Inhaltsverzeichnis

Geben Sie die Kapitelüberschrift ein (Stufe 1)1

Geben Sie die Kapitelüberschrift ein (Stufe 2)2

Geben Sie die Kapitelüberschrift ein (Stufe 3)3

Geben Sie die Kapitelüberschrift ein (Stufe 1)4

Geben Sie die Kapitelüberschrift ein (Stufe 2)5

Geben Sie die Kapitelüberschrift ein (Stufe 3)6

# Vorbereitung

Zuerst um mit der Arbeit anzufangen, bereite ich die Folgenden Programme vor.

Auf der offiziellen Webseite vom Cheat Engine (Abk. CE) <https://www.cheatengine.org/> installiere ich das Tool, um damit herausfinden zu können, wie ein Spiel/eine Anwendung funktioniert, und Änderungen daran vornehmen können.

Zunächst installiere ich die integrierte Entwicklungsumgebung (IDE) Software für die Anwendungsentwicklung Visual Studio von Microsoft auf meinen Computer. Dazu muss die Erweiterung für das Programmieren auf der Sprache C++ installiert werden. Mit dieser IDE werde ich den Code für das Mod Menu implementieren.

Als nächstes kommt das Programm ReClass, das die Prozess-Handles eines anderen Prozesses verwendet, um auf die Daten des Ziels zuzugreifen. Hiermit wird die Suche von Offsets vereinfacht.

Das Programm IDA (Interactive Disassembler) wird zuletzt für das Cheaten benötigt, dieser ermöglicht Binärcode in Assemblersprache umzuwandeln, um leichter den Programmcode von dem Spiel zu verstehen.

Zuletzt installiere ich das Spiel Assault Cube, für welches dann dieses Mod Menu erstellt werden soll.

# Durchführung

[Cheat Engine]

**[1]** *Pointer für Munition erstellen*

1.Temporäre RAM-Speicher Adresse für die Munition im Magazin finden.

2.-> "Find out what accesses this address"

3.-> Im Spiel die Menge an Munition verändern und CE betrachten welche Adresse auf diesen Wert zugreift.

4.-> Die Adresse in [ ] eckigen Klammern kopieren, Offset abspeichern

5.-> Adresse und Offset in die pointerTabelle reinkopieren

Address = Value = ?

base ptr -> address + offset4 = address

base ptr -> address + offset3 = address

base ptr -> address + offset2 = address

static base -> address + offset1 = address

6.-> Ab Schritt 3 wiederholen, solange bis man die static base pointer address findet

7.-> "Add Address Manually" ->Pointer Häckchen ->static base address und alle offsets eintragen.

8. Pointer erstellt.

**[2]** *Entity base pointer finden*

1. - > Rechtsklick Munition Pointer

2. - > "Find out what accesses this address"

3. - > Im Spiel die Munitionsanzahl verändern

4. - > [edi]-Adresse kopieren und dann nach diesem hex Wert "New Scan"

5. - > Wenn es schon die statische Adresse für das [edi] vorhanden ist, hat man den Pointer gefunden

**[3]** *Die* *nötigen* *Pointer über die Entity List herausfinden*

-> "Memory View" -> "Tools" -> "Dissect data/structures"

-> Die Enity base address einsetzen -> "S

|  |
| --- |
| **[4]** *Recoil finden* |
|  |

|  |
| --- |
| -> Zuerst füge ich ReClass den ac\_client.exe Prozess an |
|  |

|  |
| --- |
| -> Danach füge ich die statische Entity Player Object Adresse in ReClass hinzu |
|  |

|  |
| --- |
| -> Es werden alle Offsets aufgelistet, die mit dieser Adresse verbunden sind |
|  |

|  |
| --- |
| -> Der Offset für Aktuelle Waffenmunition liegt bei 0x374 |
|  |

|  |
| --- |
| -> Zunächst ändere ich den Typ dieser Adresse auf Pointer, und füge 256 Bytes hinzu, damit die verbundenen Adressbereiche aufgelistet werden |
|  |

|  |
| --- |
| -> Danach finde ich die aktuelle Waffe als String dargestellt, diese ändere ich wieder auf Pointer |
|  |

|  |
| --- |
| -> Ich füge dem Pointer 256 Bytes hinzu und ganz unten finde ich Adressen Werte die vielleicht etwas mit der Aktuellen Waffen zu tun habe |
|  |

|  |
| --- |
| -> Dies prüfe ich indem, ich die Werte auf Int16 Typ wechsle und somit sehe gleiche Werte, die ich im Spiel beobachten kann |
|  |

|  |
| --- |
| -> Zunächst versuche ich herauszufinden, welche der Adressen dem Recoil entsprechen, indem ich die Werte einfach bearbeite und die Folgen im Spiel beobachte |
|  |

|  |
| --- |
| -> Zwei Recoil Adressen wurden gefunden. Eine bewegt die Spieler Sicht nach oben und andere nach links und rechts |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| **[5]** *Recoil Funktion finden* |
|  |

