. Don't worry, the `GetProcID()` function will be taught to you in the video later.

Unicode vs MBCS vs TCHAR

Notice the `wchar_t* modName`?
wchar_t means we're using unicode so you must pass in a unicode string which we do using the L macro to mark our literal string as unicode:
`L"client.dll"`

If you want to use MBCS use `char*` instead of `wchar_t*`.
If you want to use TCHAR change to `TCHAR*`

You will also have to change `_wcsicmp` to a comparable string compare function: `_stricmp` or `_tcsicmp`

Keep in mind that using Unicode is standard procedure in 2022, we try to do that in all our tutorials but some use regular ANSI/ASCII encoding.

How to Get Module Base Address Internally?

Getting a module base address in an internal hack is much easier, you just call GetModuleHandle and you're done.

C++:

```
uintptr_t clientDLLBaseAddr = (uintptr_t)GetModuleHandle(L"client.dll");
```

MSDN Links
Remember everything is on MSDN if you're confused.

- MODULEENTRY32
- CreateToolhelp32Snapshot
- Module32First
- Module32Next
- wcscmp
- CloseHandle

Having trouble?

As we said earlier, this is explained in extreme detail later in our first External C++ Trainer Tutorial.
We just want you to get familiar with this topic before we present it in video format.

If you haven't been coding for long, this will be very confusing.
Did you follow the Learn C++ Guide first?

Confused about the Windows API?
Read: Overview of the Windows API

How did we know how to do this?
MSDN is where you learn everything about the Windows API. It's actually explained write on their website:

- Taking a Snapshot and Viewing Processes
- Traversing the Module List

Is your Process ID zero?
Make sure you run as admin

Misc. Issues?
Make sure you target the same architecture as the game (x86 vs x64)

Looking for a C# version?
We've got everything you need: C# GetModuleBaseAddress

Homework
No homework here, if you want to checkout the MSDN resources to solidify your knowledge, go right ahead. Otherwise proceed to the next chapter.

GHB Navigation
Prev: GHB112 - Understanding Strings Unicode, TCHAR, MBCS
Curr: GHB113 - Get Module Base Address Tutorial
Next: GHB114 - FindDMAAddy - C++ Multilevel Pointer Function

Why do we use dwGetModuleBaseAddress?
When a binary gets mapped into memory, the PE header defines it's preferred imagebase, which is the address in virtual memory in which to load the module. All addresses in the module are addresses relative to this module.

An .exe always gets loaded first so it's always loaded into it's preferred imagebase. So addresses in this module are sometimes referred to as being "static" because they're always in the same place.

DLL files get loaded after the .exe and are loaded 1 by 1 as the import tables are parsed. All imports that rely on libraries, will have those libraries loaded. Because of this, one DLL may already exist in the preferred imagebase of another DLL. For which reason, the new DLL will be mapped to another address.

Due to this, never assume an address in a DLL is static.

BUT, you should NEVER use static addresses for any reason. This will only make you feel stupid later when you f*ck it up. If you start writing code using static addresses now, you will start a poor coding practice that will f*ck you later, we've seen it happen 10,000 times and we're sick of it.

ALWAYS interact with all addresses as relative to the base address of a module. Always get the module base address and then add the relative offset, this way you NEVER have a problem.

But wait, there's more!

Address Space Layout Randomization
Viruses & exploits were easy to make when the address of something was always in the same place. So most Operating Systems added ASLR as an extra layer of security, but this was invented 25 years ago and is basically worthless in 2021. ASLR makes the .exe load into a random virtual address.

All you have to do to bypass this is, get the module base address at runtime and add the relative offset.

moduleBaseAddress+0x9ADBD

Sauerbraten has ASLR enabled
In static analysis tools, the tool always "maps" the binary into it's preferred imagebase, even with ASLR enabled. So the addresses won't match if the .exe has ASLR enabled.
So you just rebase it to 0x0: Tutorial - How to rebase a module in Ghidra & IDA Pro
Then sauerbraten.exe is 0x0 in your static analysis tool.
So if you want to view sauerbraten.exe+0x9ADBD (CE) in Ghidra/IDA after rebasing, just visit 0x9ADBD

This is why me made this dwGetModuleBaseAddress tutorial, it's very confusing for noobs and we get the question all the time.

How to Get a Module Base Address in Cheat Engine?

MZ-Start is the address of the module as it is currently loaded into memory. Preferred ImageBase is parsed straight from the PE Header and is the location that it prefers to be loaded into. If this memory address is already taken, it will relocate.

When an .exe is executed, the windows loader create a process for it and give it it's own virtual memory space. The loader loads the executable into memory and then any .dlls that are called by the process. The PE header for the .dll defines a ImageBase address. The windows loader will try to load the .dll into the virtual memory space of the process that requires it. If that space is already occupied, it will be loaded into a different location. If this happens hardcodes addresses in our hacks will not work.

Now let's say we have a pointer:
ac_client.exe + 109B74

Now the ImageBase pulled from the PE header of ac_client.exe is "00400000"
We can only have one executable for each process which is an empty memory space until ac_client.exe is loaded. There is nothing blocking ac_client.exe from loading into it's ImageBase. So the base address of a .exe is always the same.

The ONLY time when a .exe isn't loaded into the imagebase stored in the PE headers is when ASLR(Address Space Layout Randomization) is enabled on the OS and the DynamicBase flag is set to enable the OS to randomize virtual address of the module.

We can just evaluate this before placing it in the code.
ac_client.exe + 109B74
00400000 + 109B74
509B74

This is the definition of a static address, it may be relative to the base address of an executable in the binary on disk, but it is always static in memory after relocations have occured.

But for .DLL's that can be relocated:

"server.dll + 004EE83" works in Cheat Engine because Cheat Engine evaluates the address of server.dll. CE will get the address of server.dll and replace it with the adress that the module is loaded.

So lets say the address of module server.dll is 0x10000000, cheat Engine will evaluate:

server.dll + 004EE83
0x10000000 + 004EE83
1004EE83

The above evaluation is done by cheat engine while the program is running.
But when you are trying to use this in an external trainer you need to evaluate "server.dll" + 004EE83 yourself. There are multiple ways of doing this and we will discuss one of them now.

Get Module Base Address Function

To do this externally you can use this function that has been widely used named dwGetModuleBaseAddress.
Basically it uses the windows API CreateToolhelp32Snapshot to get a snapshot of all loaded modules for the given process, it then iterates through all the loaded modules and finds the module with the module name you give it. It returns a uintptr_t to the module address. You input the ProcessID and the name of the module and it ouputs the address of the module.
You must set your project to UNICODE for this to work, if you're not a noob you can easily change this to work with MBCS.
Check the original post in this article for the actual code, we've merged all our best information into 1 big post at the top.

You first have to read the base address and then the offset. Depening on wether you're making an internal or an external hack you have to either dereference addresses or use ReadProcessMemory.

Internal would look like this:
C++:

```
DWORD ac_client = (DWORD)GetModuleHandleA("ac_client.exe"); //getting the base address of
the process
DWORD base = ac_client + 0x10F4F4; //calculate the full base address
DWORD address = *(DWORD*)base + 0xF8; //dereference ("read") the base address and add the
offset
//now you can write to the address:
*(DWORD*)address = 1337;
```

External (assuming you already got the process handle):
C++:

```
DWORD ac_client = dwGetModuleBaseAddress(pid, "ac_client.exe"); //check
https://guidedhacking.com/threads/get-module-base-address-tutorial-
dwgetmodulebaseaddress.5781/ for this function
DWORD base = ac_client + 0x10F4F4; //calculate the full base address

DWORD address = 0;
ReadProcessMemory(hProc, (void*)base, &address, sizeof(address), nullptr); //read the
base address
address += 0xF8; //add the offset

DWORD value = 1337;
WriteProcessMemory(hProc, (void*)address, &value, sizeof(value), nullptr); //write
```

Keep in mind that you should add error checking to the code above (eg. if ReadProcessMemory fails or *(DWORD*)base returns 0). Otherwise you will probably crash the process.

Problem with dwGetModuleBaseAddress Tutorial

Hello, I'm about to write a tool for an old game Anno 1503 (32 bit) but I'm having trouble getting the module base addres.

I have already written similar tools with scripting languages, but this I want to write in c to have more possibilities through the efficiency. So far, I did not have to determine the base address myself. I picked out a pointer with cheat engine, which I want to implement first. I tried the method presented here to find out the base addresses of the modules. I had to notice that some modules are not listed, the cheat engine shows me. In addition, cheat engine displays other addresses in the listing than it uses itself. And my pointer uses a module that does not appear in the list at all.
I changed my code a bit to output all addresses. I also use simple strings because "modEntry.szModule" is recognized as int *. However, that works fine. Only the found addresses, which are mostly found in ce, are not the correct base addresses.

