Advanced Modding Workshop: C++ Doom External Trainer

Hosted By:
Trevor Mooney
Elliot Tarbet (not present)

Background

Our goal is to teach you what is happening underneath the hood of programs such as Cheat Engine. We hope to give you insight as to how to perform similar modifications by reverse engineering game code through the development of an external trainer for GZDoom.

Requirements:

- **Programming:** Some familiarity with Object-Oriented Programming and C++
- **OS:** Windows
- **Compiler/Interpreter:** A MSVC via VS Code/CLion or GCC and Python 3
- **IDE:** CLion or VSCode for proper configuration and compilation of C++ files.

Setup and Installation (Windows Only)

Follow the Instructions of the Readme to Install the necessary components:

https://github.com/Game-Hacking-Village/cpp_external/blob/main/README.md

Freedoom Engine: https://freedoom.github.io/

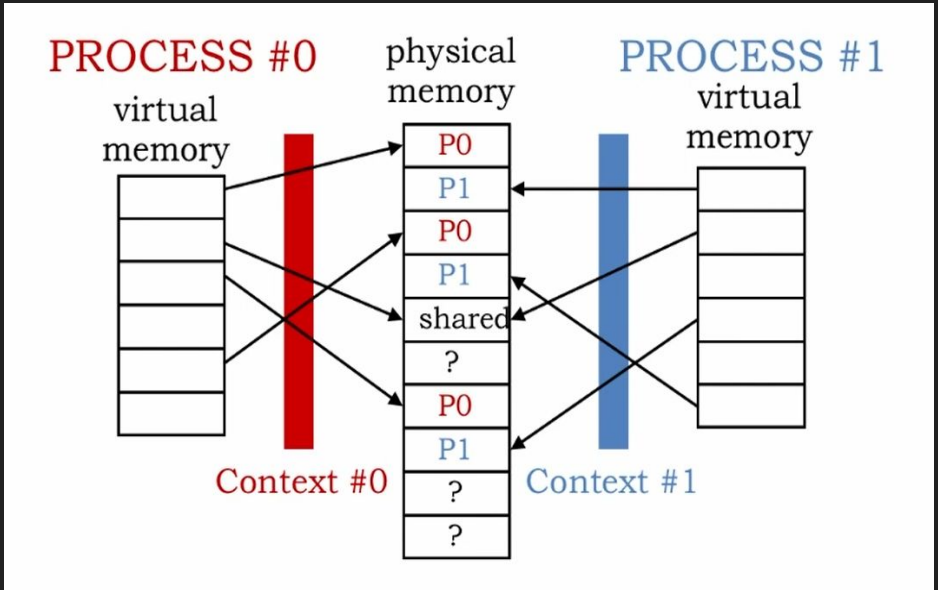
GZDoom Port: https://github.com/ZDoom/gzdoom

Demo code will use CLion IDE and Cheat Engine for straight forward building and compilation

Cheat Engine

- Cheat Engine enables players to triangulate specific memory addresses that are associated with particular values that can be altered
- Functions well with many older games such as Diablo 2 and some modern games such as GTA5
- Keep in mind that Anti-Cheat mechanisms have become quite sophisticated on most modern multiplayer games blocking Cheat Engine’s functionality
- Cheat Engine is only being demonstrated for educational purposes and only works with selected games
- Our focus will be on runtime modding of Weapon Ammo, Armor and Health using an External Trainer
- External Trainer vs. DLL Injection

Accessing Processes



- When a process created, a Virtual Address Space is mapped to physical memory using page tables and MMU; each process is isolated
- In order for one process to access another process Administrative/Root privileges are needed
- We will be making use of three Win32 APIs to attach to a process' handle and read a particular area of memory.

