ASSAULT CUBE MOD MENU

Valentyn Repinskyi

Inhaltsverzeichnis

1 Vorbereitung

1.1 Lernen

Um etwas über das Material für das Modding von Spielen zu erfahren, habe ich den folgenden Beitrag im Internet benutzt: https://guidedhacking.com/threads/ghb1-start-here-beginner-guide-to-game-hacking.5911/

Diese Seite enthält alles, was man wissen muss, wenn man mit dem Hacken beginnen will. Auf dieser Seite habe ich mir zahlreiche Tutorials zum Thema Game Hacking angesehen. Zuerst lernt man, wie der RAM-Speicher funktioniert, dann kann man mit Tools wie Cheat Engine oder ReClass direkt mit dem Hacken loslegen. Aber wenn man solche Dinge automatisieren will, muss man eine Programmiersprache lernen. Auf der folgenden Website: learncpp.com, habe ich 21 Kapitel durchgearbeitet und unterschiedliche Aufgaben absolviert. Da diese Sprache nicht meine erste ist, war es für mich viel einfacher, sie zu lernen. Während ich C++ lernte, programmierte ich meine eigenen kleinen Spiele, Tools usw. Dadurch bekam ich auch einen Überblick darüber, wie Spiele auf dieser Sprache aufgebaut sind und wie man sie hacken kann.

1.2 Software

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

1.2.1 Cheat Engine

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.

1.2.2 Visual Studio

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.

1.2.3 ReClass

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.

1.2.4 Interactive Disassembler

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

1.2.5 Assault Cube

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

2 Durchführung

2.1 Cheat Engine / ReClass

2.1.1 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.

2.1.2 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

2.1.3 Die nötigen Pointer über die Entity List herausfinden

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

-> Die Enity base address einsetzen -> "S

2.1.4 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

2.1.5 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

2.2 Visual Studio

2.2.1 Konsolenkursor entfernen

Damit der Konsolenkursor nicht in der Konsole blinkt, schalte ich ihn mit einer Standartfunktion aus MSDN-Webseite.

2.2.2 Funktion gotoXY

Um die Werte in der Konsole zu ändern, wie z.B. Heath auf 999 zu setzen, muss ich in der Konsole zu einer bestimmten Position springen und den Text an dieser Stelle umschreiben.

2.2.3 Funktion MoveWindow

Damit die wichtigsten Sachen in der Konsole angezeigt werden, wird eine Funktion zur Anpassung der Konsolengröße benutzt.

2.2.4 Pointer erstellen

Damit alle gefundenen Pointer eine Adresse im Spiel zugeordnet werden könnten, erstellet man eine Variable mit der Modulebase Adresse. Diese kann man auch im IDA als Imagebaseaddress nachprüfen.

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

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

2.2.5 Darstellung

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

Bibliothek für Farben in der Konsole implementieren.

2.2.6 Cheat - Aktivierung

Mittels Funktion GetAsyncKeyState Tasten für die Cheat-Aktivierung hinzufügen.

Für den Health-Cheat werden fünf verschiedene States eingefügt. 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.

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 Funktion wird der Cheat zb. NOP/ADD... in der Konsole mittels gotoXY den Wert umschreiben.

Um das Cheat zu beenden wird die Taste 0 auf dem Numpad benutzt und alle Hacks werden aus dem Spiel entfernt.

3 Anhang

Projekt auf GitHub mit Bildern und Cheat Tables: https://github.com/A2E9/AC_External

3.1 Tagesberichte

14.02.2022

Erlernen der C++ Sprache: Fundamental Data Types: learncpp.com

21.02.2022

Installieren und Konfigurieren der Software: Cheat Engine, ReClass, AC

28.02.2022

Installieren und Konfigurieren der Software: Visual Studio und IDA. (lange Installation). Lernen der C++ Sprache: Compound Types: References and Pointers: learncpp.com

07.03.2022

Pointer für Munition mittels Cheat Engine erstellen.

14.03.2022

Entity base Pointer finden und damit die nötigen Pointer herausfinden.

21.03.2022

Recoil Funktion finden.

28.03.2022

Weitersuche der Recoil-Funktion.

04.04.2022

In Visual Studio neues Projekt erstellen und für das Mod Menu vorbereiten.

Mod Konsole anpassen.

25.04.2022

Verweis auf Pointer erstellen.