My function:

C:

```
DWORD GetModuleBaseAddress(DWORD dwProcId, const char* szModuleName)
{
    HANDLE hSnap = CreateToolhelp32Snapshot(TH32CS_SNAPMODULE | TH32CS_SNAPMODULE32,
dwProcId);
    if (hSnap != INVALID_HANDLE_VALUE)
    {
        MODULEENTRY32 modEntry;
        modEntry.dwSize = sizeof(modEntry);
        if (Module32First(hSnap, &modEntry))
        {
            do
            {
                if (!strcmp(modEntry.szModule, szModuleName))
                {
                    CloseHandle(hSnap);
                    return (DWORD)modEntry.modBaseAddr;
                }
                printf("0x%x - %s\n",(int)modEntry.modBaseAddr,modEntry.szModule);
            } while (Module32Next(hSnap, &modEntry));
        }
    }
    CloseHandle(hSnap);
    return 0;
}
```

The output looks like this:

The address to my pointer is calculated from "THREADSTACK0" -00000560.
I can not find this module in my list. Also CE does not show it in the whole list.

When I try to put in a module from the list, CE shows a different base address than the one in the list:

What is missing to calculate the correct address and why do not some of the modules from CE appear on my list? What is "THREADSTACK0" if not a module?

Source Code FindDMAAddy - C++ Multilevel Pointer Function

How to Use a Multi-Level Pointer in C++?

Do you have a multilevel pointer from Cheat Engine that you're trying to use in a C++ hack? Our FindDMAAddy function is exactly what you need. Learn how to use the FindDMAAddy function to follow the pointer chain and get the dynamic address of the variable you need using this article.

Easy to copy and paste code is found near the bottom of this post.

What is a multi-level pointer?

A multilevel pointer also called a pointer chain, is just a base pointer with a list of offsets. When you de-reference each pointer in the chain and then add each offset, you end up with the "end address" of the pointer, which is the address of the variable you want to read from or overwrite.

This is what an annotated multi level pointer from Cheat Engine looks like, if this looks confusing, don't worry, it's all explained in our video tutorials.

Very Common Questions

- How do I find the end address of a multi level pointer using C++?
- How to use a multilevel pointer in C++?
- How to calculate pointers with offsets?
- How to convert Cheat Engine pointer to C++?

These questions have been asked hundreds of thousands of times on game hacking forums and other websites. We made this article to quickly & easily answer these questions.

What is FindDMAAddy?

FindDMAAddy or the "Find Dynamic Memory Allocation Address" function was created more than 10 years ago, published on MPGH and primarily promoted by Fleep, the original owner of GH who gave it this now iconic name. FindDMAAddy automates the parsing of multi level pointers so you don't have to do it manually. We have taken this old function and updated it to 2022 standards, meaning it works on x86 & x64.

GHB Considerations

This article is a chapter of our Game Hacking Bible, which basically just tells you to use our FindDMAAddy function if you need to parse multilevel pointers. You are being pointed to this article before we show it to you in a video, because people often get ahead of the GHB and start asking questions we haven't explained yet. Rather than waiting for you to ask the question, we're giving you the answer now so you know where to find it when you need it. So just read this article and move on to the next chapter, you can comeback later when you need to paste this function. This is a very important function, which you MUST learn how to use, all beginner hacks require it.

In-depth Multi Level Pointer Explanation

A regular pointer points to only one address. But when it's accompanied by a list of offsets you can walk that pointer chain to find the *final* or *end address*, which is the address of the variable you want.

By walking this pointer chain you're actually just replicating the actual logic the game uses to access that variable. When you find a pointer in Cheat Engine, what you're finding is a path from one address to another using pointers and relative

offsets. This is how computers locate and act on data stored in memory. It's not magic!

The logic behind this is based on 2 important features of modern object oriented programming:

1) Applications are made memory efficient by dynamically allocating objects only when needed. Pointers are used to point at these objects, so you can access them via the pointers.

2) Classes can contain member variables that are pointers, specifically pointers to other objects in memory.

The baseAddress or first pointer in the pointer chain is generally one that exists in static memory, meaning it's always located at the same address or can be accessed using a relative offset from the base address of a module.

A Simple Example of Real Multi-Level Pointer Logic

C++:

```
//defining a class
class PlayerClass
{
    public:
        int health;
        int armor;
        char* name;
}


//creating a pointer capable of pointing at a PlayerClass object
PlayerClass* localPlayer;

//creating a new dynamic PlayerClass object and making our pointer point at it
localPlayer = new PlayerClass();
```

Let's explain shortly

class PlayerClass
Health is offset 0x0 and armor is offset 0x4, because an integer is a 4 byte variable in this example. The name variable is a pointer to a char array at offset 0x8, this char array actually contains the ASCII letters of the name.

PlayerClass* localPlayer
PlayerClass is the name of the class, the "*" tells the compiler that this is a pointer to an object of type PlayerClass and it's identifier is "localPlayer". In this example it is not initialized. Meaning, no memory has been allocated for it. It's also technically a dangling pointer right now because it hasn't been initialized to anything. It's best behavior to initialize the value to nullptr or zero.

localPlayer = new PlayerClass
When the player starts a game, the localPlayer object is allocated a spot in memory on the heap and the pointer is assigned the address of the object. So our pointer now points at our dynamic object.

In this example, the address of the localPlayer object wouldn't be consistent and you would need to find a pointer to it. If you used Cheat Engine's "Find What Accesses this Address" or the pointer scanner on the name variable, you would find a multi level pointer where the BaseAddress is the address of `localPlayer` and offset 0x8 leads you to the name pointer, which when de-referenced yields you the address of the actual char array containing the name.

It simply uses the same logic that the assembly code does to find the address of the variable. In the source code of this

application if you wanted to use the name variable you would use the pointer arrow "->" (called the *structure de-reference operator*) like this:

Code:

```
localPlayer->name
```

When the compiler compiles this code into assembly the logic it creates basically looks like:

1. Dereference the localPlayer pointer to get the dynamic address of the object
2. Add offset 0x8 to get to the name pointer
3. Dereference the name pointer to get the dynamic address of the char array

So how do you do it in C++?

We can manually do it for every pointer by doing something similar to:
C++:

```
ReadProcessMemory(handle, (LPVOID)pointeraddress, &newpointeraddress,
sizeof(newpointeraddress), NULL);
ReadProcessMemory(handle, (LPVOID)(newpointeraddress+ offset[0]), &newpointeraddress,
sizeof(newpointeraddress), NULL);
ReadProcessMemory(handle, (LPVOID)(newpointeraddress + offset[1]), &newpointeraddress,
sizeof(newpointeraddress), NULL);
ReadProcessMemory(handle, (LPVOID)(newpointeraddress + offset[2]), &newpointeraddress,
sizeof(newpointeraddress), NULL);
```

But you don't want to waste your time doing that for every pointer, instead you make a function that does it for you. We emulate that exact logic in our FindDMAAddy function.

C++ FindDMAAddy Source Code

This is our official function which works on x86 and x64 code because we use uintptr_t to represent addresses. uintptr_t compiles to the correct variable size under each architecture, that's why we use it. It's 32 bits on x86 and 64bits on x64, so you just compile your hack as the same architecture as the game uses and it will always be the correct size. This is a hybrid of code assembled by Fleep, Rake & IXSO.

External Function

Use this in external hacks/trainers. It uses ReadProcessMemory and requires a process handle (use OpenProcess to get the handle).

C++:

```
uintptr_t FindDMAAddy(HANDLE hProc, uintptr_t ptr, std::vector<unsigned int> offsets)
{
    uintptr_t addr = ptr;
    for (unsigned int i = 0; i < offsets.size(); ++i)
    {
        ReadProcessMemory(hProc, (BYTE*)addr, &addr, sizeof(addr), 0);
        addr += offsets[i];
    }
    return addr;
}
```

To get the Current Weapon's Ammo address using our example:

C++:

```
uintptr_t ammoAddr = FindDMAAddy(hProcess, dynamicPtrBaseAddr, { 0x374, 0x14, 0x0 });
```

At this point you can use ReadProcessMemory to read the value at that address, or you can over-write it with WriteProcessMemory.

Internal Function

When you're internal (injected DLL or shellcode), you don't need to use ReadProcessMemory because you have direct memory access.

C++:

```
uintptr_t FindDMAAddy(uintptr_t ptr, std::vector<unsigned int> offsets)
{
    uintptr_t addr = ptr;
    for (unsigned int i = 0; i < offsets.size() ; ++i)
    {
        addr = *(uintptr_t*)addr;
        addr += offsets[i];
    }
    return addr;
}
```

C# FindDMAAddy Function?

Do you mainly use C#? Don't worry we have you covered: C# FindDMAAddy Function

Python FindDMAAddy?

Yep, we got that covered also: Python FindDMAAddy Function

Our GHB FindDMAAddy Video Tutorial

The next chapter of the GHB will teach you this function in extreme detail, in addition to many other skills. Most people who visit this article are just here to copy and paste the function.

Still confused about what a Multilevel pointer actually is?

Most people are but don't worry in a few months it'll all make perfect sense. The next chapter in the GHB is a video tutorial that explains everything. Even after this video, people were still having trouble understand what a multilevel pointer really is, so we made another video that should hopefully help you: WTF are Multi Level Pointers

Homework

You are meant to quickly read this article and then move on to the next chapter.