OpenProcess()

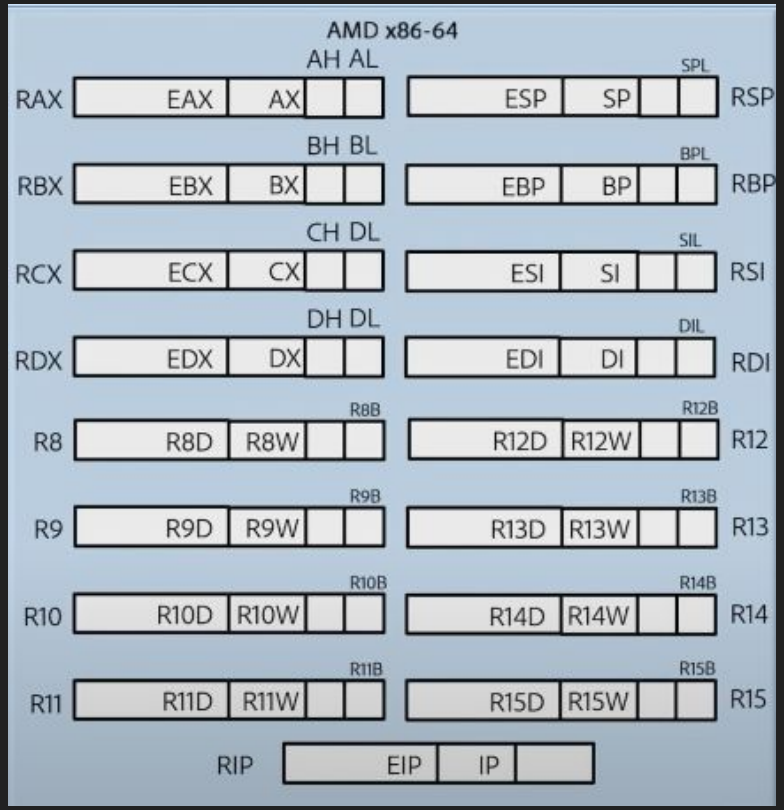
ReadProcessMemory()

WriteProcessMemory()

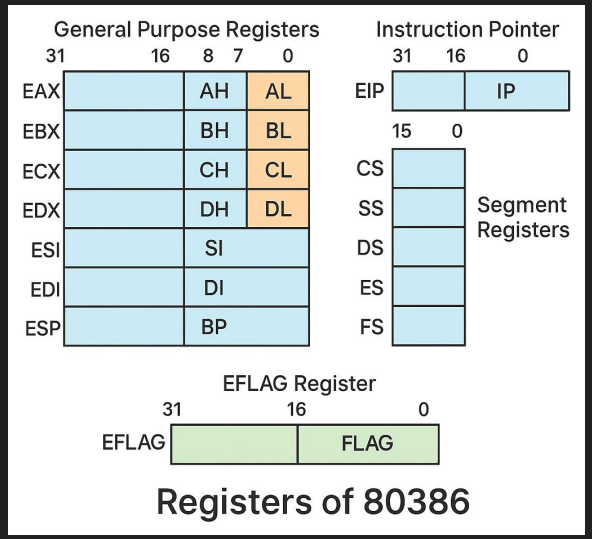
Source: https://www.unknowncheats.me/forum/general-programming-and-reversing/523359-introduction-physical-memory.html

- In Windows many modern games and Anti-Cheat Software are run as protected processes barring the use of these three APIs
- **OpenProcess()** is roughly equivalent to **process_vm_readv()** or **/proc/<pid>/mem** + **ptrace** in Linux