02.05.2022

Programmierung von Cheats. Tasten für Cheat-Aktivierung einbinden

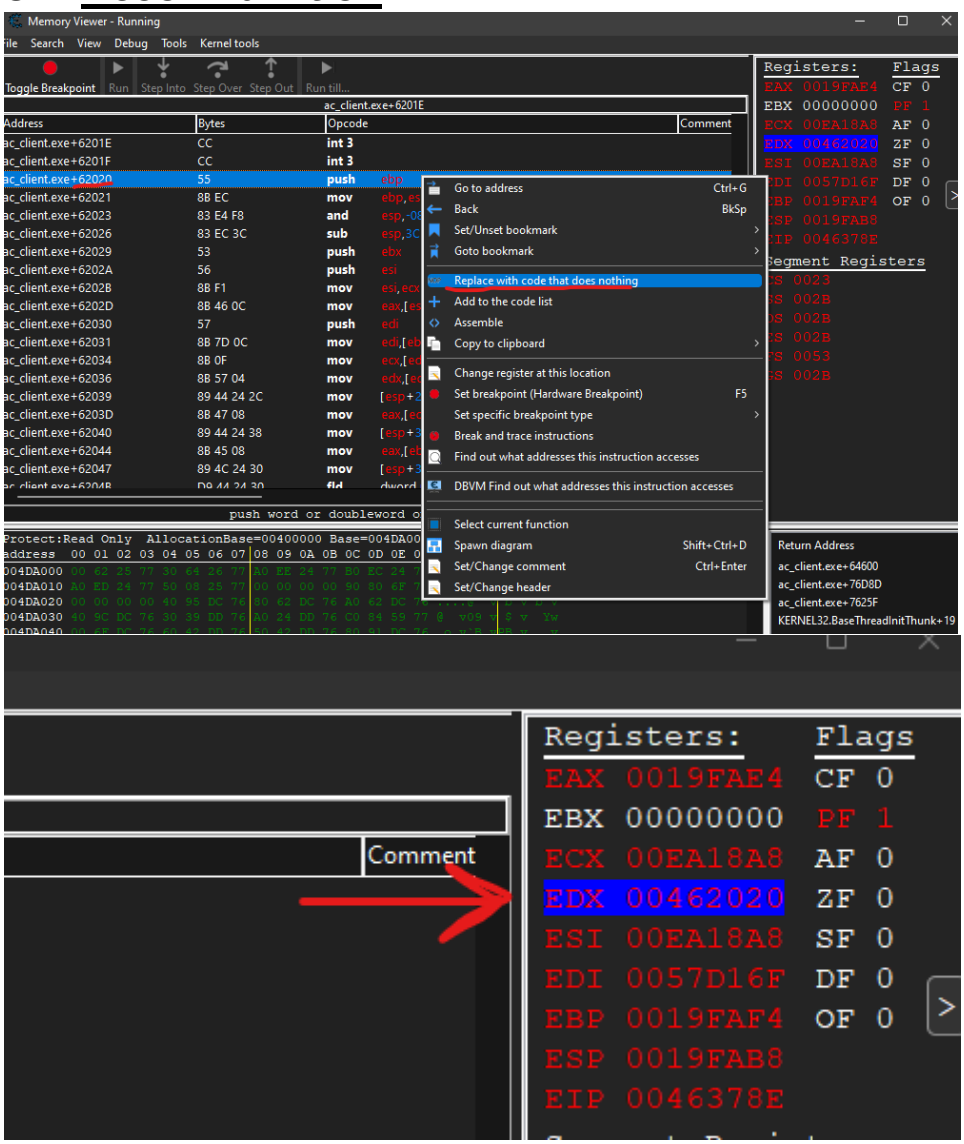
09.05.2022

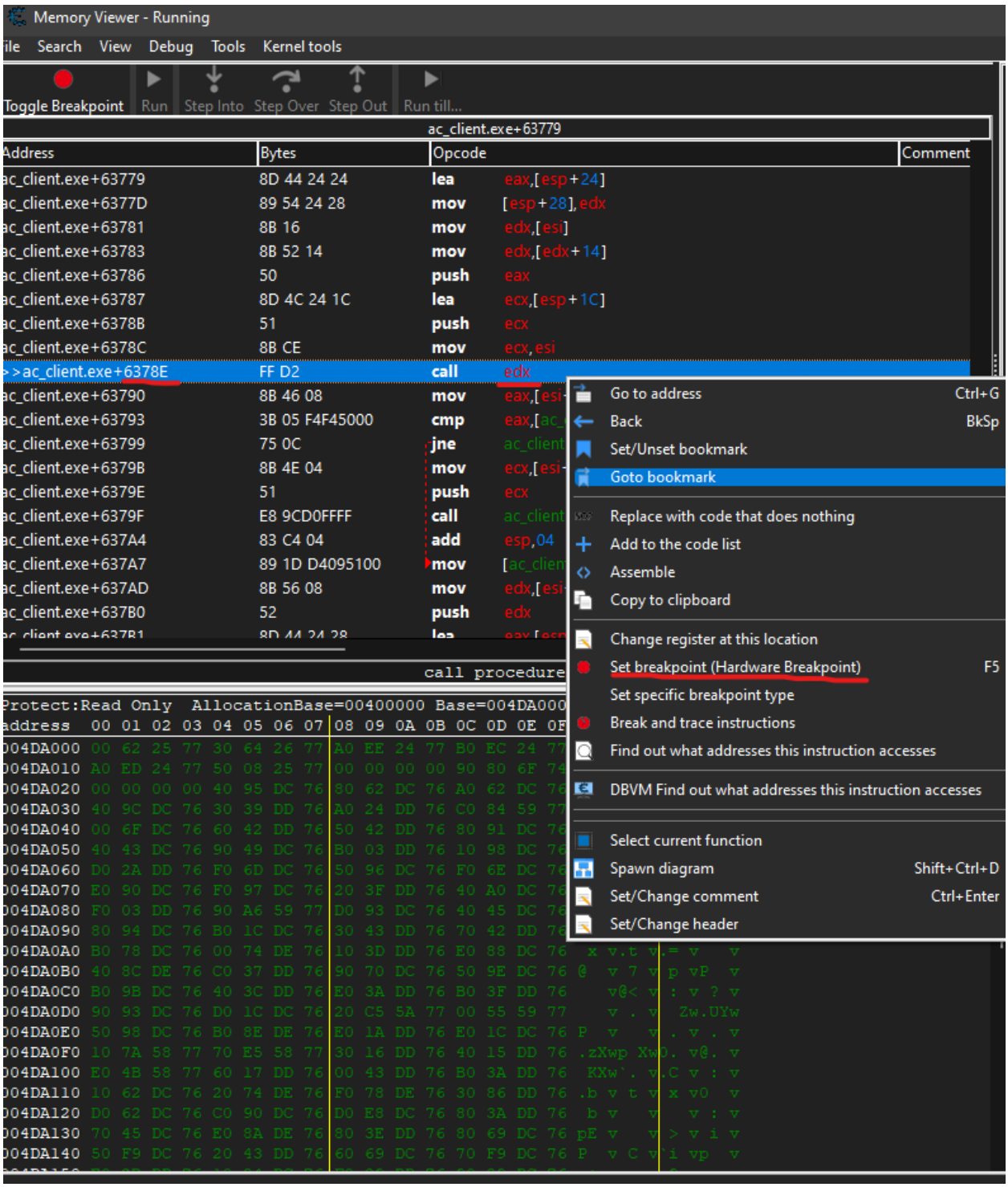
Programmierung von Cheats. Leben, Munition, Recoil, Speed Hack

