INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE MINAS GERAIS - *CAMPUS* BETIM
BACHARELADO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Gabriel Santos Gollo do Amaral

TÉCNICAS DE ENGENHARIA REVERSA E MANIPULAÇÃO DE MEMÓRIA: uma abordagem prática em jogos digitais

Betim
2025

GABRIEL SANTOS GOLLO DO AMARAL

TÉCNICAS DE ENGENHARIA REVERSA E MANIPULAÇÃO DE MEMÓRIA: uma abordagem prática em jogos digitais

Trabalho de Conclusão de Curso apresentado à banca examinadora do curso de Engenharia de Controle e Automação do Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais *Campus* Betim, como parte dos requisitos para obtenção do título de Bacharel em Engenharia de Controle e Automação.

Orientador: Prof. Me. Virgil Del Duca Almeida

Betim
2025

FICHA CATALOGRÁFICA

A485t Amaral, Gabriel Santos Gollo do

Técnicas de engenharia reversa e manipulação de memória: uma abordagem prática em jogos digitais / Gabriel Santos Gollo do Amaral. – 2025.

46 f. : il.

Trabalho de conclusão de curso (Bacharelado em Engenharia de Controle e Automação) - Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais, Câmpus Betim, 2025.

Orientação: prof. Me. Virgil Del Duca Almeida

1. Engenharia reversa. 2. Análise de software. 3. Manipulação de memória. 4. Jogos digitais. 5. Engenharia de Controle e Automação. I. Amaral, Gabriel Santos Gollo do. II. Título.

CDU: 004.41

Ficha catalográfica elaborada pelo Bibliotecário Denísio Pereira Marcos CRB-6/3142

Gabriel Santos Gollo do Amaral

TÉCNICAS DE ENGENHARIA REVERSA E MANIPULAÇÃO DE MEMÓRIA: uma abordagem prática em jogos digitais

Trabalho de Conclusão de Curso apresentado à banca examinadora do curso de Engenharia de Controle e Automação do Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais *Campus* Betim, como parte dos requisitos para obtenção do título de Bacharel em Engenharia de Controle e Automação.

Aprovado em: 28 / 07 / 2025 pela banca examinadora:

Prof. Me. Virgil Del Duca Almeida (Orientador) - IFMG

Dr. Arthur Hermano Rezende Rosa - IFMG

Prof. Me. Maurício Monteiro da Silva - IFMG

Dedico este trabalho à minha família, que sempre foi minha base e inspiração, e aos amigos que estiveram ao meu lado ao longo da graduação.

À minha criança interior, que um dia sonhou em desvendar os mistérios por trás dos jogos e agora realiza o desejo de entender como essas peças de arte digital podem ser exploradas e modificadas.

Aos apaixonados por jogos digitais, que encontram neles não apenas diversão, mas também conexões e refúgio, e que, com paciência, enfrentam as inevitáveis artimanhas de trapaceiros nas suas jornadas virtuais.

AGRADECIMENTOS

Agradeço, primeiramente, à minha família, por todo o apoio, pela paciência e por acreditarem em mim mesmo nos momentos mais desafiadores desta jornada. Ao Professor Mestre Virgil, pelas aulas bem estruturadas que tive o prazer de presenciar na faculdade e pela sua forma humanizada, revigorante e amigável de tratar os alunos. Suas orientações e dedicação foram fundamentais para meu desenvolvimento acadêmico e pessoal. Aos amigos que estiveram ao meu lado durante a graduação, compartilhando desafios, alegrias e aprendizados que tornaram essa caminhada mais leve e marcante. À minha namorada, Aline Morais, por sempre estar ao meu lado nas horas mais difíceis, por seus indispensáveis conselhos e ensinamentos, e por ser uma fonte de inspiração e motivação.

"Você não precisa ser um astronauta para ser um explorador, apenas precisa de um espírito curioso e imaginação".
Carl Sagan

RESUMO

Este estudo investiga técnicas de engenharia reversa e manipulação de memória aplicadas a jogos digitais, visando demonstrar os processos na prática. Apresenta-se uma revisão teórica sobre engenharia reversa, abordando métodos de análise estática e dinâmica, bem como o uso de ferramentas como desmontadores de binários, depuradores e o software Cheat Engine. A manipulação de memória é utilizada para leitura e alteração de variáveis críticas durante a execução do jogo. Para validação dos conceitos apresentados, foi desenvolvida uma prova de conceito utilizando o jogo de código aberto Assault Cube, selecionado pela acessibilidade e ausência de mecanismos de proteção. Com o auxílio da biblioteca Memory.dll e do framework .NET com a linguagem C#, foi criada uma ferramenta capaz de ler e modificar variáveis como pontos de vida, munição e quantidade de granadas do jogador. Entre as funcionalidades implementadas, destacam-se a visualização da posição de inimigos através das paredes (ESP), o auxílio de mira (aimbot) e a atualização contínua de atributos do jogador. Os resultados evidenciam a viabilidade da aplicação das técnicas de engenharia reversa para mapear estruturas internas e manipular variáveis em tempo real, mesmo sem acesso ao código-fonte do jogo. Observa-se, contudo, que a ausência de sistemas anti-trapaça facilitou o processo, o que limita a generalização dos resultados para jogos comerciais protegidos. Conclui-se que o domínio dessas técnicas favorece a compreensão do funcionamento interno dos jogos digitais e incentiva o desenvolvimento de habilidades em análise de software.

Palavras-chave: Engenharia Reversa. Manipulação de Memória. Jogos Digitais. Análise Dinâmica. Prova de Conceito.

ABSTRACT

This study investigates reverse engineering and memory manipulation techniques applied to digital games, aiming to demonstrate these processes in practice. A theoretical review of reverse engineering is presented, covering methods of static and dynamic analysis, as well as the use of tools such as binary disassemblers, debuggers, and the Cheat Engine software. Memory manipulation is used to read and modify critical variables during the execution of the game. To validate the presented concepts, a proof of concept was developed using the open-source game Assault Cube, selected for its accessibility and lack of protection mechanisms. With the support of the Memory.dll library and the .NET framework using the C# language, a tool was created capable of reading and modifying variables such as health points, ammunition, and the player's grenade count. Among the implemented features, the visualization of enemy positions through walls (ESP), aim assistance (aimbot), and continuous updating of player attributes stand out. The results demonstrate the feasibility of applying reverse engineering techniques to map internal structures and manipulate variables in real time, even without access to the game's source code. It is observed, however, that the absence of anti-cheat systems facilitated the process, which limits the generalization of the results to protected commercial games. It is concluded that mastering these techniques enhances the understanding of the internal functioning of digital games and encourages the development of software analysis skills.

Keywords: Reverse Engineering. Memory Manipulation. Digital Games. Dynamic Analysis. Proof of Concept.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
DMA	Direct Memory Access
DLL	Dynamic-link library
FPS	First-Person Shooter
ESP	Extra Sensory Perception
IDE	Integrated Development Environment
RAM	Random-access Memory
MMO	Massively Multiplayer Online
ASLR	Address Space Layout Randomization
MVP	Model-View-Projection

SUMÁRIO

1 INTRODUÇÃO

O cenário da cibersegurança global tem se tornado cada vez mais complexo e preocupante. Segundo European Union Agency for Cybersecurity (2023), entre 2022 e 2023, observou-se um aumento significativo na variedade e na quantidade de ataques cibernéticos, impulsionado por diversos fatores, como a intensificação de conflitos geopolíticos, a emergência de novos grupos *hackers* e ativistas, a profissionalização de modelos criminosos como o *Ransomware-as-a-Service* e o avanço de técnicas de engenharia social. Esses ataques afetam desde sistemas financeiros até plataformas de entretenimento e reforça a centralidade da segurança da informação na era digital.

Entre os sistemas que apresentam desafios relevantes do ponto de vista da segurança, destacam-se os jogos digitais, especialmente os jogos online massivos (*Massively Multiplayer Online - MMO*), por operarem sobre arquiteturas distribuídas complexas e por envolverem interações em tempo real com milhares de usuários. Segundo Robles *et al.* (2008), essas características tornam os jogos online massivos particularmente suscetíveis a falhas de segurança relacionadas a estado e sincronização, o que pode ser explorado por jogadores mal-intencionados. Além disso, muitos desses jogos possuem economias virtuais robustas que, ao se conectarem ao mundo real por meio da comercialização de itens, personagens e moedas, criam incentivos concretos para atividades fraudulentas, como a exploração de bugs, manipulação de memória e trapaças.

Nesse cenário, a engenharia reversa e a manipulação de memória apresentam-se como técnicas amplamente utilizadas, tanto para a exploração de fraquezas quanto para a criação de soluções de segurança. A engenharia reversa, que envolve a análise de softwares para entender as lógicas de seu funcionamento interno, tem sido historicamente associada à criação de trapaças em jogos digitais. No entanto, essas mesmas técnicas, quando utilizadas de forma ética e com fins educacionais, podem fornecer conhecimentos valiosos tanto para a identificação de vulnerabilidades quanto para melhoria da compreensão do funcionamento interno de sistemas digitais. Já a manipulação de memória, em particular, permite que se acesse ou altere o estado interno de um software ou jogo, expondo falhas que podem ser exploradas por trapaceiros para obterem vantagens injustas.