GHB Navigation

Prev: GHB113 - Get Module Base Address Tutorial
Curr: GHB114 - FindDMAAddy - C++ Multilevel Pointer Function
Next: GHB114-2 Visual Studio Run as Administrator + Manifest File

Video Tutorial Quick Tip: Visual Studio Run as Administrator + Manifest File

Just 2 quick tips, basic knowledge for coding applications in Visual Studio that are particularly useful when making external hacks that use OpenProcess and require administrator access. When you run them under the VS debugger, you don't have admin rights so your hacks won't work. So you have to do this trick to make it work. You will also force UAC to prompt the user for admin credentials if they download it.

Learn How to run Visual Studio as an administrator every time and how to compile your applications to run as admin when executed by your end users. By using a manifest file which is basically a resource built into the binary, it will prompt User Account Control to elevate the process to administrator mode.

GHB Considerations

This is a chapter in the Game Hacking Bible, just quickly watch this video so you know what this is talking about. Not running your hack as admin is the #1 cause of noob problems. We are making you watch this before you need it so you don't waste your time trying to solve a problem you don't even know exists yet. Always run Visual Studio and your hacks as administrator.

How to Run Visual Studio as Admin Video

https://youtu.be/rY0pEfZ9GTg

Run Visual Studio as Administrator

This will let your app run as admin when debugging using visual studio.

Windows 8 & 10:

1. In the start menu type "devenv.exe"
2. Right click it and select "open file location"
3. Right click the actual file devenv.exe and select "troubleshoot compatibility"
4. Click "Troubleshoot Program"
5. Select "The program requires additional permissions"
6. Click "Next" then Test the Program
7. Click Next
8. Save the Settings
9. Click "Close"

Force Application to Run as Admin using Manifest File:

Project Properties -> Linker -> Manifest File -> UAC Execution Level -> Run as Administrator

GHB Navigation

Prev: GHB114 - FindDMAAddy - C++ Multilevel Pointer Function
Curr: GHB114-2 Visual Studio Run as Administrator + Manifest File
Next: GHB115 - How to Hack Any Game Tutorial C++ Trainer #1 - External

Video Tutorial How to Hack Any Game Tutorial C++ Trainer #1 - External

How to Hack Any Game Tutorial - First C++ External Trainer

This beginner C++ tutorial builds upon our previous tutorials, to teach you how to make your first C++ External Trainer. This traineruses pointers and addresses that we located previously with Cheat Engine. This is **the** essential How to Hack Any Game with C++ tutorial that you must complete before moving to more complicated topics. It replicates everything Cheat Engine does by simply parsing multilevel pointers and overwriting the value contained at the final address.

What You Will Learn

- How to find the CurrentWeaponAmmo Pointer
- Visual Studio Project Setup and organization
- PreCompiled Headers
- **GetProcId** - Gets the correct process identification numbers
- **OpenProcess** - Gets a memory access handle with correct permissions
- **GetModuleBaseAddress** - Gets the correct module's base address
- **FindDMAAddy** - Calculate the final address of multilevel pointers at runtime
- **ReadProcessMemory** - Gets the value contained in a variable
- **WriteProcessMemory** - Overwrites the value contained in a variable

GHB Considerations

This is the #1 most important tutorial we made for the Game Hacking Bible, which is a 70 chapter game hacking course which you are intended to follow step by step. It will answer 99% of the questions you first have when migrating from Cheat Engine to C++ for the first time. It's incredibly important that you spend time learning this tutorial the best you can. You learn a ton of stuff in this video, you will feel extremely confused, but I will walk you through everything.

Do not skip this video - everything you do for the next 6 months builds off of this tutorial.

Questions after following the video?

1. Read this entire post, it includes many frequently asked questions
2. Read all the replies to this post for more information
3. The next 2 videos show you how to improve this code and answers more questions

Video Naming Confusion

The naming convention we used in this batch of videos may confuse you, and it's not your fault. We screwed up.
Our first "How To Hack Any Game #1" video was remade with a new name: How to Find Addresses and Offsets
The current article you're reading was originally named "How to Hack Any Game External C++ Trainer #1"
The next video also improves upon this code and followed our old naming convention.
After that we convert it to an internal and improve upon it in 2 future videos.

I apologize for the confusion. Just follow the GHB chapter by chapter and you should be fine.

How to Hack Any Game C++ Video

Like all of our videos: watching is not enough! You must follow along and do exactly what I do in the video. You should be watching the video on one monitor and writing the code as I write it on your second monitor. If you don't have 2 monitors, git ur bread up homie.

This is our first GHB C++ Coding Tutorial

Visual Studio updates all the time so things might be a bit different when you watch this video. Learn to adapt.

You will make many mistakes while following along with the video, that's normal. If you get completely stuck, download the zip from the attachments and compare it against your code. You can use a program like WinMerge to compare your code and mine to easily find the problems.

Requirements

- Assault Cube 1.2.0.2 - You must use this specific version for all our tutorials
- Cheat Engine - Always use the latest version of Cheat Engine
- Visual Studio Community - Always use the latest version of Visual Studio

https://youtu.be/wiX5LmdD5yk

C++ How To Hack Any Game Source Code

C++:

```
int main()
{
    //Get ProcId of the target process
    DWORD procId = GetProcId(L"ac_client.exe");

    //Getmodulebaseaddress
    uintptr_t moduleBase = GetModuleBaseAddress(procId, L"ac_client.exe");

    //Get Handle to Process
    HANDLE hProcess = 0;
    hProcess = OpenProcess(PROCESS_ALL_ACCESS, NULL, procId);

    //Resolve base address of the pointer chain
    uintptr_t dynamicPtrBaseAddr = moduleBase + 0x10f4f4;

    std::cout << "DynamicPtrBaseAddr = " << "0x" << std::hex << dynamicPtrBaseAddr <<
std::endl;

    //Resolve our ammo pointer chain
    std::vector<unsigned int> ammoOffsets = { 0x374, 0x14, 0x0 };
    uintptr_t ammoAddr = FindDMAAddy(hProcess, dynamicPtrBaseAddr, ammoOffsets);

    std::cout << "ammoAddr = " << "0x" << std::hex << ammoAddr << std::endl;

    //Read Ammo value
    int ammoValue = 0;

    ReadProcessMemory(hProcess, (BYTE*)ammoAddr, &ammoValue, sizeof(ammoValue), nullptr);
    std::cout << "Curent ammo = " << std::dec << ammoValue << std::endl;

    //Write to it
    int newAmmo = 1337;
    WriteProcessMemory(hProcess, (BYTE*)ammoAddr, &newAmmo, sizeof(newAmmo), nullptr);
```

```
    //Read out again
    ReadProcessMemory(hProcess, (BYTE*)ammoAddr, &ammoValue, sizeof(ammoValue), nullptr);

    std::cout << "New ammo = " << std::dec << ammoValue << std::endl;

    getchar();

    return 0;
}
```

C++ How to Hack Any Game Frequently Asked Questions

After 9 years of making tutorials and answering questions, we already know exactly what you're going to ask.

Why doesn't my C++ hack work?

You must use Assault Cube Version 1.2.0.2
You must run your hack and Visual Studio as Admin
If your game is x86, you must compile your hack for x86.
If your game is x64, you must compile your hack for x64.

You probably just have a couple small typos, watch the end of the video where I run the debugger and step through the code and replicate that process to find your bugs. This a mandatory skill which you will do for hundreds of hours in your game hacking career.

If you're still having trouble, download the working code from the attachments and use WinMerge to compare the files and find the typos.

What's the multi level pointer template you used?

Code:

```
Address = Value = ?

base ptr -> address + offset4 = address

base ptr -> address + offset3 = address

base ptr -> address + offset2 = address

static base -> address + offset1 = address
```

Where's the cheat table you used in the video?

We make that cheat table in the previous video: How To Hack Any Game - Cheat Engine
But we teach you how to find the Current Weapon Ammo pointer in the current video

What is the CurrentAmmoPointer so I can paste it into Cheat Engine?

Base Address: ac_client.exe+0x10F4F4
Offsets: 0x374, 0x14, 0x0
Screenshot:

How did we know which address was the right one so fast?

Because I'm an expert game hacker and I used my intuition. You on the other hand, have to use trial and error and suffer for hours. There is no avoiding trial and error in game hacking, get used to it.

How did we learn to make the GetModuleAddress function?

Microsoft teaches you how to do it: CreateToolhelp32Snapshot & Taking a Snapshot

Why don't we teach GetWindowThreadProcessId to get the process id?

Because this function has notoriously been the cause of thousands of noob problems.
This function will cause you grief when multiple windows have the same title or when the title changes (such as a browser).
If you use our ToolHelp32 method, you will never have any of these problems.

What is 0x10f4f4?