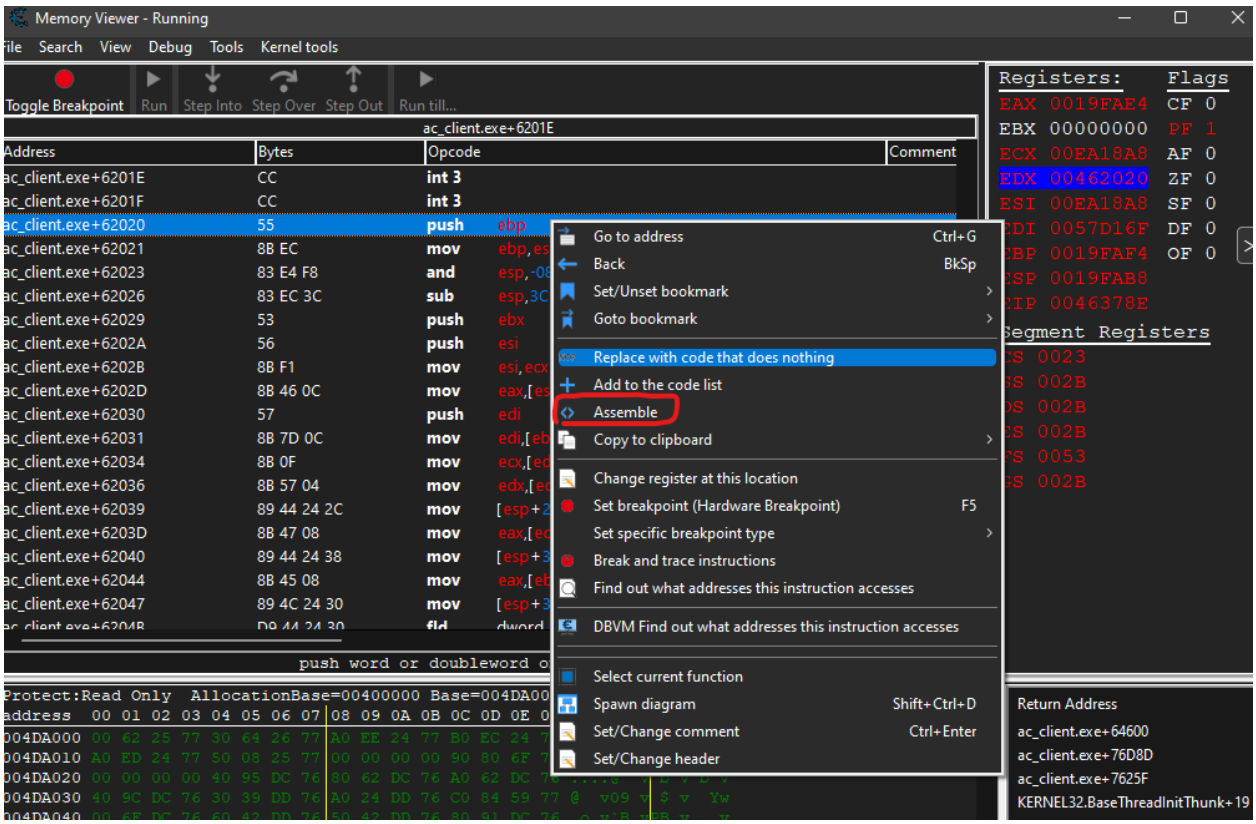
23.05.2022

Programmierung von Cheats. Teleport, Jump Hack, Cheat Exit

3.2 Recoil Funktion




Memory Viewer - Running
File Search View Debug Tools Kernel tools
Toggle Breakpoint
Run
Step Into
Step Over
Step Out
Run till...
ac_client.exe+63779
Address
Bytes
Opcode
Comment
ac_client.exe+63779 8D 44 24 24 lea eax,[esp+24]
ac_client.exe+6377D 89 54 24 28 mov [esp+28],edx
ac_client.exe+63781 8B 16 mov edx,[esi]
ac_client.exe+63783 8B 52 14 mov edx,[edx+14]
ac_client.exe+63786 50 push eax
ac_client.exe+63787 8D 4C 24 1C lea ecx,[esp+1C]
ac_client.exe+6378B 51 push ecx
ac_client.exe+6378C 8B CE mov ecx,esi
>>ac_client.exe+6378E FF D2 call edx
ac_client.exe+63790 8B 46 08 mov eax,[esi
ac_client.exe+63793 3B 05 F4F45000 cmp eax,[ac_
ac_client.exe+63799 75 0C jne ac_client
ac_client.exe+6379B 8B 4E 04 mov ecx,[esi
ac_client.exe+6379E 51 push ecx
ac_client.exe+6379F E8 9CD0FFFF call ac_client
ac_client.exe+637A4 83 C4 04 add esp,04
ac_client.exe+637A7 89 1D D4095100 mov [ac_clien
ac_client.exe+637AD 8B 56 08 mov edx,[esi
ac_client.exe+637B0 52 push edx
Go to address Ctrl+G
Back BkSp
Set/Unset bookmark
Goto bookmark
Replace with code that does nothing
Add to the code list
Assemble
Copy to clipboard
Change register at this location
Set breakpoint (Hardware Breakpoint) F5
Set specific breakpoint type
Break and trace instructions
Find out what addresses this instruction accesses
DBVM Find out what addresses this instruction accesses