Portanto, este trabalho busca por meio das técnicas de engenharia reversa e de manipulação de memória, demonstrar o processo de exploração de vulnerabilidades em jogos digitais desprovidos de mecanismos de proteção. Ao longo deste estudo, serão abordadas as ferramentas e métodos utilizados para realizar a engenharia reversa e a manipulação de memória, bem como os desafios e resultados obtidos durante este processo.

1.1 Objetivos

1.1.1 *Objetivo geral*

O objetivo geral deste estudo é explorar e demonstrar técnicas de engenharia reversa e manipulação de memória aplicadas a jogos digitais. Para isso, será desenvolvido um estudo teórico sobre os principais conceitos da área, seguido da implementação de uma prova de conceito, na qual uma ferramenta será criada para manipular a memória do jogo digital *Assault* Cube.

1.1.2 *Objetivos específicos*

Para alcançar o objetivo geral, o trabalho está dividido nas seguintes etapas:

- *Apresentar a análise estática e dinâmica, e ferramentas amplamente empregadas na engenharia reversa e manipulação de memória, como desmontadores de binários, depuradores e o software* ***Cheat Engine*** *(HEIJNEN, 2000).*
- *Explorar as técnicas de manipulação de memória, com foco na leitura e escrita de dados em tempo de execução, por meio da utilização de uma biblioteca que simplifica o acesso à API do sistema operacional.*
- *Desenvolver uma prova de conceito prática, implementando uma ferramenta de manipulação de memória no jogo open source Assault Cube, a fim de validar e demonstrar as técnicas estudadas.*

1.2 Justificativa

Este trabalho foi motivado por um interesse pessoal no estudo de engenharia reversa e manipulação de memória aplicadas a jogos digitais.

A escolha de aplicar essas técnicas em jogos digitais se deve ao fato de que os jogos digitais são tecnicamente relevantes e oferecem um ambiente prático, desafiador e acessível para a aplicação dessas técnicas.

Embora a engenharia reversa e a manipulação de memória sejam frequentemente utilizadas para fins de exploração e identificação de vulnerabilidades, seu uso com finalidade educacional, especialmente no contexto de jogos, permite compreender o funcionamento interno dos jogos digitais e desenvolver habilidades técnicas em análise de software.

1.3 Isenção de Responsabilidade

Este trabalho é realizado exclusivamente para fins acadêmicos. O autor e o orientador não se responsabilizam por qualquer uso indevido das técnicas e exemplos descritos. É importante

ressaltar que o jogo digital Assault Cube, abordado no Capítulo 4, é um software freeware, cuja modificação e exploração são lícitas conforme as leis brasileiras e internacionais.

1.4 Organização do Texto

No Capítulo 2, é apresentada uma revisão bibliográfica sobre Engenharia Reversa e Manipulação de Memória, abordando conceitos fundamentais como *disassembly*, *debugging*, análise estática e dinâmica, além de técnicas utilizadas para explorar e modificar a memória de jogos digitais.

O Capítulo 3 descreve a metodologia utilizada no desenvolvimento do trabalho, detalhando as ferramentas, estratégias e processos adotados para a implementação da prova de conceito.

No Capítulo 4, são apresentados os resultados obtidos com a manipulação da memória no jogo Assault Cube, demonstrando as funcionalidades implementadas e discutindo as implicações da técnica aplicada.

Por fim, o Capítulo 5 traz as conclusões do estudo, destacando os principais achados do trabalho e sugerindo possíveis direções para pesquisas futuras na área de Engenharia Reversa aplicada a jogos digitais.

2 REFERENCIAL TEÓRICO

Neste capítulo, serão discutidos os principais conceitos e fundamentos do estudo sobre técnicas de engenharia reversa e manipulação de memória, aplicadas ao desenvolvimento de trapaças em jogos digitais.

2.1 Engenharia Reversa de Software

2.1.1 *O que é engenharia reversa?*

Segundo Eilam (2011), a engenharia reversa é uma prática que pode ser comparada ao método científico, pois busca entender e documentar o design e a estrutura de algo já existente, permitindo obter conhecimento a partir da observação e análise de um sistema ou software já existente. E no contexto de software, essa prática envolve analisar seu funcionamento sem acesso ao código-fonte original. A engenharia reversa pode ser utilizada nos mais diversos ambientes e cenários.

Segundo Melo *et al.* (2011), a análise de *malware* utiliza técnicas de engenharia reversa para compreender o funcionamento de códigos maliciosos, visando criar métodos eficazes de detecção e proteção. Esse processo permite que profissionais de segurança da informação identifiquem a origem, o comportamento e o impacto potencial do *malware*, fornecendo uma base para desenvolver contramedidas a ataques específicos.

No cenário dos jogos digitais, o trabalho de engenharia reversa realizado por um desenvolvedor independente destacou a importância do conhecimento dessa prática na otimização de desempenho, ao identificar uma falha que causava um aumento significativo no tempo de carregamento do ***Grand Theft Auto V (2013)***, e o mesmo apenas utilizando-se de ferramentas de engenharia reversa identificou, e desenvolveu a sua própria correção que reduziu o tempo de espera em até 70% (T0ST, 2021). Essa descoberta exemplifica como ferramentas de engenharia reversa podem ajudar em uma análise detalhada, revelando até mesmo uma ineficiência no software.

2.1.2 *Abordagens*

Segundo Eilam (2011), a engenharia reversa pode ser conduzida por diferentes abordagens, dependendo do software analisado, da plataforma em que ele opera e dos objetivos da análise. Dentre essas abordagens, destacam-se duas metodologias fundamentais: a análise estática, que examina o código sem executá-lo, e a análise dinâmica, realizada com o software em execução.

A combinação das abordagens estática e dinâmica permite uma análise mais completa do software, reunindo informações sobre sua estrutura interna, e também sobre seu comportamento em tempo de execução. Essa estratégia é amplamente empregada na identificação de vulnerabilidades e na engenharia reversa de programas sem acesso ao código-fonte.

2.1.2.1 *Analise Estática*

A análise estática, também chamada de análise offline, consiste em examinar os arquivos de um software sem executá-lo diretamente. De acordo com Eilam (2011), esse processo pode envolver o uso de ferramentas desmontadoras de binário (*disassemblers*), que traduzem o código binário em uma representação legível em linguagem *assembly*. Esse tipo de análise permite identificar estruturas do programa, localizar funções e compreender sua lógica sem a necessidade de interagir com sua execução. No entanto, uma limitação desse método é a ausência de informações sobre o fluxo de dados em tempo de execução, o que pode dificultar a interpretação completa do comportamento do software.

2.1.2.2 *Análise Dinâmica*

Segundo Eilam (2011), a análise dinâmica, envolve a execução do software em um ambiente controlado para observar seu comportamento em tempo real. Essa abordagem possibilita examinar como os dados internos do programa influenciam seu fluxo de execução. Ferramentas como os depuradores (*debbugers*) são amplamente empregadas nesse contexto, pois permitem que o programa seja executado passo a passo, e fornecem recursos como inspeção de registradores, análise da pilha de chamadas e pontos de interrupção para interromper a execução do código em locais estratégicos. Além disso, monitoradores de sistema podem ser utilizados para capturar interações com arquivos, rede e dispositivos de entrada e saída, enriquecendo a análise do comportamento do software.

2.1.3 *Ferramentas e técnicas*

Na prática, diferentes ferramentas e técnicas podem ser utilizadas para atingir os objetivos da análise. As abordagens variam conforme o alvo e o tipo de informação que se deseja obter. Entre as principais, destacam-se:

2.1.3.1 *Desmontagem de Binários (Disassembly)*

Essa técnica envolve converter o código de máquina (binário) de um software em uma representação legível, geralmente utilizando a linguagem *assembly*. Ferramentas como **IDA Pro** (HEX-RAYS, 1996) e **Ghidra** (NATIONAL SECURITY AGENCY, 2019) são amplamente utilizadas para essa tarefa, permitindo ao desenvolvedor inspecionar o fluxo de execução do software, descobrir funções, rotinas e identificar interações entre módulos.