This is the relative offset used to get the base address of the multilevel pointer.

```
uintptr_t dynamicPtrBaseAddr = moduleBase + 0x10f4f4;
```

0x109B74 vs 0x10f4f4

0x109B74 is the relative offset to the local player pointer which only works in single player
0x10f4f4 is the relative offset to the local player pointer which works in single player and multiplayer

0x10f4f8, the next variable after 0x10f4f4 is the entitylist pointer so we're clearly using the correct game logic when using these two variables in unison in our hacks. You should always use 0x10f4f4**.**

0x509B74 vs 0x50f4f4

These are the actual addresses of the 2 different pointers above, after adding the relative offsets to the module base address (0x4000000)

Precompiled Header: stdafx.h vs pch.h

Visual Studio keeps changing the default name of the precompiled headers. It doesn't matter which you use as long as you're consistent in your project. Use whatever Visual Studio generates. You can always change it in the project settings if you want.

Why use Precompiled Headers?

It makes compilation of large projects much faster, which is very helpful when you're debugging and solving errors in your code. Sometimes you will be hitting the compile button 5 times per minute and saving 5 seconds per compile is huge.

Confused by the windows API?

Everyone is during this tutorial. Just do the GHB and you will learn 1 piece of it at a time. Make sure you read What is the Windows API?

The ammo value changed but it's not frozen?

To "freeze" the value, you have to overwrite the value inside a while loop that runs indefinitely. Our easy to learn PoC doesn't do that, it just overwrites the variable once. We'll teach you some tricks for this in the next 2 videos.

Why doesn't this work on all guns?

The CurrentWeaponAmmo pointer points to the current weapon's ammo, we only parse that once. The pointer changes when you change guns. To solve this, you would call FindDMAAddy before each call to WriteProcesMemory to get the updated address.

What's up with _wcsicmp?

```
if (!_wcsicmp(procEntry.szExeFile, procName))
```

_wcsicmp = wide character string insensitive compare

This means it compares two strings where each character is a wide char (2 byte char instead of a regular 1 byte char), wide characters strings can accommodate UNICODE which requires 2 bytes per char. We use Unicode in almost all our tutorials because it's the industry standard now.

Insensitive means it's case insensitive, it ignores the capitalization, which is very helpful. If you forget to capitalize something, it still works, so you don't have to hunt down rogue bugs because you forgot to capitalize a letter in the process name.

_wcsicmp is one of many string comparison functions, they're basically all the same with slight variations. The variations are in the type of strings they accept as arguments, you just use the one that works on the string types you already are using. This might be super confusing now but it's really simple once you have more experience.

Often people will do multiple string conversions to get the correct type to work with a function they want to use. If you're converting more than 1 string before a compare, chances are you're just using the wrong function.

For example if you have:

C++:

```
char* string1;
wchar_t* string2;
```

in this example, do not convert them both to std::strings to use the standard template's string compare.

Just convert the char* to wchar_t and then use regular _wcsicmp. You should only need 1 conversion.

If you're doing some insane shit like this, please kill yourself

C++:

```
std::string string1 = "FiveM_GTAProcess.exe";
std::wstring string2 = std::wstring(string1.begin(), string1.end());
LPCWSTR string3 = string2.c_str();
_bstr_t string4(string3);
std::string string5(string4);
const char* string6 = string5.c_str();

if (!strcmp(PE32.szExeFile, string6))
{
```

_wcsicmp return value checking with the not operator (!) ??

```
if (!_wcsicmp(procEntry.szExeFile, procName))
```

using the not operator (!) before a function call, is essentially checking if the return value of the function is zero

When the return value of this function is zero, it means the strings are identical and we found the correct process:

What is a trainer?

The word trainer typically means it's a simple external cheat.

Internal vs. External?

External
External Hacks use WriteProcessMemory and ReadProcessMemory to retrieve variables and overwrite them. RPM/WPM is slow because you have the overhead of the API calls. Just wanna do something simple? Use an external.

Internal
Internal hacks are created by injecting DLLs into the game process, when you do this you have direct access to the process's memory which means fast performance and simplicity. Making a real hack? Go internal.

The one caveat is this: pasters who can't bypass anticheat will use externals combined with kernel drivers to easily bypass anticheats because bypassing the detection of an injected DLL is too difficult for them.

Homework

Make sure you can get the C++ How To Hack Any Game external trainer code to compile and work as it does in the video. You will probably want to do this video twice if you're confused because we drop so much knowledge on you in one video.

GHB Navigation

Prev: GHB114-2 Visual Studio Run as Administrator + Manifest File
Curr: GHB115 - How to Hack Any Game - First External C++ Trainer
Next: GHB116 - How to Hack Any Game Tutorial C++ Trainer #2 - External v2

Video Tutorial How to Hack Any Game Tutorial C++ Trainer #2 - External v2

C++ External Trainer 2 - Making Improvements

In our previous External C++ Trainer Tutorial, we took our addresses, pointers & offsets from our Cheat Engine Table and we built a basic C++ hack with them. In this tutorial we will add some important feature improvements and new functionality.

This is our last beginner external tutorial for GHB1, in the next video we will teach you how to convert this into an internal DLL. It's important to master these basic external trainer skills now, but don't worry you will get more experience in GHB3 when we teach you how to make external CSGO cheats.

The previous video was just a simple PoC, this video turns it into a more serious hack. We previously made a Process Class and in this video we start working on our Memory Class to automate common memory manipulations.

What you will learn

These fundamental C++ trainer features and concepts will create the basis of everything you do in the future.

- Precompiled Headers
- Project Organization
- Proper Hack loop
- Setting up Hotkeys using GetAsyncKeyState
- Patching instructions and variables
- nopping instructions
- continuous write aka freezing
- how to detect application status
- memory library basics
- **PatchEx Function** - Patch bytes in an external process
- **NopEx Function** - Overwrite assembly instructions with the NOP instruction

Hacks we will create or modify

- health hack - continuous over-writing like Cheat Engine freeze
- unlimited ammo - nopping
- no recoil hack - NOPing the call (new)

GHB Considerations

This is a chapter in our Game Hacking Bible, if you're not following that course step by step, you should be doing so. This video will answer many questions you probably had from the previous 2 videos.

Requirements

- Assault Cube 1.2.0.2 - You must use this specific version for all our tutorials
- Cheat Engine - Always use the latest version of Cheat Engine
- Visual Studio Community - Always use the latest version of Visual Studio
- The Cheat Table from the attachments below

How to Do this Tutorial

Follow the video and replicate everything I do as you watch, you should be updating your Visual Studio project as I do it.

Video Order Confusion

Between the first External Trainer tutorial and this one, we posted this tutorial: Reverse Engineering Tutorial - Assault Cube Recoil
The video you're currently looking at requires the Recoil address found in that tutorial.
The way we organized the GHB chapters, this video was put with the reverse engineering content in GHB2, which is confusing.
Sorry about that, we will try to organize our content better when we have time.

Required Cheat Table with Recoil Pointer

You need this Cheat Table to complete this video tutorial.
Download it from the attachments in this tutorial: Reverse Engineering Tutorial - Assault Cube Recoil

External C++ Trainer Tutorial #2

https://youtu.be/UMt1daXknes

MSDN pages referenced in the video:

- GetExitCodeProcess
- GetAsyncKeyState
- VirtualProtectEx

C++ External Trainer #2 Source Code

This should give you an idea of what you learn in this tutorial and what improvements we made to the original source code.

C++:

```
int main()
{
    HANDLE hProcess = 0;

    uintptr_t moduleBase = 0, localPlayerPtr = 0, healthAddr = 0;
    bool bHealth = false, bAmmo = false, bRecoil = false;

    const int newValue = 1337;

    //Get ProcId of the target process
    DWORD procId = GetProcId(L"ac_client.exe");

    if (procId)
    {
        //Get Handle to Process
        hProcess = OpenProcess(PROCESS_ALL_ACCESS, NULL, procId);

        //Getmodulebaseaddress
        moduleBase = GetModuleBaseAddress(procId, L"ac_client.exe");
```

```
        //Resolve address
        localPlayerPtr = moduleBase + 0x10f4f4;

        //Resolve base address of the pointer chain
        healthAddr = FindDMAAddy(hProcess, localPlayerPtr, { 0xF8 });
    }
    else
    {
        std::cout << "Process not found, press enter to exit\n";
        getchar();
        return 0;
    }

    DWORD dwExit = 0;
    while (GetExitCodeProcess(hProcess, &dwExit) && dwExit == STILL_ACTIVE)
    {
        //Health continuous write
        if (GetAsyncKeyState(VK_NUMPAD1) & 1)
        {
            bHealth = !bHealth;
        }

        //unlimited ammo patch
        if (GetAsyncKeyState(VK_NUMPAD2) & 1)
        {
            bAmmo = !bAmmo;

            if (bAmmo)
            {
                //FF 06 = inc [esi]
                mem::PatchEx((BYTE*)(moduleBase + 0x637e9), (BYTE*)"\xFF\x06", 2,
hProcess);
            }

            else
            {
                //FF 0E = dec [esi]
                mem::PatchEx((BYTE*)(moduleBase + 0x637e9), (BYTE*)"\xFF\x0E", 2,
hProcess);
            }
        }

        //no recoil NOP
        if (GetAsyncKeyState(VK_NUMPAD3) & 1)
        {
            bRecoil = !bRecoil;

            if (bRecoil)
            {
                mem::NopEx((BYTE*)(moduleBase + 0x63786), 10, hProcess);
            }

            else
            {
                //50 8D 4C 24 1C 51 8B CE FF D2; the original stack setup and call
                mem::PatchEx((BYTE*)(moduleBase + 0x63786),
(BYTE*)"\x50\x8D\x4C\x24\x1C\x51\x8B\xCE\xFF\xD2", 10, hProcess);
            }
        }

        if (GetAsyncKeyState(VK_INSERT) & 1)
        {
            return 0;
```

```
        }

        //Continuous write
        if (bHealth)
        {
            mem::PatchEx((BYTE*)healthAddr, (BYTE*)&newValue, sizeof(newValue),
hProcess);
        }

        Sleep(10);
    }

    std::cout << "Process not found, press enter to exit\n";
    getchar();
    return 0;
}
```

Upon completing this C++ trainer tutorial you will have a fundamental understanding of some of basic things you will be doing in every hack you make.