|  |
| --- |
| -> Da die zwei Adressen nicht ausreichen, um vollständig Recoil zu stoppen, versuche ich die Funktion im Disassembler zu finden |
|  |

|  |
| --- |
| -> Nun kopiere ich die Recoil Adresse und gehe zu IDA Anwendung |
|  |

|  |
| --- |
| -> Mit IDA öffne ich die exe Datei von diesem Spiel |
|  |

|  |
| --- |
| -> Danach öffne ich das Plugin Class Informer, welches mir die Namen der Strukturen auflistet |
|  |

|  |
| --- |
| -> Ich suche nach dem Namen "Weapon" und wähle eine Waffe-Struktur aus |
|  |

|  |
| --- |
| -> Jetzt gehe durch alle Funktionen und schaue ob sie für mich interessant sein könnten |
|  |

|  |
| --- |
| -> Ziel ist es eine Funktion zu finden die den Recoil berechnet und aufruft |
|  |

|  |
| --- |
| -> (Man findet sie in dem man sich den Code ausließt und herausfindet wo die zwei Offsets für die Recoil Adressen sich befinden) |
|  |

|  |
| --- |
| -> In der CallRecoil Funktion findet man die Adresse wo die Funktion maybeRecoil aufgerufen wird |
|  |

|  |
| --- |
| -> CE Memory View -> Suche 0046378E |
|  |

|  |
| --- |
| -> Setze ein Breakpoint um diese Adresse zu debuggen und herauszufinden welche Adresse sie aufruft |
|  |

|  |
| --- |
| -> Nun gehe ich zu der Adresse und ändere die Assembler Anweisung auf ret 8 (Mit RET wird ein mit dem CALL-Befehl aufgerufenes Unterprogramm beendet) |
|  |

-> So hat man in dem Spiel keinen Recoil

**[Visual Studio]**

Konsolenkursor entfernen, damit er nicht blinkt. Funktion in der Standartbibliothek von MSDN.

Zunächst Funktion gotoXY, um zu einer bestimmten Zeile in der Konsole springen und den Text umschreiben.

Funktion MoveWindow um die Größe der Konsole anzupassen.

Modulebase Adresse herausfinden. (Im IDA Imagebaseaddress überprüfen)

LocalplayerAddress aus dem Cheat Engine zu der Module Base Address zufügen.

Pointer zu folgenden Adressen mittels FindDMAAddy erstellen: xyz - Positionen, health, superJump.

Großes Titel ACMOD mittels Unicodes (wchar) und einer Art Seite einfügen.

Bibliothek für Farben in der Konsole implementieren.

mit GetAsyncKeyState Tasten für die Cheat-Aktivierung hinzufügen.

Für den Health-Cheat fünf verschiedene States einfügen. Zuerst reset auf 100 und Health-Subtract. Danach Funktion NOP (Funktion mit toten Bytes ersetzen) und anschließend ADD-Cheat (Beim Damage eigenes Leben addieren).

Für ADD und SUM wird die Funktion PatchEX benutzt, um die ausgewählten Bytes zu ersetzen.

Um für das Leben einen bestimmten Wert zu setzen benutzt man aus der windows.h Bibliothek die Funktion WriteProcessMemory, um einen bestimmten Wert einer Adresse aus dem RAM-Speicher zu ändern.

Zunächst für das Munition-Patch benutze ich nur einen Cheat, die Munition bei jedem Schuss zu addieren. Dafür schreibe ich nur ein Byte dieser Funktion um. Und somit ändert sich die ASMBLER Anweisung von dec zu inc (decrement, increment).

Zunächst NOPe ich die Funktion des Recoil und setze die wieder zurück.

Um das Cheat zu beenden wird die Taste 0 auf dem Numpad benutzt.

Für superJump wird die jump Adresse auf vier floats umgeschrieben. dh., wenn ich dies ausführe, springt der Spieler auf vier LE nach oben. Diese Funktion verbinde ich dann mit dem Leerzeichen und somit, wenn man einmal springt, bleibt alles ganz normal und wenn man zwei Mal die Taste drückt, aktiviert sich der Cheat.

Teleport Funktion

Zuerst wird die gewünschte stelle mit einer Taste ausgelesen und wenn die Koordinaten nicht 0 sind, teleportiert man sich auf die gewünschte Stelle.

Speed Hack

Mit dem Cheat Engine sucht man nach Adressen, welche die xyz Position beeinflusst und am Ende patcht man jeweils ein Byte pro laufende Richtung. Also verändert sich die maximale Geschwindigkeit von 1f auf 2f. Im Game Design wird für die Bewegung eine maximale float-Zahl zum Multiplizieren der Geschwindigkeit benutzt und diese schreibe ich somit um.

Und jeweils pro Cheat wird der Cheat zb. NOP/ADD... in der Konsole mittels gotoXY und Farbe geändert.