2.1.3.2 *Depuração (Debugging)*

A depuração é o processo de executar um software de forma controlada para monitorar o comportamento de variáveis, memória e registradores. Ferramentas como **OllyDbg** (OLLYDBG, 2004) e **x64dbg** (x64dbg Team, 2015) permitem que o desenvolvedor inspecione a execução

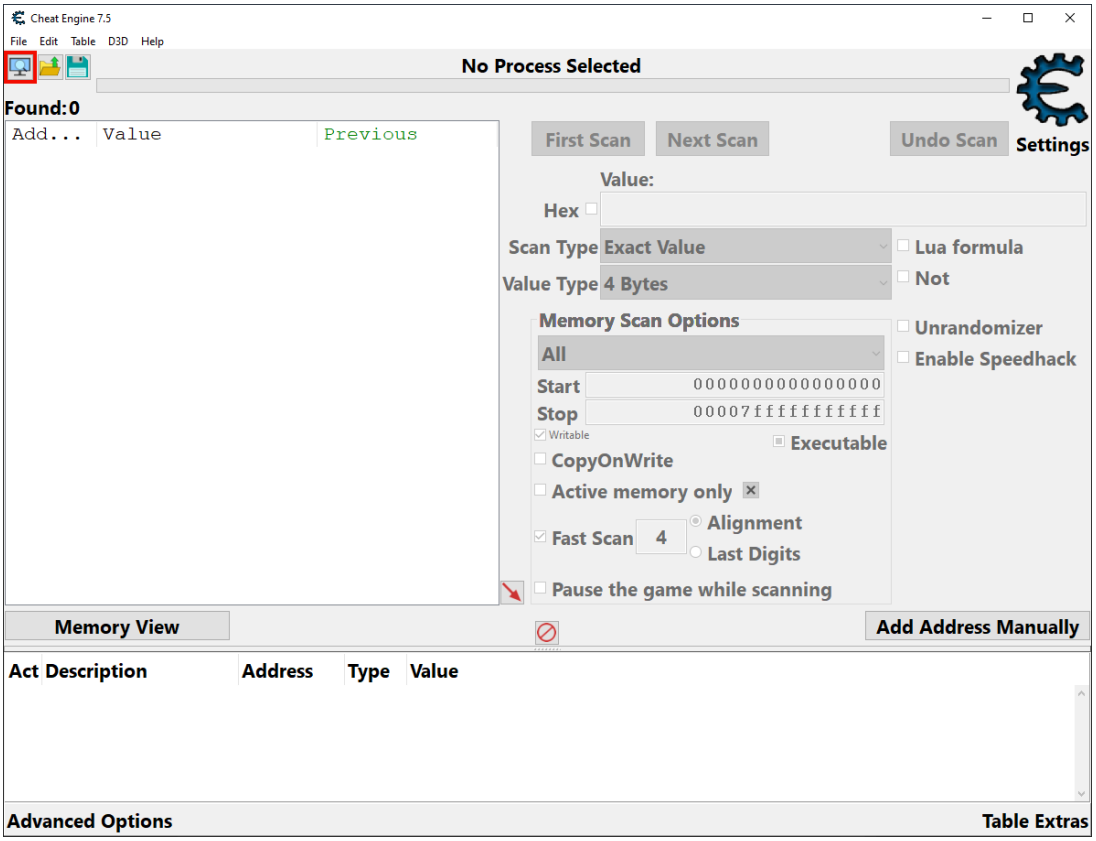
do software passo a passo, possibilitando a identificação de variáveis e funções críticas, como funções de validação de licenças ou de verificações de integridade.

2.1.3.3 *Cheat Engine*

O **Cheat Engine** (HEIJNEN, 2000) é uma ferramenta de código aberto, criada no início dos anos 2000, amplamente utilizada na análise dinâmica de jogos e programas. Seu principal objetivo é permitir a varredura e manipulação de regiões de memória de processos em execução, sendo especialmente útil na identificação de variáveis em tempo real, como vida, munição e posição.

Além dessa funcionalidade central, o software se destaca por oferecer uma série de recursos avançados que facilitam o processo de engenharia reversa, especialmente em ambientes interativos como jogos. A integração de diferentes módulos, como depuração e desmontagem de binários, permite uma análise profunda do comportamento do programa durante sua execução.

Figura 1 – Tela inicial do Cheat Engine.



Fonte: Elaborado pelo autor, 2025.

A Figura 1 apresenta a interface principal do software, onde é possível selecionar o processo alvo, configurar os tipos de busca e visualizar os valores monitorados durante a execução do programa.

Tabela 1 – Funcionalidades principais do Cheat Engine.

Funcionalidade	**Descrição**
Escaneamento de memória	Busca por valores específicos na RAM durante a execução de um processo
Depurador (Debugger)	Permite executar o programa passo a passo, inserir breakpoints e inspecionar registradores
Desmontador de binários (Disassembler)	Exibe o código em linguagem assembly correspondente ao binário do processo
Injetor de código	Permite modificar ou adicionar instruções diretamente na memória do processo
Escaneamento com múltiplos filtros	Suporta buscas refinadas com base em tipos, variações e condições
Controle de velocidade	Acelera ou desacelera a execução do processo alvo para análise ou manipulação
Visualização de ponteiros	Identifica e resolve cadeias de ponteiros para acesso a variáveis dinâmicas

Fonte: Elaborado pelo autor, 2025.

A Tabela 1 resume as principais funcionalidades oferecidas pelo Cheat Engine, destacando suas aplicações no contexto da análise e manipulação de memória.

2.2 Análise e Manipulação de memória e Engenharia Reversa

A análise e manipulação da memória de um software consiste no acesso e na alteração de dados armazenados na memória de acesso aleatório (Random Access Memory - RAM) durante sua execução. Essa prática é essencial na engenharia reversa, pois permite compreender o funcionamento interno de um software, identificar dados críticos e modificar comportamentos. Conforme destacado por Rolim (2023 apud MERCES, 1981, p. 22):

> "Entende-se por análise de memória a capacidade do hacker de entender os dados do jogo que estão sendo armazenados na memória do computador e ao que esses dados correspondem na prática. Engenharia reversa é uma prática de desconstrução de um software, de forma que a pessoa que a realiza tem a capacidade de compreender como o mesmo foi construído. A integração desses dois conhecimentos, geralmente aplicada por hackers ou pesquisadores de segurança, torna possível as manipulações mais complexas e eficientes." (ROLIM, 2023 apud MERCES, 1981, p. 22).

2.2.1 Estratégias de Manipulação de Memória

2.2.1.1 *Manipulação Por Meio de APIs do Sistema Operacional*

Os sistemas operacionais modernos, como o Windows fornecem APIs (*Application Programming Interface*) para acesso à memória de processos. Segundo Tanenbaum (2010, p.

647), a API do Windows (Win32) oferece um conjunto de funções que permitem a um processo gerenciar diretamente sua memória, e, em certos casos, acessar a memória de outros processos, desde que possua as permissões apropriadas.

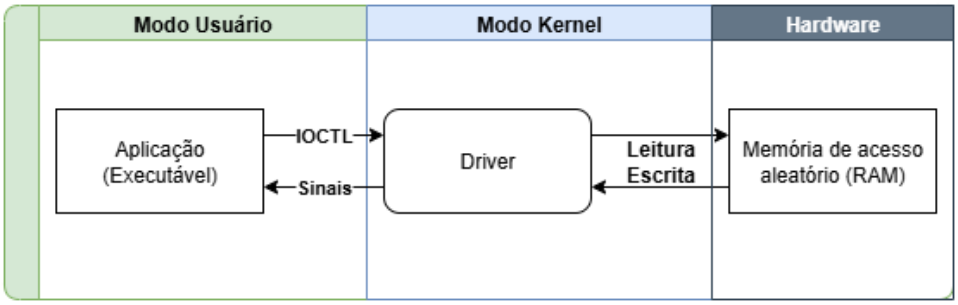
Apesar da flexibilidade oferecida, o uso dessas APIs implica em chamadas de sistema, o que torna sua utilização mais suscetível à detecção por mecanismos de segurança, como os implementados pelos sistemas anti-trapaça. Essa característica representa uma limitação importante em cenários que demandam maior furtividade.

2.2.1.2 *Manipulação por meio de Driver em Modo Kernel*

Segundo Korkin (2018), drivers operando em modo kernel são capazes de controlar o acesso à memória física e executar tarefas críticas com isolamento entre diferentes espaços de execução, evidenciando seu potencial tanto para segurança quanto para exploração de sistemas.

A manipulação de memória por meio de drivers em modo kernel é uma técnica avançada empregada em contextos nos quais o acesso à memória de outros processos é restrito, como em jogos protegidos por mecanismos anti-cheat. Essa abordagem baseia-se na criação de um driver que opera com privilégios elevados, possibilitando a leitura e escrita em regiões da memória física, independentemente das permissões do sistema operacional em modo usuário. Essa técnica requer conhecimento aprofundado sobre o desenvolvimento de drivers, funcionamento do kernel e segurança em nível de hardware.

Figura 2 – Modelo de comunicação entre modo usuário e modo kernel via IOCTL.



Fonte: Elaborado pelo autor, 2025.

Para que o software em modo usuário possa interagir com o driver, utiliza-se um modelo conhecido como comunicação entre modo usuário e modo kernel (User Mode/Kernel Mode Communication – UM/KM). Nesse modelo, a aplicação em modo usuário se comunica com o driver utilizando canais como a API *DeviceIoControl*, enviando comandos e parâmetros estruturados (via códigos IOCTL), os quais são interpretados pelo driver para executar operações privilegiadas diretamente na RAM, conforme ilustrado na Figura 2.

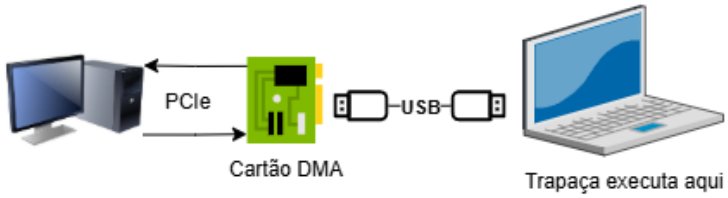
Vale destacar que, embora essa técnica represente uma abordagem avançada e relevante em contextos com sistemas de proteção mais robustos, ela **não foi aplicada neste trabalho**.