FindDMAAddy With Single Offset

In this tutorial we show you this code which in hindsight, I shouldn't have done. It teaches a bad habit.

This is a single level pointer. FindDMAAddy works fine on single level pointers with 1 offset, but it's kind of overkill. Keep in mind that calling ReadProcessMemory inside the FindDMAAddy function can be inefficient. If you only need the address of 1 or two variables in a class, this is fine just go ahead and use the function. But if you need more than 2, you should just get the address of the object once per execution, and then add the offsets manually to get each variable in the class which you need. There is no reason to call FindDMAAddy 8 times to get each variable in the class, this is inefficient.

The better solution would be:

C++:

```
//Resolve base address
uintptr_t localPlayerPtr = moduleBase + 0x10f4f4;


uintptr_t = localPlayerAddr = FindDMAAddy(hProcess, localPlayerPtr, { 0x0 });

uintptr_t healthAddr = localPlayerAddr + 0xF8;
uintptr_t armorAdddr = localPlayerAddr + 0xFC;
//etc...
```

See? No redundant calls to ReadProcessMemory, just keep in mind you have to do this once per execution to stay updated.

Frequently Asked Questions

Where is the Solaire tutorial mentioned in the video?

The Solaire video was replaced by this much better tutorial: [How to Hack Any Game With Cheat Engine]

Precompiled Header: stdafx.h vs pch.h

Visual Studio keeps changing the default name of the precompiled headers. It doesn't matter which you use as long as you're consistent in your project. Use whatever Visual Studio generates. You can always change it in the project settings if you want.

Why use Precompiled Headers?

It makes compilation of large projects much faster, which is very helpful when you're debugging and solving errors in your code. Sometimes you will be hitting the compile button 5 times per minute and saving 5 seconds per compile is huge.

Homework

Make sure you can get the code to compile and work as it does in the video

GHB Navigation

Prev: GHB115 - How to Hack Any Game - First External C++ Trainer
Curr: GHB116 - How to Hack Any Game Tutorial C++ Trainer #2 - External v2
Next: GHB117 - How to Hack Any Game Tutorial C++ Trainer #3 - First Internal

Video Tutorial How to Hack Any Game Tutorial C++ Trainer #3 - First Internal

How to Make An Internal Hack Tutorial

In this video you will learn how to convert our external hack from the previous tutorial into an internal DLL hack. We will show you the internal equivalent of all the functions we've used before and incorporate them into our memory library. By then end of this video you will start to understand the true power of an internal hack.

What is an internal hack?

Internal hacks are DLLs which contain all your hack code, which is injected into the game process. By injecting a DLL (Dynamic Link Library) into the game, you get direct access to the process's memory which results in much faster performance and simplicity. Normally a DLL is used as a self contained library, providing an API you can use in separate projects via exported functions. Normally you would access these functions only when needed, but in our case, we will use a DLL to get our code to run in the game's own memory. Injected DLLs can be made more sneaky by using different injection methods such as Manual Mapping, which you will learn much later in the GHB.

Why are internal hacks so cool?

This is how easy it is to manipulate memory inside a DLL

C++:

```
int* ammo = (int*)0xDEADC0DE;
*ammo = 1337;
```

Much nicer than all that ReadProcessMemory and WriteProcessMemory garbage right? This is a simple example, we'll show you the different ways you can do it in the next couple videos.

What you'll learn in this tutorial

- a basic DLL hack template
- a main internal hack thread
- Freeze values like Cheat Engine internally
- Ejecting the DLL so you don't have to restart the game during debugging
- Using pointer to validate game's hackable state
- Typecasting addresses to pointers to modify them
- **DllMain()** - handler for DLL
- **CreateThread()** - starts our main function outside of DllMain
- **AllocConsole()** - Spawns a text console for visual output
- **GetModuleHandle()** - internal ModuleBaseAddress Function
- **Nop()** - Internal instruction nopping function
- **Patch()** - Internal code patching function
- Internal FindDMAAddy function

GHB Considerations

This is a chapter in our Game Hacking Bible, make sure you're following our tutorial series step by step. This video is the basis of all future internal hack tutorials, so make sure you understand these basics. In the future we will convert this to use ReClass generated classes which will truly open your eyes to the power of an internal hack.

Requirements

- Assault Cube 1.2.0.2 - You must use this specific version for all our tutorials

- Cheat Engine - Always use the latest version of Cheat Engine
- Visual Studio Community - Always use the latest version of Visual Studio Community
- ReClass.NET - Always use GH's 100% up to date perfect version
- DLL Injector - Download the Official GH Injector
- Source Code we made in the previous video, either do that video first or download the attachment

First C++ Internal Hack Tutorial

https://youtu.be/hlioPJ_uB7M

Video Tutorial References

- Create C/C++ DLLs in Visual Studio
- Walkthrough: Create and use your own Dynamic Link Library
- AllocConsole function - Windows Console
- FreeLibraryAndExitThread function
- GetModuleHandleA function
- DllMain entry point - Windows applications
- Console Handles - Windows Console
- freopen, freopen_s - cppreference.com
- std::fclose - cppreference.com

Sample code from dllmain.cpp

C++:

```
#include "stdafx.h"
#include <iostream>
#include "mem.h"

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

    std::cout << "OG for a fee, stay sippin' fam\n";

    uintptr_t moduleBase = (uintptr_t)GetModuleHandle(L"ac_client.exe");

    //calling it with NULL also gives you the address of the .exe module
    moduleBase = (uintptr_t)GetModuleHandle(NULL);

    bool bHealth = false, bAmmo = false, bRecoil = false;

    while (true)
    {
        if (GetAsyncKeyState(VK_END) & 1)
        {
            break;
        }

        if (GetAsyncKeyState(VK_NUMPAD1) & 1)
            bHealth = !bHealth;

        if (GetAsyncKeyState(VK_NUMPAD2) & 1)
        {
            bAmmo = !bAmmo;
```

```
        }

        //no recoil NOP
        if (GetAsyncKeyState(VK_NUMPAD3) & 1)
        {
            bRecoil = !bRecoil;

            if (bRecoil)
            {
                mem::Nop((BYTE*)(moduleBase + 0x63786), 10);
            }

            else
            {
                //50 8D 4C 24 1C 51 8B CE FF D2 the original stack setup and call
                mem::Patch((BYTE*)(moduleBase + 0x63786),
(BYTE*)"\x50\x8D\x4C\x24\x1C\x51\x8B\xCE\xFF\xD2", 10);
            }
        }

        //need to use uintptr_t for pointer arithmetic later
        uintptr_t* localPlayerPtr = (uintptr_t*)(moduleBase + 0x10F4F4);

        //continuous writes / freeze

        if (localPlayerPtr)
        {
            if (bHealth)
            {

                //*localPlayerPtr = derference the pointer, to get the localPlayerAddr
                // add 0xF8 to get health address
                //cast to an int pointer, this pointer now points to the health address
                //derference it and assign the value 1337 to the health variable it
points to
                *(int*)(*localPlayerPtr + 0xF8) = 1337;
            }

            if (bAmmo)
            {
                //We aren't external now, we can now efficiently calculate all pointers
dynamically
                //before we only resolved pointers when needed for efficiency reasons
                //we are executing internally, we can calculate everything when needed
                uintptr_t ammoAddr = mem::FindDMAAddy(moduleBase + 0x10F4F4, { 0x374,
0x14, 0x0 });
                int* ammo = (int*)ammoAddr;
                *ammo = 1337;

                //or just
                *(int*)mem::FindDMAAddy(moduleBase + 0x10F4F4, { 0x374, 0x14, 0x0 }) =
1337;
            }

        }
        Sleep(5);
    }

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

```
BOOL APIENTRY DllMain(HMODULE hModule,
    DWORD  ul_reason_for_call,
    LPVOID lpReserved
)
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
        CloseHandle(CreateThread(nullptr, 0, (LPTHREAD_START_ROUTINE)HackThread, hModule,
0, nullptr));
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```

Frequently Asked Questions

What does GetModuleHandle do?

It's a Windows API macro function that gives you a handle to the module you define by name. Windows implements module handles as the address of the module itself. Internal hacks are so simple right?