Select current function
Spawn diagram Shift+Ctrl+D
Set/Change comment Ctrl+Enter
Set/Change header
call procedure
Protect:Read Only AllocationBase=00400000 Base=004DA000
address 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
004DA000 00 62 25 77 30 64 26 77 A0 EE 24 77 B0 EC 24 77
004DA010 A0 ED 24 77 50 08 25 77 00 00 00 00 90 80 6F 74
004DA020 00 00 00 00 40 95 DC 76 80 62 DC 76 A0 62 DC 76
004DA030 40 9C DC 76 30 39 DD 76 A0 24 DD 76 C0 84 59 77
004DA040 00 6F DC 76 60 42 DD 76 50 42 DD 76 80 91 DC 76
004DA050 40 43 DC 76 90 49 DC 76 B0 03 DD 76 10 98 DC 76
004DA060 D0 2A DD 76 F0 6D DC 76 50 96 DC 76 F0 6E DC 76
004DA070 E0 90 DC 76 F0 97 DC 76 20 3F DD 76 40 A0 DC 76
004DA080 F0 03 DD 76 90 A6 59 77 D0 93 DC 76 40 45 DC 76
004DA090 80 94 DC 76 B0 1C DC 76 30 43 DD 76 70 42 DD 76
004DA0A0 B0 78 DC 76 00 74 DE 76 10 3D DD 76 E0 88 DC 76
004DA0B0 40 8C DE 76 C0 37 DD 76 90 70 DC 76 50 9E DC 76
004DA0C0 B0 9B DC 76 40 3C DD 76 E0 3A DD 76 B0 3F DD 76
004DA0D0 90 93 DC 76 D0 1C DC 76 20 C5 5A 77 00 55 59 77
004DA0E0 50 98 DC 76 B0 8E DE 76 E0 1A DD 76 E0 1C DC 76
004DA0F0 10 7A 58 77 70 E5 58 77 30 16 DD 76 40 15 DD 76
004DA100 E0 4B 58 77 60 17 DD 76 00 43 DD 76 B0 3A DD 76
004DA110 10 62 DC 76 20 74 DE 76 F0 78 DE 76 30 86 DD 76
004DA120 D0 62 DC 76 C0 90 DC 76 D0 E8 DC 76 80 3A DD 76
004DA130 70 45 DC 76 E0 8A DE 76 80 3E DD 76 80 69 DC 76
004DA140 50 F9 DC 76 20 43 DD 76 60 69 DC 76 70 F9 DC 76