Sua inclusão no referencial teórico tem como objetivo apenas apresentar alternativas técnicas comumente utilizadas na área, podendo ser explorada como proposta de extensão em pesquisas futuras.

2.2.1.3 *Manipulação com Direct Memory Access (DMA)*

O acesso direto à memória (*Direct Memory Access – DMA*) é uma técnica que permite a leitura e escrita de dados diretamente na memória física, sem a intervenção do processador. Segundo Tanenbaum (2010, p. 238), isso é possível graças a dispositivos de hardware dedicados, denominados controladores de DMA, que possuem acesso direto ao barramento do sistema e realizam transferências de dados de forma autônoma em relação à CPU.

Figura 3 – Arquitetura simplificada de acesso à memória via DMA com dois computadores.



Fonte: Elaborado pelo autor, 2025.

No contexto de jogos e manipulação de memória, o DMA é utilizado em ataques mais sofisticados, geralmente por meio de dispositivos externos, como placas FPGA ou cartões PCI express. Essa abordagem é particularmente difícil de ser detectada, pois o acesso ocorre fora do controle do sistema operacional. O uso de DMA na engenharia reversa é comum para evitar a proteção de software contra manipulações, como mecanismos de anti-cheat que monitoram acessos de memória padrão. Porém, um ponto negativo dessa técnica é que os dispositivos são de alto custo e geralmente é preciso um segundo computador que irá atuar como o agente analisador e modificador da memória, conforme ilustrado na Figura 3.

2.3 Trapaças

2.3.1 *O que são cheats?*

Segundo Vicentine *et al.* (2022), o termo *Cheat* é uma linguagem própria de jogadores para denominar trapaças, código e truques que servem para alterar a experiência normal de um jogo. Existem jogos que já foram projetados de fábrica com modos de trapaça, como foram os casos dos jogos ***Grand Theft Auto: San Andreas (2004)***, ***God Of War (2005)*** e ***The Sims 2 (2004)***, em que bastava apenas realizar uma sequência de botões ou código para ativar essas trapaças.

Figura 4 – Trapaça que permite visualizar os inimigos através da parede no Jogo Counter-Strike 2.

Fonte: BLIX.GG, 2023

No entanto, existem as trapaças que não são autorizadas pelos jogos, como a apresentada na Figura 4 que ilustra uma trapaça que permite visualizar os inimigos através das paredes no jogo **Counter-Strike 2 (2023)**. Trapaças não são permitidas principalmente em jogos competitivos ou online, pois representam uma ameaça à integridade do jogo. Elas podem prejudicar a estabilidade e a justiça das competições, além de gerar frustração entre os jogadores honestos. Como aponta Kaspersky (2019):

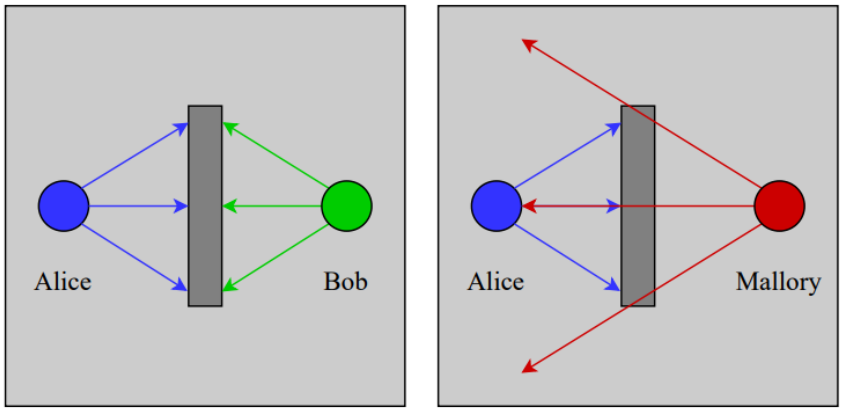
> "Aqueles que violam as regras têm potencial para impedir que jogadores honestos ou profissionais participem de competições, afetando os relacionamentos na comunidade dos jogos. Além disso, essas violações podem ter impacto sobre a popularidade e a reputação de um torneio e, eventualmente, reduzir a receita ou o número de possíveis parcerias para os organizadores." (KASPERSKY, 2019).

Ou seja, as consequências não se limitam apenas à experiência dos jogadores, mas também ao ecossistema em torno dos jogos digitais, impactando os desenvolvedores, organizadores de eventos e até a indústria de esportes eletrônicos como um todo.

2.3.2 *Tipos de Trapaças em Jogo*

As trapaças em jogos digitais podem ser categorizadas com base no método utilizado para explorar vulnerabilidades ou modificar o comportamento do jogo. A natureza e a popularidade de cada tipo de trapaça variam conforme o gênero do jogo, suas mecânicas internas e a presença (ou não) de elementos competitivos.

Figura 5 – Jogando limpo e jogando com wallhack.



Fonte: RENDENBACH, 2022.

Em jogos de tiro em primeira pessoa (*First-Person Shooters* - FPS), são comuns trapaças como o *wallhack* ou ESP (Extra Sensory Perception), exemplificado nas Figuras 4 e 5. Esse tipo de técnica permite visualizar inimigos ou itens através de paredes, alterando o processo de renderização ou sobrepondo elementos gráficos à tela.dor veja inimigos ou itens através de paredes (*wallhack* ou ESP).

Outra trapaça frequente em FPS é o *aimbot*, que automatiza a mira do jogador, oferecendo vantagem em combates. Existem também, trapaças de velocidade que manipulam parâmetros da física do jogo, acelerando ou desacelerando o movimento do personagem.

Em jogos de estratégia em tempo real, é comum o uso de trapaças que revelam o mapa completo, ignorando o conceito de "nevoeiro de guerra" (*fog of war*). Já em jogos baseados em economia ou progressão, como MMORPGs (*Massive Multiplayer Online Role-Playing Game*) ou jogos com microtransações, há manipulações que alteram valores de moedas virtuais, inventários ou atributos de personagens.

Vale destacar que **nem toda trapaça é aplicável a qualquer tipo de jogo**: cada uma depende das vulnerabilidades específicas da aplicação, do acesso à memória e da forma como os dados são processados localmente ou em servidores remotos.

2.3.3 *Sistemas Anti-Trapaça*

Os sistemas anti-trapaça surgiram como uma resposta direta à proliferação de trapaças, principalmente em jogos multi-jogadores competitivos, onde o comportamento injusto é impactante. Os métodos de funcionamento dos anti-cheats variam em sofisticação e abordagem, mas geralmente podem ser classificados em três categorias principais:

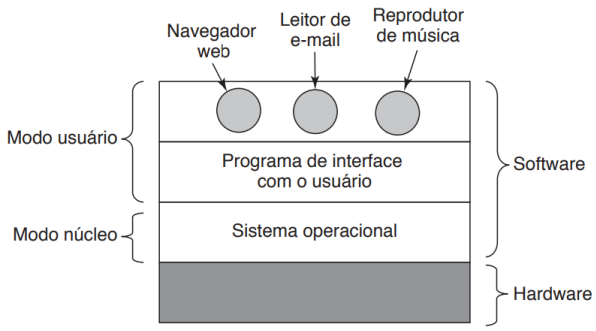
- **Baseados em assinaturas:** detectam trapaças conhecidas comparando assinaturas de código ou comportamentos com uma base de dados preexistente.

- **Monitoramento em tempo real:** analisam o comportamento do jogador durante as partidas, buscando padrões atípicos, como precisão sobre-humana ou movimentos impossíveis.
- **Controle de integridade:** verificam se os arquivos do jogo ou da memória foram modificados durante a execução, muitas vezes por meio de técnicas de hash ou varredura de memória.

2.3.4 *Níveis de Operação: Usuário vs. Núcleo*

Segundo Tanenbaum (2010, p. 1), a maioria dos computadores possuem dois modos de operação, sendo eles o modo usuário e o modo núcleo (comumente conhecido como modo *kernel*).

Figura 6 – Modo usuário vs. Modo núcleo.



Fonte: Sistemas Operacionais Modernos, TANENBAUM, 2010.

Conforme apresentado na Figura 6 o próprio sistema operacional atua em modo núcleo, junto a softwares para fazer a interface com o hardware, como os drivers de dispositivos. Essa mesma categoria pode ser aplicada tanto aos sistemas anti-trapaça como as próprias trapaças em si, que podem operar em um dos modos:

- **Modo usuário (*User Mode*):** Nesse nível, o software possui permissões limitadas e opera no mesmo espaço que os aplicativos comuns do usuário. Algumas de suas vantagens é que o software tem um impacto menor no sistema já que possui acesso restrito ao sistema, e diminui o riscos de vulnerabilidades graves. E geralmente é mais fácil de implementar e depurar.
- **Modo núcleo (*Kernel Mode*):** Operar no modo núcleo significa que o software tem acesso privilegiado ao sistema operacional e ao hardware, permite maior controle e capacidade de monitorar o que acontece em todo o sistema. Uma das vantagens de um anti-cheat operando neste modo é que é mais difícil de contornar esse tipo de proteção, pois o software tem acesso completo ao sistema e pode detectar alterações em áreas críticas, e é capaz de interceptar e prevenir manipulações no núcleo do sistema ou no hardware.