What are the pros and cons of Internal and External Hacks?

Internal hacks are more efficient, besides that learn more here: [Internal vs External Hacks]

How do we make a C++ DLL injector?

This is taught in the next couple videos, for now just use the GH Injector.

Can you explain this code?

```
*(int*)(*localPlayerPtr + 0xF8) = 1337;
```

Break it down into parts

`*localPlayerPtr` - this is getting the value pointed to by the pointer, which is the address of our local player
`+ 0xF8` - this is adding the health offset to our local player address, resulting in the address of our health offset
`(int*)` - this is casting our health address to a pointer of the type: integer
`*` - this operator on the far left is de-referencing the health pointer
`= 1337;` - "The value pointed to by the pointer is assigned the value 1337"

DllMain Loader Lock

Loader lock is when you something goes wrong and DllMain never returns, making new calls to DllMain have problems, basically freezing the process. Despite hundreds of websites telling you not to call `CreateThread()` inside DLL_PROCESS_ATTACH, that's exactly what you should be doing, as we show in the video. Trust me, I've been doing this for 9 years.

The video will give you a sort of example of what Loader Lock looks like when we inject with the debugger attached. This is what people tell you will happen when you use CreateRemoteThread() in `DLL_PROCESS_ATTACH`, but thousands of actively distributed piece of software with millions of downloads do this and never have any problems (including Microsoft's own software).

DLL Ejection?

If you don't eject your DLL, you won't be able to load it again. In the video we show you how to setup a hotkey that results in the ejection of the DLL using, but you have to free your console first. You can read more here: How to Eject a DLL

Program Crashing? Access Violation?

In an external, calling ReadProcessMemory on an incorrect memory address won't crash the program. But in an internal, if you try to directly access the wrong memory, you will get an access violation causing the program to crash. Typically this is because your addresses or offsets are wrong, so always check those first. But your pointer processing logic could also be the problem.

The second best way to cause crash is to overwrite the wrong address and end up overwriting code instructions. When these malformed instructions execute, you will crash.

The third most common cause is dangling pointers. A dangling pointer is one that is not assigned to a valid or correct memory address. Perhaps it pointed to the correct thing 1 second ago and now it doesn't, on the next loop of your hack you will get an access violation or other crash.

So how you do solve these memory violations?

This simplest and best way is to find a variable that can define the "state" of the program. Sometimes these exist in the game logic but they can be hard to find. More often then not you will find a pointer to something like the local player, which is valid while playing but invalid during cutscenes, the menu & the lobby. We can use this to check the "hackable state" of the game.

You can use this as a state variable, just by testing if it's NULL or not to check if the game is in "hackable state". This sounds so simple but it really does work, every hack I've ever made I simply use the local player pointer for this "status" check at the beginning of the hack loop. This works on Assault Cube, CSGO, Sauerbraten and all the other games we use in our tutorials.

The second best way is to do everything inside of a hook. Inside a hook, you're executing inside the game's own thread inside a proper game function. In this way, you have the best chance of everything being valid.

In an internal you can also revert to using ReadProcessMemory() for variables which are problematic, which helps avoid access violations, but it will slow your code down a little bit.

We don't teach a lot of error checking in the GHB because we don't want to over-complicate the code, it's already hard enough to teach, as it is. As you further your skills you can start adding your own error checking.

Still having Problems?

Read the comments and if that doesn't help, turn off your computer, pray to Jesus and then lay in bed pissed off about it for 8 hours and try again tomorrow.

Homework

The thing that most people get confused with is how we cast and de-reference the pointers which we use to modify the game variables. Use the Visual Studio debugger and step through the code while you compare what you see in ReClass & Cheat Engine. Add and subtract the de-reference operator (*), and debug it again and watch how it performs differently. You may need to do this 500 times before you really understand what's going on. For practice, try adding some more features to this hack, like infinite armor etc...

You will run into some issues learning the ins and outs of internals, take your time.

GHB Navigation

Prev: GHB116 - How to Hack Any Game C++ External Trainer #2
Curr: GHB117 - How to Hack Any Game - First Internal Hack
Next: GHB118 - Simple C++ DLL Injector Tutorial

Video Tutorial Simple DLL Injector Source Code

DLL Injector Tutorial Walkthrough

This Simple DLL Injector source code tutorial uses the functions `VirtualAllocEx`, `CreateRemoteThread` & `WriteProcessMemory`. DLL injection is a method used for executing foreign code within the address space of a separate process by forcing the process to load a DLL. Foreign programs utilize DLL injection to alter the operation of another program with set instructions that reassign objectives of certain components of the process.

What will you learn?

- The Basics of DLL Injection
- How to make a simple DLL Injector
- VirtualAllocEx, CreateRemoteThread, WriteProcessMemory
- Visual Studio Debugging

https://youtu.be/PZLhlWUmMs0

GHB Considerations

In this Simple C++ DLL Injector Source Code Tutorial you will learn about the fundamental concepts in DLL injection. You will need this to inject the first internal hack you make in the GHB.

Video Walkthrough

First, create a new project and select "Console Application". For the purpose of this tutorial, let's name it "Simple DLL Injector Tutorial".

Setting Project Properties

Once your project has loaded up, navigate to the project properties. We'll need to set a few things:

1. **Linker**: Set a manifest file and set the User Account Control (UAC) execution level to 'require administrator'. This precaution will prevent someone from running your code without administrative privileges, avoiding potential issues.
2. **Character Code Set**: We'll be using 'multi-byte character set' for this tutorial.

Getting the Process ID

To get started with the code, you'll need to include a few headers. For this example, we'll include Windows.h and TlHelp32.h.

Now let's write our GetProcID function, which will accept a pointer to our process name. This function is designed to obtain the process ID by iterating through a snapshot of all currently loaded processes.

C++:

```
#include <iostream>
#include <Windows.h>
#include <TlHelp32.h>
DWORD GetProcId(const char* procName)
{
    DWORD procId = 0;
    HANDLE hSnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);

    if (hSnap != INVALID_HANDLE_VALUE)
```

```
    {
        PROCESSENTRY32 procEntry;
        procEntry.dwSize = sizeof(procEntry);

        if (Process32First(hSnap, &procEntry))
        {
            do
            {
                if (!_stricmp(procEntry.szExeFile, procName))
                {
                    procId = procEntry.th32ProcessID;
                    break;
                }
            } while (Process32Next(hSnap, &procEntry));
        }
    }
    CloseHandle(hSnap);
    return procId;
}
```

Working with Main Function

In our main function, let's define the DLL path and the process name. We'll then call our GetProcID function to fetch the process ID, followed by calling the OpenProcess function.

Creating Thread and Closing Handles

The next step involves creating a remote thread in the target process using CreateRemoteThread. The thread will call LoadLibraryA and pass the location where we stored the DLL path. After that, close the thread and process handles.

C++:

```
int main()
{
    const char* dllPath = "C:\\Users\\me\\Desktop\\dll.dll";
    const char* procName = "csgo.exe";
    DWORD procId = 0;

    while (!procId)
    {
        procId = GetProcId(procName);
        Sleep(30);
    }

    HANDLE hProc = OpenProcess(PROCESS_ALL_ACCESS, 0, procId);

    if (hProc && hProc != INVALID_HANDLE_VALUE)
    {
        void* loc = VirtualAllocEx(hProc, 0, MAX_PATH, MEM_COMMIT | MEM_RESERVE,
PAGE_READWRITE);

        WriteProcessMemory(hProc, loc, dllPath, strlen(dllPath) + 1, 0);

        HANDLE hThread = CreateRemoteThread(hProc, 0, 0,
(LPTHREAD_START_ROUTINE)LoadLibraryA, loc, 0, 0);

        if (hThread)
        {
            CloseHandle(hThread);
        }
    }
```

```
    if (hProc)
    {
        CloseHandle(hProc);
    }
    return 0;
}
```

Reviewing the DLL Injector Code

The DLL injector code we've created essentially gets the process ID of the target process, opens that process, allocates memory within it, and writes the DLL path to that memory. A new thread is then created in the target process, which calls the LoadLibraryA function using the memory location of our DLL path. This maps the DLL into the process memory and executes it. At the end, the function closes all handles and exits.

Running the Debugger and Injecting

To test our DLL injector, we'll run our application with administrative privileges and inject our DLL into a test process (in this case, a game). If everything is working correctly, you'll see your DLL working within the target process.

Potential Problem - OpenProcess Requires Admin Permissions

Most likely, you had an issue with `WriteProcessMemory` because your process handle is NULL due to OpenProcess failing. As we said in the video, this can be fixed by changing the UAC settings to require administrative privileges to execute. This ensures our program will initialize with administrative privileges and can remove some issues.

If you encountered another issue, rewatch the video and compare your code with the visual studio source attachment we've provided by using a tool like WinMerge.

Conclusion

This DLL Injection tutorial has provided a comprehensive guide on how to build a simple DLL injector using C++. By meticulously working through each step, you should now have a clear understanding of how to construct a DLL injector, from setting up the project in a C++ environment to creating threads and allocating memory. The hands-on approach offered in this tutorial, paired with the provided DLL injector source code, has hopefully given you a strong foundation to further your skills in this field.