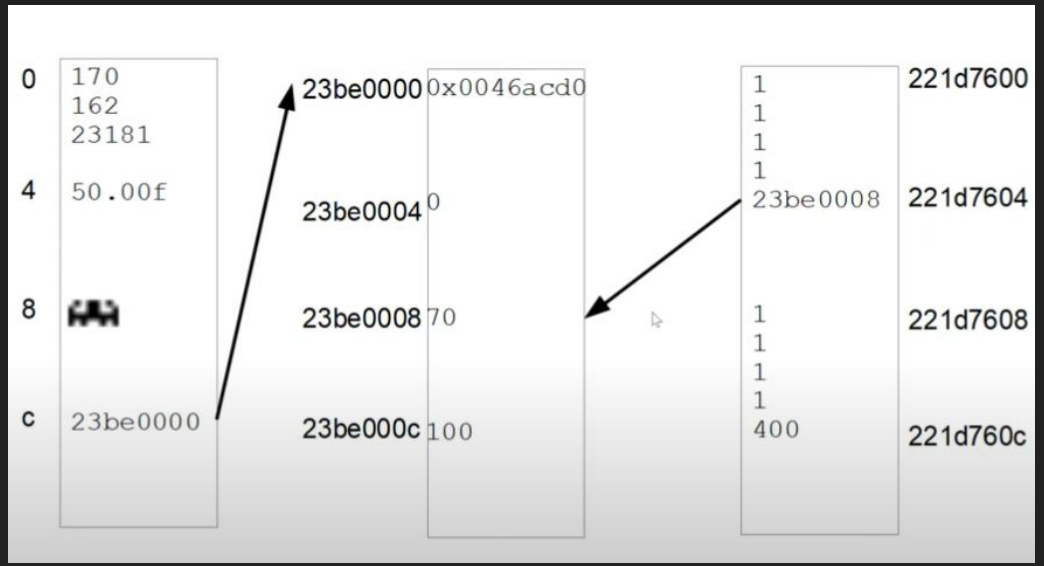
A Brief Introduction to X86 Assembly



- mov [rax], edx -> writes register value to memory
- mov edx, [rax] -> reads from memory into register
- mov eax, [rbp + 00000678] -> Register Indirect Addressing with Displacement; takes memory address stored rbp and add 0x678 (d1656) to it and copy the contents stored at that memory address into eax



Pointer Chasing using Pointer Maps in Cheat Engine



- Each time a game/process is restarted it will be mapped to different location of memory
- Our goal is to find the correct path of pointers and offsets that will not change as we play the game
- Pointer Maps in Cheat Engine

- **Pointer Chasing:** Following chains of pointers until you find the value the desired value in memory
- ie: Memory location c contains a pointer to address 23be0000 which contains address 0x0046acd0

Useful commands and GZDoom cheat codes

1) **Identify process handle using CLI/Cheat Engine**

   Helpful powershell commands:

   gzdoom.exe

   et-Process -name process_name

   ocess -Id 1234

   ***Process Id in PowerShell is base10 in CE it is hex

   ./

   G

   Get-Pr

2) **Identify and target values in Cheat Cheat Engine**

   GZDoom Cheats to assist in finding target addresses:
   https://steamcommunity.com/sharedfiles/filedetails/?id=2197552469

   nlock all weapons: in-game click ‘back tick’ and type *give weapon*

   Unlock max ammo: in-game click ‘back tick’ and type *give ammo*

Workflow to for Cheat Engine and the External Trainer

Our goal is to find stable pointer chains using Cheat Engine to feed into our External Trainer

Cheat Engine Pointer Scanning Tutorial | GH105: https://www.youtube.com/watch?v=rBe8Atevd-4

1) Open Cheat Engine and attach to GZDoom process.

2) Find Target Addresses -> Need to "Next Scan" with changed value (through gameplay) until a single valid address is found.

2) Generate First Pointer Map.

3) Need to restart game and repeat the actions to generate a second pointer map to see which addresses are valid.

4) Do a first address pointer scan (main screen right click) and in Pointerscanner scanneroptions menu click "Use saved pointermap"and the click "Compare Pointermap with other saved pointermaps", select a file and then chose the corresponding address.

Workflow to for Cheat Engine and the External Trainer (Continued)

5) Copy active address from Main Window.

6) Filter results by offset. Pointer Scanner >>> Rescan Memory (Removes pointer chains not using the last pointer offset.)

In the Rescan Pointerlist Window paste copied address into "Address to find" and click on "Must End with Offsets" and fill out the hex offset. Overwrite the previous saved file.

7) Click on 20 or so PointerChains that have the correct base address of "gzdoom.exe".