2.3.5 *Controvérsias sobre uso do modo núcleo*

Softwares que operam em modo núcleo geram muitas controvérsias devido ao alto nível de permissões, que concede acesso total ao hardware e à memória do sistema operacional. Embora sistemas anti-trapaças que operam nesse nível sejam necessários para combater trapaças sofisticadas, essas implementações podem gerar preocupações significativas quanto à privacidade e segurança dos usuários.

Um exemplo emblemático é o sistema anti-trapaças do jogo *Genshin Impact* (miHoYo, 2020), cujo driver *mhyprot2.sys* apresentou vulnerabilidades críticas (CVE Program, 2020), permitindo execução arbitrária de código e sendo exploradas por *ransomwares* para desativar antivírus (TREND MICRO RESEARCH, 2022). Este acontecimento ilustra o risco desses sistemas: além de exporem usuários a falhas, podem ser explorados por agentes maliciosos.

Além disso, como esses softwares operam diretamente no espaço do modo núcleo, erros e vulnerabilidades podem comprometer diretamente a estabilidade e segurança do sistema operacional. Essa abordagem, embora eficaz contra trapaças, exige um alto grau de responsabilidade ética por parte das empresas que administram esses softwares anti-trapaças, dado a sensibilidade dos dados acessados e o impacto potencial em dispositivos dos jogadores (ROLIM, 2023).

3 METODOLOGIA

Este capítulo apresenta a metodologia adotada para a realização deste estudo, detalhando os procedimentos e ferramentas utilizadas para alcançar os objetivos propostos. O estudo foi conduzido em etapas sequenciais, iniciando-se com a análise de memória do jogo Assault Cube, seguida pelo desenvolvimento de uma prova de conceito para a manipulação dessas informações e, por fim, a validação da solução implementada.

3.1 O Jogo Assault Cube

O jogo Assault Cube (Figura 7) foi escolhido como objeto de estudo devido à sua natureza de código aberto, permitindo a aplicação de técnicas de engenharia reversa sem restrições legais. Sendo um jogo de tiro em primeira pessoa (FPS - *First Person Shooter*), Assault Cube possui diversas variáveis dinâmicas, como pontos de vida, munição e posição dos jogadores, o que o torna um ambiente ideal para experimentação de manipulação de memória. Além disso, por ser um jogo leve e acessível, ele permite testes eficientes sem consumo excessivo de recursos computacionais, facilitando a análise e a implementação das técnicas propostas.

Figura 7 – Perspectiva de um jogador no Assault Cube.

Fonte: Elaborado pelo autor, 2025.

3.2 Estrutura da Metodologia

O estudo foi desenvolvido seguindo nas seguintes etapas principais:

- **Análise dinâmica e mapeamento de endereços de memória:** inspeção da memória do jogo em tempo de execução para identificação de endereços de memória das propriedades relevantes para o jogador, como pontos de vida, munição e posição dos jogadores inimigos.
- **Implementação da prova de conceito:** desenvolvimento de uma ferramenta em C# e .NET Framework, utilizando-se do Windows Forms para a criação de uma interface gráfica, e da biblioteca *Memory.dll* para manipulação de memória do jogo Assault Cube em tempo de execução.

3.3 Análise Dinâmica e Mapeamento dos Endereços de Memória

Nesta etapa, foi realizada uma análise dinâmica do jogo Assault Cube com o objetivo de identificar e mapear os endereços de memória relevantes para a manipulação. Essa análise permitiu localizar valores relacionados às variáveis de interesse, como pontos de vida, munição e posição do jogador, utilizando o software Cheat Engine. [1]

Tabela 2 – Softwares utilizados na análise dinâmica

Software	**Descrição**	**Função na Análise**
Assault Cube	Jogo de tiro em primeira pessoa, de código aberto.	Ambiente alvo da engenharia reversa e manipulação de memória.
Cheat Engine	Ferramenta de código aberto para escaneamento e edição de memória em tempo real.	Localização e teste de endereços das variáveis do jogo.

Fonte: Elaborado pelo autor, 2025.

A Tabela 2 resume os principais softwares utilizados durante essa etapa, destacando suas funções no processo de engenharia reversa e manipulação de memória. É importante destacar, que o software Cheat Engine será operado no modo usuário e no sistema operacional Windows 10.

3.4 Implementação da Prova de Conceito

O desenvolvimento da prova de conceito foi realizado com base nas informações obtidas na etapa de análise e identificação dos endereços de memória. A implementação foi desenvolvida em linguagem C#, utilizando o Framework .NET, por ser compatível com aplicações nativas do sistema Windows. Para facilitar o acesso à memória de outros processos em tempo de execução, foi empregada a biblioteca *Memory.dll*[2], que fornece abstrações da API do sistema operacional. O ambiente de desenvolvimento utilizado foi o Visual Studio 2022.

[1] Optou-se por não realizar análise estática, pois a análise dinâmica foi suficiente para atingir os objetivos do estudo, considerando que o jogo pode ser executado livremente e não possui mecanismos de proteção.

Tabela 3 – Recursos utilizadas na implementação da prova de conceito

Recurso	Descrição	Função na Implementação
C#	Linguagem de programação orientada a objetos desenvolvida pela Microsoft.	Implementação da aplicação de manipulação de memória.
.NET Framework 4.7.2	Plataforma de desenvolvimento para aplicações Windows.	Suporte à execução e integração com APIs do sistema.
Memory.dll	Biblioteca C# para acesso simplificado à memória de processos.	Abstração das chamadas da API do Windows para leitura e escrita na memória de outro processo.
Visual Studio 2022	Ambiente integrado de desenvolvimento (IDE) da Microsoft.	Utilizado para codificação, depuração e compilação da aplicação.

Fonte: Elaborado pelo autor, 2025.

A Tabela 3 apresenta os principais recursos utilizados durante a implementação.

3.4.1 *Requisitos da prova de conceito*

O software desenvolvido para manipular a memória do jogo Assault Cube deverá atender aos seguintes requisitos que fornecem vantagens ao jogador:

1. Permitir a alteração de propriedades do jogador, como pontos de vida e quantidade de munição;
2. Exibir a localização de inimigos através das paredes e objetos no jogo (ESP);
3. Fornecer um assistente de mira que posicione automaticamente a mira nos inimigos (aimbot);
4. Ser intuitivo e de fácil utilização, permitindo que o usuário ative e desative as trapaças com atalhos do teclado.

Cada uma dessas funcionalidades será demonstrada com exemplos visuais e descrições detalhadas na seção de resultados, a fim de evidenciar sua implementação prática e funcionamento no ambiente de jogo.

3.4.2 *Estratégia de Implementação*

O desenvolvimento da prova de conceito será implementado nas seguintes fases principais:

- **Configuração do ambiente de desenvolvimento:**
  Instalação e configuração das ferramentas necessárias no sistema operacional Windows 10,

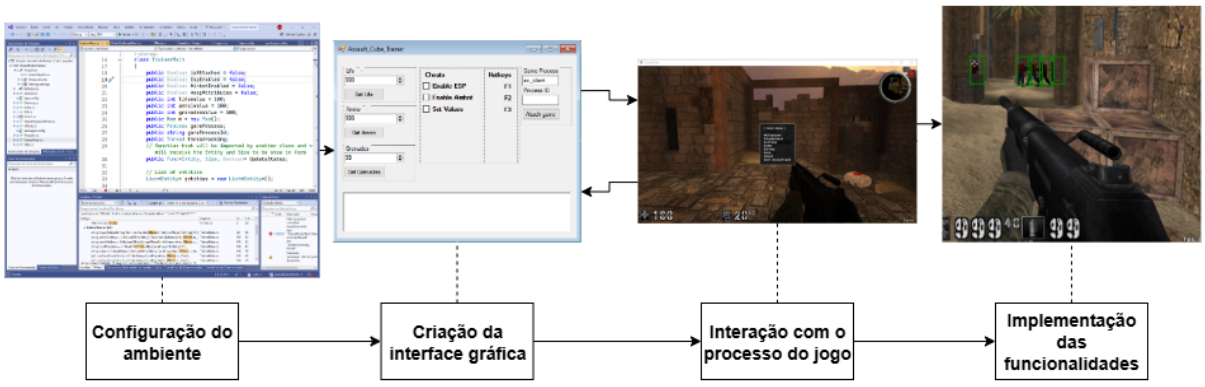
incluindo um ambiente de desenvolvimento integrado (IDE) como o Visual Studio 2022, além de bibliotecas específicas para manipulação de memória, como a Memory.dll[2] que fornece funções simplificadas para interagir com a Windows API, que por debaixo dos panos utiliza funções como *OpenProcess*, *ReadProcessMemory* e *WriteProcessMemory*, para realizar a leitura e escrita de memória de forma segura e eficaz.

- **Criação da Interface gráfica:**
  Será desenvolvida a interface do software utilizando-se do Windows Forms que é fornecido nativamente pelo próprio framework .NET. Esta interface deve conter botões para conectar o aplicativo ao processo do jogo, e botões para ativar e desativar as trapaças.