Remember, the LoadLibrary injector method utilized here is one of the most straightforward techniques in DLL injection and serves as an excellent starting point for beginners. As you grow more comfortable, you are encouraged to explore more complex DLL injection techniques. We hope that this DLL injection tutorial has been insightful and has effectively demonstrated the process of creating a DLL injector.

GHB Navigation

Prev: GHB 117 - How to Hack Any Game Tutorial C++ Trainer #3 - First Internal
Curr: GHB 118 - Simple C++ DLL Injector Source Code
Next: GHB 119 - How to Debug Your Hack or Aimbot with Visual Studio

More DLL Injector Source Codes

- DLL Injection without WriteProcessMemory
- Extreme Injector v3.6.1 Download
- How to embed DLL in your Injector loader
- Windows C++ Shellcode Injection Tutorial
- C# DLL Injector Tutorial
- C# Manual Map Injector with GUI
- C# DLL Loader / Embedded Injector

Video Tutorial How to Debug Your Hack with Visual Studio Debugger

Visual Studio Debugger Tutorial

In this tutorial, we'll learn how to debug an external and internal project using Visual Studio Debugger. We'll also cover some general tips and tricks that can help you identify and fix errors in your code.

Every problem you have can be solved by using the debugger.
I'm not the best coder so I spend a lot of time in the debugger, but this is a necessary skill for all coders.
This tutorial is very basic but has some tips and tricks that you may not be aware of.
If you're using C# then you can use Marshal.GetLastWin32Error instead of GetLastError()

Before you ask for help, always debug!

How To Debug Your Cheat with Visual Studio Debugger

This video will showcase the Visual Studio debugging process on assault cube aimbot, however, this tutorial is applicable to almost any project because we are teaching you the fundamentals of general debugging.

What will you learn?

- How to debug cheats
- How to use the Visual Studio debugger
- How low-level computer architecture works
- How to debug internal and external projects
- General Tips and Tricks about debugging

https://youtu.be/IVRS_P58Q2I

GHB Considerations

Learning how to debug is an important part of game hacking because you will need these skills to determine why your hack or aimbot is not functioning correctly. Sometimes, the time it takes to debug a new hack can take more time than it took to program it. We want you to learn the absolute standards of debugging before proceeding with future tutorials due to the increased frequency of running into problems with your code

Video Walkthrough

To start, make sure you're compiling your project in debug mode, as this will provide the debugger with more information about the executable and avoid any optimizations that might make it difficult to step through the code.

Starting the Debugger

When debugging an executable file, all you have to do is press the play button. This will fire up the executable and display any relevant information about the game.

Setting Breakpoints

To investigate your code, you'll want to set breakpoints at specific points. In this example, we'll set a breakpoint at the beginning of the main function.

Debugger Controls

The debugger offers several controls:

- **Continue**: Resumes the execution of the process until the next breakpoint is hit.
- **Stop**: Stops the debugger.
- **Restart**: Restarts the debugger.
- **Step Into**: Steps into the current function or statement.
- **Step Over**: Steps over the current function or statement.
- **Step Out**: Steps out of the current function.

Stepping Through the Code

When you hit a breakpoint, you can use the debugger controls to step through your code and investigate the state of your program.

In this example, we'll step through the Set Window function and observe how the program interacts with the Windows API. We'll also step into the print function and then use the "Step Out" control to return to the main loop.

Investigating Variables and Errors

The Autos window in Visual Studio displays variables that are being acted on by the current function. This is useful for checking the state of your variables as you step through your code.

You can also use the Watch window to add specific variables or expressions that you want to keep an eye on, such as the GetLastError function, which can help you diagnose issues.

Debugging Tips

- Be mindful of strings and character sets, as these can often cause issues when working with Windows API functions.
- If you encounter an error, simply Google the error code to find more information about the issue.

Debugging with Visual Studio 2015

Visual Studio 2015 has introduced some interesting new features, such as allowing you to change the code while debugging, then stepping over the modified code, and the debugger will automatically recompile it. This feature wasn't available in previous versions, making it a great addition to Visual Studio 2015.

Stepping Out of Functions

If you don't care about a specific function, you can simply step out of it using the "Step Out" option in the debugger. This allows you to move on to the next line after the function call, saving time and effort.

Working with Addresses and Breakpoints

When working with addresses, it's important to ensure everything is set up correctly. You can set breakpoints at specific lines of code and use the "Continue" option to proceed with the debugging process. To check if your code is working properly, you can use tools like Cheat Engine to inspect memory addresses.

Stepping Through Code and Identifying Errors

To ensure your code is working as expected, you can step through each line of code one by one. If you encounter a line of code that doesn't do what it's supposed to, you can simply edit it and continue debugging. Most of the time, errors are due to simple typos or incorrect syntax.

Attaching Debugger to a Process

When working with an internal project, you can attach the debugger to the process by clicking "Debug" and then "Attach to Process." In Visual Studio 2015, you can use the new breakpoint window to manage your breakpoints effectively.

Automating Process Attachment

To save time, you can automate the process attachment by right-clicking on the project, going to "Properties," then "Debugging," and setting the "Attach" option to "Yes." This will automatically attach the debugger to the correct process when you click the "Play" button.

Debugging and Modifying Variables

While debugging, you can modify variables using the "Autos" or "Watch" windows. You can change the display format of variables by adding a comma followed by "H" for hexadecimal or "D" for decimal. You can also typecast variables directly in the watch window, allowing you to inspect complex data structures easily.

Finding Player Arrays and Troubleshooting

The Visual Studio debugger can be an invaluable tool for finding player arrays and troubleshooting game-related issues. By stepping through the code and checking elements in arrays, you can identify issues and fix them accordingly.

What are some examples of common errors?

- Syntax error
- Runtime error
- Logistical error
- Calling an invalid function
- Using an incorrect variable name in the wrong section

Debugging Methods

- Static analysis - analyzing the files on disk
- Print debugging - monitoring execution by printing out debug messages
- Remote debugging - used for debugging devices on a PC
- Post-mortem debugging - debugging by reading logs

Having trouble?

Debugging can be a difficult and frustrating procedure and can even require more work than writing the actual code of your program. This tutorial is meant to demonstrate debugging as a whole and some of your issues may not be covered in this article. If you are having trouble and cannot fix your error whatsoever, it is recommended to dismiss it and move on in future tutorials because your issue could be a complicated process that you won't be able to fully comprehend due to your novice entrance into debugging. If you'd like to get additional insight, check out timb3r's debugging guide and the MSDN resources below.

- Learn to debug C++ code using Visual Studio
- Full Debugger Documentation
- How to debug effectively

Homework

Your homework is to set up a debugging environment to find the solution to a problem with your source code using the Visual Studio Debugger without referencing the tutorial.

In conclusion, the Visual Studio debugger is a powerful and versatile tool that can significantly improve your development process. By following this step-by-step walkthrough, you'll be well-equipped to tackle any debugging challenges you may encounter.

Prev: GHB 118 - Simple C++ DLL Injector Source Code
Curr: GHB 119 - How to Debug Your Hack or Aimbot with Visual Studio Debugger
Next: GHB 120 - Game Hacking with Reclass Tutorial Video 2

Video Tutorial C++ Detour / Hooking Function Tutorial

GHB Considerations

This is a very difficult tutorial for noobs. It really doesn't belong in this part of the GHB, but it's here because this is one of the #1 questions we get from noobs. Everyone wants to turn their Cheat Table Injection Scripts into C++ cheats. You can only do that with a detour, that's what the Cheat Engine injection template does in the background. So that's why we're showing you this early. It may be very difficult for you to understand, watch it, try it out and if you get stuck, just move on. You can come back to this later when you get more experience.

What is a detour?

The word detour describes the act of changing the assembly instructions to jump to a different location, essentially re-directing the flow of execution. Typically you are doing this to detour the code into a memory region where your own code exists. Thus you're forcing the game to execute your code.

The key to doing this is to make sure you do not corrupt the stack, you execute the bytes that you overwrote and you jump back to the proper position after your code executes.

Mid Function Hooking

A detour is sometimes referred to as a mid function hook, the only caveat being that if you detour the first byte of a function, then this is not a mid function hook. Placing your detour at the first byte of a function is easily detectable by anti-cheats, which is why a mid function hook is less detectable. Anti-cheats can easily check the first byte, but it's not always likely that they check every byte in the function.

External Detouring / Hooking

Typically you will do this by using WriteProcessMemory to write your code into the target process, then you use WPM in the same way to change the existing code to jump into yours. External hooking / detouring is more complicated because you must write shellcode into the target process. Internally you can do it quite easy. But externally you have to turn your assembly into bytes and write it to the process as shellcode. When you're internal it's much easier and you don't need to play with assembly as much.