Memory Viewer - Running
File Search View Debug Tools Kernel tools
Toggle Breakpoint
Run
Step Into
Step Over
Step Out
Run till...
ac_client.exe+6201E
Address
Bytes
Opcode
Comment
ac_client.exe+6201E CC int 3
ac_client.exe+6201F CC int 3
ac_client.exe+62020 55 push ebp
ac_client.exe+62021 8B EC mov ebp,es
ac_client.exe+62023 83 E4 F8 and esp,-08
ac_client.exe+62026 83 EC 3C sub esp,3C
ac_client.exe+62029 53 push ebx
ac_client.exe+6202A 56 push esi
ac_client.exe+6202B 8B F1 mov esi,ecx
ac_client.exe+6202D 8B 46 0C mov eax,[es
ac_client.exe+62030 57 push edi
ac_client.exe+62031 8B 7D 0C mov edi,[eb
ac_client.exe+62034 8B 0F mov ecx,[ec
ac_client.exe+62036 8B 57 04 mov edx,[ec
ac_client.exe+62039 89 44 24 2C mov [esp+2
ac_client.exe+6203D 8B 47 08 mov eax,[ec
ac_client.exe+62040 89 44 24 38 mov [esp+3
ac_client.exe+62044 8B 45 08 mov eax,[eb
ac_client.exe+62047 89 4C 24 30 mov [esp+3
Go to address Ctrl+G
Back BkSp
Set/Unset bookmark
Goto bookmark
Replace with code that does nothing
Add to the code list
Assemble
Copy to clipboard
Change register at this location
Set breakpoint (Hardware Breakpoint) F5
Set specific breakpoint type
Break and trace instructions
Find out what addresses this instruction accesses
DBVM Find out what addresses this instruction accesses
Select current function
Spawn diagram Shift+Ctrl+D
Set/Change comment Ctrl+Enter
Set/Change header
Registers:
Flags
EAX 0019FAE4 CF 0
EBX 00000000 PF 1
ECX 00EA18A8 AF 0
EDX 00462020 ZF 0
ESI 00EA18A8 SF 0
EDI 0057D16F DF 0
EBP 0019FAF4 OF 0
ESP 0019FAB8
EIP 0046378E
Segment Registers
CS 0023
SS 002B
DS 002B
ES 002B
FS 0053
GS 002B
push word or doubleword o
Protect:Read Only AllocationBase=00400000 Base=004DA00
address 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0
004DA000 00 62 25 77 30 64 26 77 A0 EE 24 77 B0 EC 24 7
004DA010 A0 ED 24 77 50 08 25 77 00 00 00 00 90 80 6F 7
004DA020 00 00 00 00 40 95 DC 76 80 62 DC 76 A0 62 DC 76
004DA030 40 9C DC 76 30 39 DD 76 A0 24 DD 76 C0 84 59 77
Return Address
ac_client.exe+64600
ac_client.exe+76D8D
ac_client.exe+7625F
KERNEL32.BaseThreadInitThunk+19