- **Interação com o processo e manipulação da memória:**
  Desenvolvimento da lógica para localizar o processo do jogo em execução, obter acesso às suas regiões de memória e implementar os métodos responsáveis pela leitura e escrita de variáveis como vida, munição e posição. Essa etapa foi realizada com o auxílio da biblioteca *Memory.dll*, que simplifica o uso das APIs nativas do sistema operacional.

- **Implementação das funcionalidades:**
  Aplicação prática das técnicas desenvolvidas para criação das funcionalidades da prova de conceito, como ESP, aimbot e manipulação de atributos, com base nos endereços mapeados e nos métodos de leitura e escrita implementados.

Figura 8 – Fluxo de desenvolvimento da prova de conceito.



Fonte: Elaborado pelo autor, 2025.

A Figura 8 ilustra o fluxo de desenvolvimento da prova de conceito. O processo teve início com a configuração do ambiente de desenvolvimento, seguido pela criação da interface gráfica com Windows Forms. Em seguida, foi implementada a interação com o processo do jogo e a manipulação das variáveis de memória. Por fim, as funcionalidades principais da trapaça foram desenvolvidas com base nos endereços previamente mapeados.

[2] A biblioteca Memory.dll foi instalada pelo gerenciador de pacotes NuGET do Visual Studio, mas pode ser encontrada no repositório do GitHub: <https://github.com/erfg12/memory.dll>.

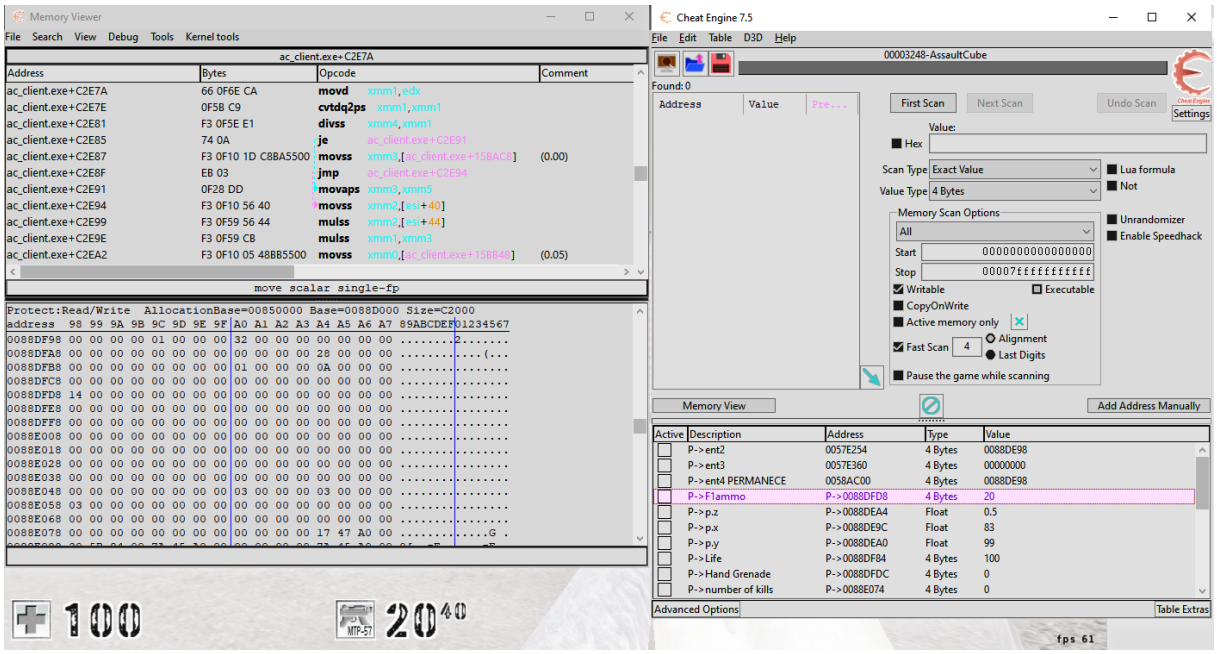
4 RESULTADOS E DISCUSSÃO

Este capítulo apresenta os resultados obtidos durante a execução das etapas descritas na metodologia, incluindo a engenharia reversa por meio da análise de memória do jogo e a implementação da prova de conceito.

4.1 Análise Dinâmica e Mapeamento dos Endereços de Memória

Para a inspeção e modificação dos valores armazenados na memória do jogo, foi utilizado o software Cheat Engine. Embora essa ferramenta possa operar com suporte a drivers em modo kernel, neste trabalho ela foi utilizada exclusivamente em modo usuário, fazendo uso das APIs nativas do Windows. Essa abordagem permitiu realizar buscas em tempo real por variáveis como pontos de vida, munição e coordenadas dos jogadores, facilitando a identificação das estruturas manipuladas pelo jogo.

Figura 9 – Análise de memória do jogo com o Cheat Engine.



Fonte: Elaborado pelo autor, 2025.

A Figura 9 ilustra o software Cheat Engine em operação durante a execução do jogo Assault Cube, no contexto da análise dinâmica de memória. No canto inferior direito da interface, é possível observar diversos endereços de memória mapeados, juntamente com os respectivos valores armazenados. Entre as variáveis identificadas estão as coordenadas tridimensionais do jogador (X, Y, Z), pontos de vida, quantidade de granadas, entre outras informações relevantes para a manipulação do estado do jogo.

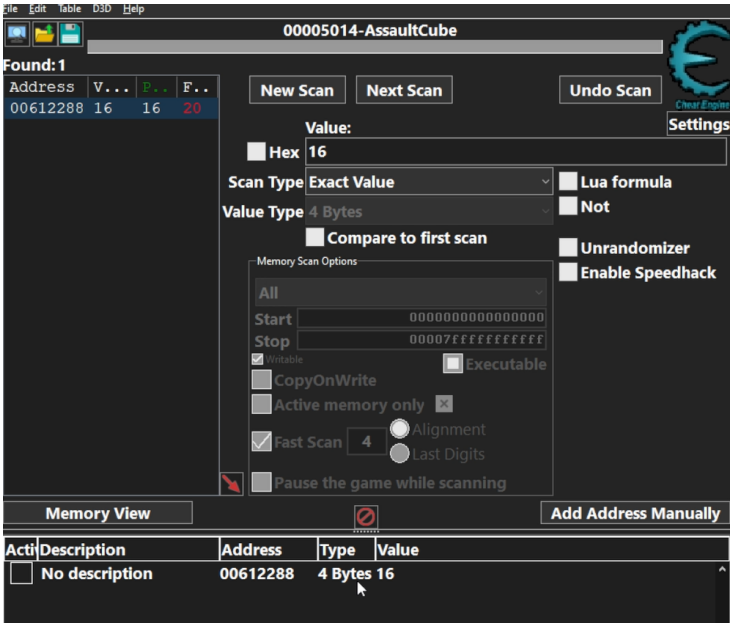
Além disso, no lado esquerdo da interface, destaca-se outra funcionalidade fundamental da ferramenta: o desmontador de binários (*disassembler*). Essa visualização permite inspecionar

diretamente as instruções de máquina em linguagem de *assembly*, oferecendo uma visão detalhada do comportamento do código em tempo de execução. Essas funcionalidades facilitam a identificação de padrões, estruturas e pontos estratégicos para modificação de memória.

4.1.1 *Busca de Endereços*

Uma das principais funcionalidades do Cheat Engine é a busca por endereços de memória com valores específicos. Para isso, o usuário insere um valor na ferramenta, que realiza uma varredura em todos os endereços de memória do processo alvo. Como diversos endereços podem armazenar o mesmo valor, é necessário refinar a busca modificando esse valor dentro do jogo e repetindo o processo até identificar o endereço correto.

Figura 10 – Resultado de uma busca refinada por valores inteiros no Cheat Engine.



Fonte: Elaborado pelo autor, 2025.

A Figura 10 apresenta o resultado de uma busca refinada por endereços de memória associados à variável de munição do jogador. Inicialmente, o valor buscado foi 20, correspondente à quantidade exibida na interface do jogo. Após disparos, o valor foi reduzido para 16 e uma nova busca foi realizada, restringindo os resultados anteriores. O *Cheat Engine* evidencia esse processo ao exibir, lado a lado, os valores atual (16) e anterior (20), ilustrando a técnica de filtragem iterativa por variação controlada em tempo real, que permite reduzir progressivamente os endereços candidatos até identificar o endereço desejado.

4.1.2 *Ponteiros*

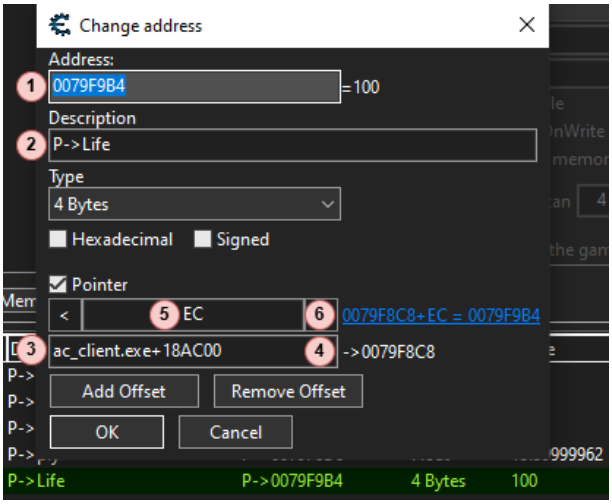
A cada reinicialização do jogo, o sistema operacional pode alocar o processo em uma região diferente da memória, alterando dinamicamente os endereços base das variáveis utiliza-

das internamente. Isso ocorre por causa de técnicas modernas de gerenciamento de memória, como o ASLR (*Address Space Layout Randomization*), que visam dificultar a exploração de vulnerabilidades.