Stolen Bytes

The bytes you overwrite when writing your detour to memory are referred to as stolen bytes, it is important that you know how many bytes you overwrote and it's you must execute these after you perform your detour or else the stack/registers will be corrupted. The number of bytes will not always be the same, let's say you're overwriting an instruction which is 8 bytes, but your detour you're writing is only 5 bytes. In this case, you want to copy all 8 bytes for the stolen bytes. You can't execute 75% of an instruction.

You must copy the stolen bytes and execute them after your own code executes to ensure you don't crash the program.

Code Caves

A code cave is a portion of the target processes memory which is not used by the process. Typically we are referring to memory which is allocated but contains nothing that the process requires to execute. When using a detour, you can write your code into this memory without having to allocate memory. By not having to allocate memory you are being less "noisy". If you're using a codecave, the memory page must have execute permissions. You can modify these memory protection constants with VirtualProtect or VirtualProtectEx.

In the early days, a code cave was memory which was not used by the process, that was already allocated. Nowadays people will allocate their own memory and call this a codecave for whatever reason, I don't know. In most cases, it really doesn't matter, just allocate your own memory because it's easier. If you've created a process handle with permissions to write, then you probably don't have an anticheat problem so you don't need to be worried about being "noisy".

What is a hook?

Hooking is often used interchangeably with the word detouring. In my mind, hooking is specifically the art of hooking into the code and redirecting the flow of execution to your own code. In my mind, detouring doesn't necessarily have to detour to your own code, you can simply be detouring to a different part of the process memory. For instance if you're just jumping over some code you don't want to execute I would describe this as a detour.

Learn more with a an example in our video

https://youtu.be/jTl3MFVKSUM

Common Problem: Mixing up hexadecimal and decimal notation

Cheat Engine Display:

```
mov edi, [esp + 014]
```

^ this is actually 0x14 (cheat engine doesn't say 0x so you may not realize it's hexadecimal)

In Visual Studio you must do:

```
mov edi, [esp + 0x14]
```

just doing `esp + 14` in Visual Studio = decimal

Source Code From the Video

The full project can be downloaded from the attachments.

C++:

```
#include <Windows.h>

bool Hook(void * toHook, void * ourFunct, int len) {
    if (len < 5) {
        return false;
    }

    DWORD curProtection;
    VirtualProtect(toHook, len, PAGE_EXECUTE_READWRITE, &curProtection);

    memset(toHook, 0x90, len);

    DWORD relativeAddress = ((DWORD)ourFunct - (DWORD)toHook) - 5;

    *(BYTE*)toHook = 0xE9;
    *(DWORD*)((DWORD)toHook + 1) = relativeAddress;

    DWORD temp;
    VirtualProtect(toHook, len, curProtection, &temp);

    return true;
}

DWORD WINAPI MainThread(LPVOID param) {
    while (true) {
        if (GetAsyncKeyState(VK_ESCAPE)) break;
```

```
        Sleep(50);
    }

    FreeLibraryAndExitThread((HMODULE)param, 0);

    return 0;
}

BOOL WINAPI DllMain(HINSTANCE hModule, DWORD dwReason, LPVOID lpReserved) {
    switch (dwReason) {
    case DLL_PROCESS_ATTACH:
        CreateThread(0, 0, MainThread, hModule, 0, 0);
        break;
    }

    return TRUE;
}
```

Detour Function Arguments

src argument = the memory address where you want to place your jmp
dst argument = the memory address you want to jump to (probably where you've written your code)
len argument = the number of bytes used by the instruction you're overwriting

The length argument is typically 5 but depending what you overwrite it may be more.

We check the length to be at least 5 bytes because this is the smallest relative jmp in x86. The instructions you will be destroying by overwriting them with a jmp will be at least 5 bytes.

We use VirtualProtect to take permissions of the memory we are overwriting.

We set the memory we're ovewriting with 0x90 which is the NOP (no operationg) instruction, this is not 100% necessary but is a nice failsafe measure. While you're debugging it's also nice to watch the 0x90 get written so you know you're doing the right thing in the right spot. We NOP the entire instruction by giving it the len argument.

Then we calculate the relative address between the destination and src address by subtracting them, we subtract len. The result of this calculation is the relative offset from the last byte we overwrote to the address of our function we are jumping to. Keep in mind, we are using a relative jump, not an absolute jump so this must be calculated at runtime.

Then we write 0xE9 which is the relative jmp instruction, then we add 1 byte so we can write the relative offset.

Then we use VirtualProtect to reset the page permissions to what they were before we modified them.

So now we've jumped to our code which we wrote to memory.

But what about our code?

You have to execute your code inside a declspec naked function, you must preserve the registers and the stack so you don't corrupt the stack of registers and you must execute the stolen bytes. Then you have to jump back to the src+len address.

Attached is the source code, the dummy process and the skeleton for the tutorial

PASSWORD: gh.com

Seperate Trampoline Hook Tutorial

Most people want to use this method, but you're really not ready to learn it yet, which is why we're teaching the regular detour first. This is shown later in the GHB and it's the proper way to hook a function:

https://youtu.be/HLh_9qOkzy0

What is a C++ Detour?

A C++ detour function, also known as function hooking or trampolining, is a technique used in reverse engineering and game hacking to intercept and modify the behavior of a target function at runtime. This is achieved by redirecting the function call to a custom function, allowing the hacker to manipulate, monitor, or extend the functionality of the original function. Detour functions are widely used in game hacking for purposes such as modifying game logic, creating cheats, or analyzing security mechanisms.

How to create a detour function in C++:

1. Identify the target function: Locate the function in the target application that you want to intercept or modify. This can be done using reverse engineering tools like IDA Pro, Ghidra, or x64dbg.
2. Create a custom function: Write your custom function with the same function signature as the target function. This custom function can contain the logic you want to execute when the target function is called.
3. Backup original function code: Before modifying the target function's code, make a backup of the original instructions (usually a few bytes) to ensure you can restore them later. This is essential for creating a trampoline function that allows you to call the original function from your custom function.
4. Replace target function instructions: Overwrite the beginning of the target function with a JMP instruction (or similar) that redirects the execution flow to your custom function. This ensures that whenever the target function is called, it will execute your custom function instead.
5. Create a trampoline function: Write a trampoline function that restores the original instructions and jumps back to the target function. This allows you to call the original function from your custom function if needed.
6. Implement the detour: Use your detour function in your application, and ensure it works as intended by testing and debugging.

]It's worth noting that detouring functions can be detected by anti-cheat systems, so implementing robust and stealthy detours is an important consideration for game hackers. There are also libraries available, such as Microsoft Detours or EasyHook, MinHook that provide a higher-level interface for implementing detours in C++.

Video Walkthrough

Today, we'll be covering hooking techniques which allow us to redirect the program flow and make edits without worrying about overwriting instructions. So, let's dive right in!

Overview

Imagine we have a function to hook, and inside it, we want to overwrite an increment instruction with several instructions that perform some fancy math. We can achieve this by placing a jump at the hook spot that goes to our custom function, where we'll handle the fancy math. Once done, we'll jump back to the hook and continue the program flow. If this seems confusing now, don't worry, as I'll be demonstrating how this works later in the walkthrough.

Coding the Hook Function

First, we'll include Windows.h and write the prototype for the function with the following parameters:

1. A void pointer called toHook, which represents the function or spot we're hooking.
2. Another void pointer called ourFunction, which is the spot our hook jumps to.

3. An integer called length, representing the number of opcodes we're overwriting.

Preparing the Hook

Make sure that the hook length is greater than 5. If it isn't, return false, as the jump we're placing is 5 bytes in size, and we need enough space for it. Next, change the protection to PAGE_EXECUTE_READWRITE to ensure we can edit the assembly at the location. Use the VirtualProtect() function for this.

Clearing Bytes and Setting the Relative Address

Now, set all the bytes to 0x90 or NOP (No Operation) to clear them. This helps avoid undefined behavior due to stray bytes at the end of the jump if the length is greater than 5.

Next, calculate the relative address as follows:

```
DWORD relativeAddress = (DWORD)ourFunction - (DWORD)toHook - 5;
```

This represents the offset from the jump to where we need to go.

Placing the Jump Instruction

Finally, place the jump instruction and restore the original protection using VirtualProtect() again. And that's it! Our hook function is now complete.

Implementing the Hook

Now that we know how to write the hook, let's put it to use. First, write your __declspec(naked) function, which ensures that there's no epilogue or prologue when it compiles. Create a DWORD variable called jumpBackAddr to store the address where we'll jump back after executing our custom function.

Next, get the address you're hooking at and calculate the length of the hook. Remember that you need at least 5 bytes, and you cannot leave any bytes left out. Assign the jumpBackAddr as follows:

```
jumpBackAddr = hookAddr + length;
```

Now, plug the parameters into the hook function and implement your custom function to perform the desired operation, such as doubling the value of a variable.

Don't forget to execute any overwritten instructions in your custom function before jumping back to the original program flow.

Wrapping Up

Once you've implemented the hook, compile and run your program to see it in action. You should see the program flow being redirected to your custom function, executing your desired operation, and then jumping back to the original flow seamlessly.

And that's it! You've now learned how to use hooking techniques to redirect program flow and perform custom operations. Thanks for following along, and don't forget to like and subscribe for more content like this!