8) Restart the game and reattach Cheat Engine. Play the game using ammo, taking attacks by enemies, etc.

9) Delete the pointers that are no longer valid and once satisfied with pointer chains that seem to be constant save the work to a .ct Cheat Table file.

Key Files for Building the External Trainer

Offsets.h -> hard code your chosen pointer map/pointer chain

doom.h and **doom.cpp** -> build setter and getter definitions and declarations

external_doom_gui.cpp -> add each target value to key file for building the interface for the GUI of the External Trainer

memory.h and memory.cpp -> resolve_PointerMap

Your Mod Challenge

1) Complete the Doom External Trainer by finding the correct pointer maps for Shotgun Ammo and at least two other weapons

2) Bonus Challenge: Find and Replace Health Instructions in order to have infinite health

Infinite Ammo Assembly Language Replacement

Replacing ammo instructions with NOPs in assembly to create infinite ammo

sigs.h >>> Optional Signature >>> x86 has variable length instructions vs. ARM that has fixed length instructions. X86 instructions can be 1 - 15 bytes

`0x44 0x89 0x1A` → `mov [rdx], r11d`

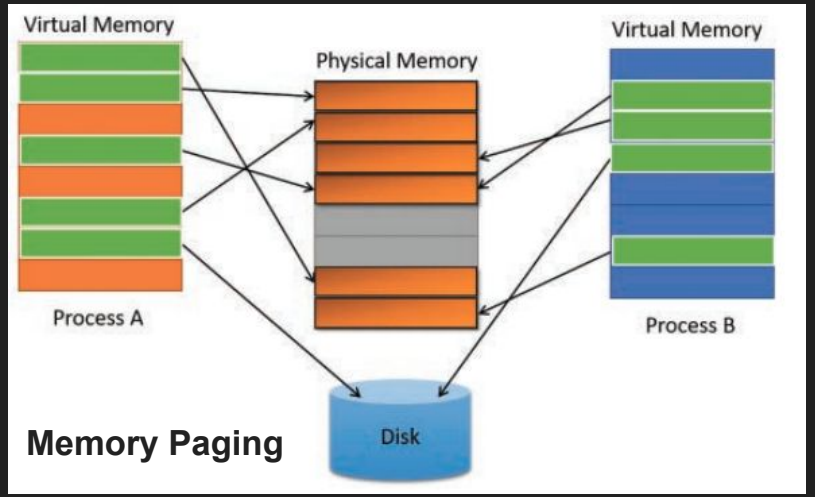
`0x44` is a REX prefix (REX.R=1, meaning it uses extended registers r8–r15),

`0x89` is the opcode for "move 32-bit register into memory",

`0x1A` is the ModR/M byte meaning `[rdx]` as destination, `r11d` as source. (d = lower 32-bits of 64-bit reg)

Key Files and Functions for Building Infinite Ammo Toggle Switch

- **external_doom_gui.cpp** >>> contains toggle switch
- **doom.h and doom.ccp** >>> toggle_unlimited_ammo() and NOPs; PatchInstructions(); setters and getters
- **memory.h and memory.cpp** >>> contains the bulk of the memory scanning logic



Source: https://connormcgarr.github.io/paging/

Additional Resources and Licenses

Additional Resources:

PyLingual White Paper: https://softsec.kaist.ac.kr/~sangkilc/papers/wiedemeier-oakland25.pdf

PyLingual Repository: https://github.com/syssec-utd/pylingual

https://www.cheatengine.org/

https://github.com/ocornut/imgui

https://www.ownedcore.com/forums/world-of-warcraft/world-of-warcraft-bots-programs/79080-cheat-engine-basics-tutorial-steps-1-7-a.html

Licenses:

GZDoom GNU V3 License:https://github.com/ZDoom/gzdoom/blob/master/LICENSE

Freedoom License: https://github.com/freedoom/freedoom/tree/master?tab=License-1-ov-file

Imgui MIT License: https://github.com/ocornut/imgui?tab=MIT-1-ov-file