Diante dessa característica, os endereços de memória identificados manualmente em uma execução do jogo tornam-se obsoletos em execuções futuras. Para contornar essa limitação, utiliza-se uma técnica de ponteiros, que consiste em localizar a variável desejada por meio de uma cadeia de referências. Essa cadeia geralmente parte de um endereço estático associado ao módulo principal do processo — como o executável do jogo ou bibliotecas dinâmicas (.dll) — e utiliza offsets fixos para acessar estruturas internas do programa.

O *Cheat Engine* fornece uma funcionalidade chamada escaneamento de ponteiros, capaz de identificar automaticamente todas as possíveis cadeias entre endereços estáticos e variáveis dinâmicas. Assim como na busca por valores simples, esse processo exige refinamento: são realizadas múltiplas execuções do jogo com verificação dos endereços mapeados, a fim de identificar uma cadeia persistente. Quando localizada, essa cadeia de ponteiros permite acessar a variável desejada de forma confiável, desde que a versão do executável analisado permaneça inalterada. Caso o jogo seja recompilado ou atualizado, é necessário refazer o mapeamento.

Figura 11 – Exemplo de Cadeia de Ponteiros para Acesso à Variável de Vida.



Fonte: Elaborado pelo autor, 2025.

A Figura 11 ilustra um exemplo de uma cadeia de ponteiros para acesso à variável de vida no *Cheat Engine*, evidenciando o processo de resolução de um endereço dinâmico por meio da combinação entre módulo base, offsets e ponteiros. Os principais elementos destacados na imagem são:

1. Endereço de memória final (campo superior), onde está armazenado o valor da variável monitorada;

2. Descrição atribuída ao valor, indicando que se trata dos pontos de vida do jogador;

3. Módulo base do executável (ac_client.exe) combinado com um offset constante (+18AC00), utilizado para localizar a estrutura principal do jogador;

4. Endereço intermediário resolvido, correspondente à base da estrutura do jogador na memória (0079F8C8);

5. Offset adicional (+EC), necessário para acessar especificamente o campo correspondente aos pontos de vida dentro da estrutura do jogador;

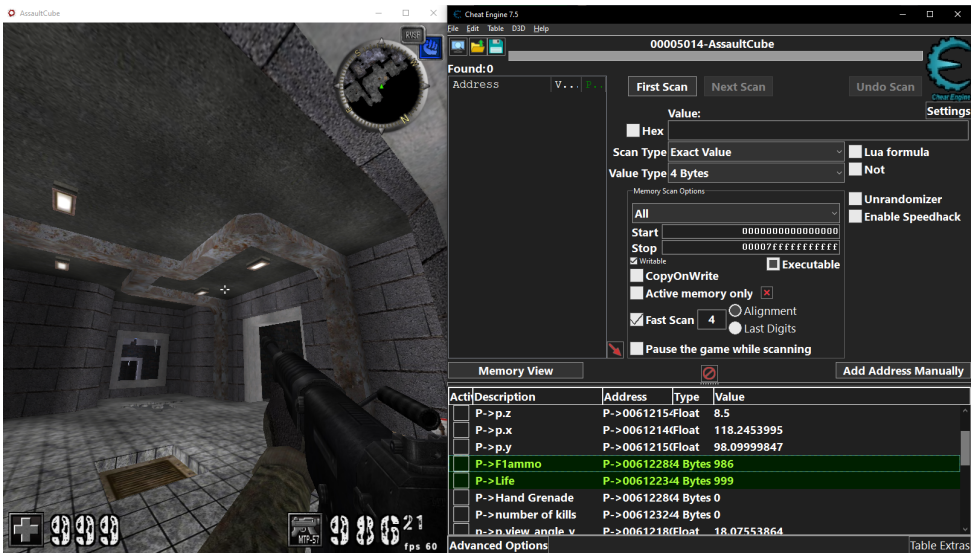
6. Endereço de memória final dos pontos de vida (0079F9B4), resultado da soma do endereço base da estrutura com o offset aplicado.

Esse encadeamento de ponteiros é fundamental para acessar variáveis cuja localização exata muda a cada execução do programa, garantindo um acesso estável mesmo em ambientes de memória dinâmica.

4.1.3 *Modificação de Valores*

Para validação de alguns endereços de memória obtidos, foi feita a modificação desses valores. A Figura 12 ilustra o processo de alteração dos valores da memória, destacando as modificações aplicadas ao longo da execução do jogo por meio do Software Cheat Engine.

Figura 12 – Modificação de valores da memória em tempo real com o cheat engine.



Fonte: Elaborado pelo autor, 2025.

Na Figura 12, observa-se o jogo Assault Cube em execução, à esquerda, e a interface do software Cheat Engine à direita, ambos utilizados para demonstração prática de manipulação de memória. A imagem destaca a alteração de valores em tempo real, como vida ("P->Life") e munição ("P->Flammo"), que foram alterados respectivamente para 999 e 986 por meio da

substituição dos dados armazenados nos respectivos endereços de memória. Esses valores, destacados em verde na interface do Cheat Engine, indicam que a manipulação foi bem-sucedida.

4.1.4 *Endereços Mapeados*

Durante a análise dinâmica da memória, foram identificados e mapeados diversos endereços de memória relevantes para a implementação da prova de conceito. Esses endereços permitem o acesso direto a variáveis críticas do jogo, como posição, vida, munição e outros atributos do jogador. A Tabela 4 apresenta os principais offsets e seus respectivos significados.

Tabela 4 – Endereços e offsets mapeados na memória do Assault Cube.

Descrição	**Endereço / Offset**
Processo do jogo	`ac_client.exe`
Endereço base do jogador local	`ac_client.exe + 0x18AC00`
Lista de entidades (jogadores)	`ac_client.exe + 0x18AC04`
Matriz Model-View-Projection	`ac_client.exe + 0x17DFD0`
Posição X do jogador	`Endereço base do jogador + 0x04`
Posição Y do jogador	`Endereço base do jogador + 0x08`
Posição Z do jogador	`Endereço base do jogador + 0x0C`
Ângulo de visão horizontal	`Endereço base do jogador + 0x34`
Ângulo de visão vertical	`Endereço base do jogador + 0x38`
Campo de visão (FOV)	`Endereço base do jogador + 0x334`
Campo de visão com mira (Scoped FOV)	`Endereço base do jogador + 0x338`
Pontos de vida (Health)	`Endereço base do jogador + 0xEC`
Armadura	`Endereço base do jogador + 0xF0`
Munição	`Endereço base do jogador + 0x140`
Granadas	`Endereço base do jogador + 0x144`
Número de abates (Kills)	`Endereço base do jogador + 0x1DC`
Nome do jogador	`Endereço base do jogador + 0x205`
Equipe (Time)	`Endereço base do jogador + 0x30C`

Fonte: Elaborado pelo autor, 2025.

Nos offsets listados na Tabela 4, destaca-se a estrutura da chamada lista de entidades (*Entity List*), representada pelo endereço base `ac_client.exe + 0x18AC04`. Essa lista funciona como um vetor de ponteiros, onde cada elemento aponta para a estrutura de dados correspondente a um jogador presente na partida.

Cada ponteiro dentro desse vetor referencia uma instância de estrutura de jogador, contendo informações relevantes como posição tridimensional (X, Y, Z), pontos de vida, nome, time, quantidade de munição, entre outros atributos. Essa organização possibilita o acesso direto às variáveis de múltiplos jogadores, permitindo, por exemplo, o monitoramento de todos os oponentes simultaneamente, que é algo essencial para a implementação de funcionalidades como a que permite visualizar os inimigos através das paredes e o assistente de mira.

O conhecimento dessa estrutura vetorial foi fundamental para o desenvolvimento da prova de conceito, pois permitiu percorrer e extrair informações de cada jogador ativo na memória do jogo.

Os Endereços mapeados na Tabela 4 foram essenciais para viabilizar o funcionamento da prova de conceito, pois permitiram a leitura e a escrita das variáveis durante a execução do jogo. Para acessar corretamente essas variáveis, utilizou-se o endereço base do processo do jogo somado ao offset de cada campo.

4.2 Implementação da Prova de Conceito

Após a execução da análise de memória e a coleta de diversos endereços de memória relevantes para o jogador, foi desenvolvido um software em C# capaz de modificar variáveis do jogo Assault Cube em tempo de execução. A ferramenta desenvolvida utiliza a biblioteca Memory.DLL para leitura e escrita direta na memória.

4.2.1 *Funcionalidades*

As funcionalidades da prova de conceito desenvolvida baseiam-se na leitura e/ou manipulação dos endereços de memória (previamente mapeados na Tabela 4) para garantir vantagens ao jogador.

4.2.1.1 *Modificação de Atributos*

Uma das funcionalidades implementadas permite modificar diretamente atributos do jogador, como pontos de vida, munição e número de granadas. Foi incluído um mecanismo que atualiza esses atributos a cada iteração do software, garantindo que os valores permaneçam constantes mesmo após ações que normalmente os reduziriam. Essas manipulações foram possíveis por meio da escrita direta nos endereços de memória correspondentes, garantindo que os valores alterados fossem refletidos em tempo real no jogo.

4.2.1.2 *Transformação de Coordenadas do Mundo para a Tela*

A determinação de quais objetos são visíveis ao jogador e como devem ser exibidos na tela é feita com base na matriz Model-View-Projection (MVP), que é uma matriz 4×4 que representa a posição e orientação da câmera (jogador) em relação ao mundo do jogo. Ela é fundamental para transformar coordenadas do espaço tridimensional do mundo (*World Space*) para coordenadas relativas à perspectiva da câmera em espaço de tela (*Screen Space*).

Nos motores gráficos modernos, como o OpenGL, utilizado pelo jogo Assault Cube, a renderização envolve a multiplicação de três matrizes fundamentais: modelo (Model), visão (View) e projeção (Projection), resultando na matriz composta da Equação 4.1 conhecida como Model-View-Projection (MVP), conforme descrito por Vries (2020) e OpenGLTutorial (2021).

$$MVP = \text{Modelo} \times \text{Visão} \times \text{Projeção} \tag{4.1}$$

No contexto deste trabalho, foi extraída diretamente da memória do jogo a matriz composta Model-View-Projection (MVP). Nos jogos digitais, é comum que o motor gráfico armazene diretamente a MVP para reduzir custo computacional.

A obtenção empírica da matriz MVP foi realizada utilizando o Cheat Engine, através da análise dos valores da rotação vertical da câmera, variando entre $+90^\circ$ (olhando para cima) e -90° (olhando para baixo). Esses ângulos extremos resultam em valores próximos de $+1.0$ e -1.0 em elementos específicos da matriz

Uma vez obtida essa matriz, é possível realizar a transformação de coordenadas tridimensionais do mundo para coordenadas bidimensionais na tela por meio da multiplicação matricial com a posição homogênea do objeto, conforme a Equação 4.2:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \times \begin{bmatrix} m11 & m21 & m31 & m41 \\ m12 & m22 & m32 & m42 \\ m13 & m23 & m33 & m43 \\ m14 & m24 & m34 & m44 \end{bmatrix} = \begin{bmatrix} screenX \\ screenY \\ screenZ \\ w \end{bmatrix} \tag{4.2}$$

O resultado dessa operação fornece as coordenadas projetadas na tela. O valor de profundidade (*screenZ*) é desconsiderado nesse contexto, e as coordenadas bidimensionais são obtidas após a divisão perspectiva pelo componente homogêneo (w), conforme a Equação 4.3:

$$x' = \frac{screenX}{w}, \quad y' = \frac{screenY}{w} \tag{4.3}$$

Valores negativos ou baixos de w indicam que o ponto não está visível (atrás da câmera ou fora do campo visual). Com isso, os elementos gráficos como os marcadores de ESP são corretamente posicionados na tela.

Após a divisão perspectiva pelas coordenadas homogêneas, é necessário adaptar os valores normalizados (x', y'), que variam entre -1 e 1, para coordenadas absolutas de tela (em pixels). Para isso, realiza-se uma translação e escala com base na resolução da janela do jogo. A largura e altura da tela são divididas por 2 para obter as coordenadas centrais da câmera $(camX, camY)$, e os valores projetados $(screenX, screenY)$ são escalados proporcionalmente em relação a esse centro. A coordenada horizontal é deslocada positivamente, enquanto a vertical é invertida (já que no sistema de tela a origem está no canto superior esquerdo), conforme a Equação 4.4:

$$x = \frac{\text{largura da tela}}{2} \cdot (1 + x'), \quad y = \frac{\text{altura da tela}}{2} \cdot (1 - y') \tag{4.4}$$

Esse ajuste posiciona corretamente os elementos na tela com base nas coordenadas do mundo, possibilitando a renderização precisa de elementos como caixas, indicadores ou marcadores visuais.

4.2.1.3 *Percepção Extra sensorial*

Com base nos conceitos explicados anteriormente, foi implementada a funcionalidade de percepção extra sensorial (ESP), que consiste na exibição visual da posição dos inimigos na tela do jogador, mesmo quando estão ocultos por obstáculos ou a grandes distâncias da posição atual da câmera.

Essa funcionalidade utiliza a matriz de MVP e as coordenadas tridimensionais dos inimigos, previamente extraídas da memória do jogo, para realizar a projeção correta dos alvos na tela. Por meio da transformação de coordenadas descrita na seção anterior, foi possível obter as posições 2D dos inimigos, em tempo real, diretamente sobre a interface do jogador.

Antes de renderizar qualquer marcador visual, foi implementada uma verificação para assegurar que os inimigos estejam efetivamente dentro do campo de visão. Essa checagem utiliza os resultados da transformação para garantir que os alvos estejam posicionados dentro dos limites visíveis da tela e à frente da câmera, evitando que elementos gráficos sejam exibidos indevidamente. Esse controle contribui para uma exibição mais precisa e consistente dos indicadores visuais.

Figura 13 – Exemplo da percepção extra sensorial.

Fonte: Elaborado pelo autor, 2025.

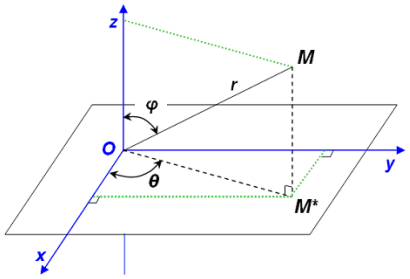
Na Figura 13, observa-se a funcionalidade da percepção extra sensorial (ESP) em execução. Os marcadores visuais foram gerados com base nas posições dos inimigos extraídas da memória e convertidas para a tela utilizando a matriz de MVP. Com isso, o jogador consegue identificar a posição dos inimigos em tempo real, mesmo que estejam fora do campo de visão normal.

4.2.1.4 *Auxílio de Mira*

Foi desenvolvido um sistema de auxílio de mira (*aim bot/aim assist*), responsável por ajustar automaticamente o direcionamento da câmera do jogador em direção aos inimigos. Para isso, foram extraídas da memória do jogo as coordenadas tridimensionais dos inimigos, bem como a posição atual do jogador. A partir dessas informações, foi possível calcular a distância e os ângulos necessários para alinhar a mira de forma precisa.

O jogo Assault Cube utiliza um sistema de coordenadas esféricas para definir o direcionamento da câmera, em que a orientação do olhar do jogador é representada por dois ângulos: um para a rotação horizontal (azimute) e outro para a rotação vertical (elevação). A Figura 14 ilustra esse sistema de coordenadas, que serviu como base para o cálculo dos ângulos necessários para alinhar a mira com precisão.

Figura 14 – Sistema de coordenadas esféricas.



Fonte: Wikimedia Commons, 2006.

Com base na diferença vetorial entre a posição do jogador (x_j, y_j, z_j) e a do inimigo (x_i, y_i, z_i), foi possível calcular os ângulos necessários para alinhar a mira. Inicialmente, definiu-se o vetor direção, conforme a Equação 4.5:

$$\vec{d} = (x_i - x_j,\ y_i - y_j,\ z_i - z_j) \tag{4.5}$$

O ângulo de rotação horizontal, correspondente ao azimute no sistema de coordenadas esféricas, foi calculado a partir da projeção do vetor direção sobre o plano horizontal (XY), conforme a Equação 4.6:

$$\theta = \arctan 2(y_i - y_j,\ x_i - x_j) \tag{4.6}$$

Já o ângulo de rotação vertical, equivalente à elevação, foi determinado com base na diferença de altura entre o jogador e o inimigo, considerando a distância euclidiana no plano XY, conforme a Equação 4.7:

$$\phi = \arctan 2(z_i - z_j,\ \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}) \tag{4.7}$$

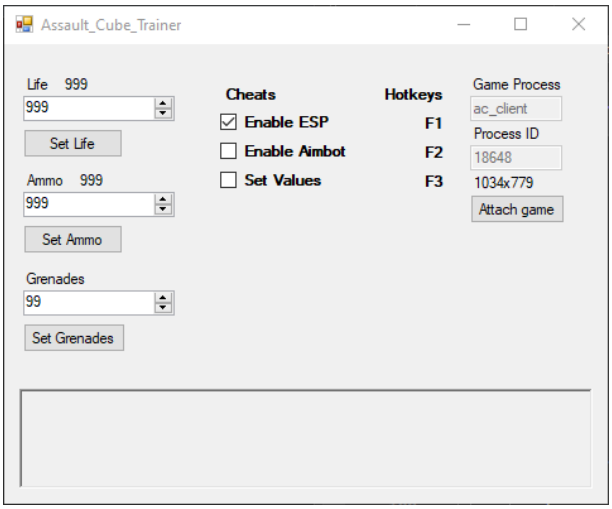
Os valores de θ e ϕ foram então convertidos para graus, no formato esperado pelo jogo, e aplicados diretamente na memória da câmera. Esse procedimento permitiu que a mira acompanhasse automaticamente o inimigo, ajustando a orientação da câmera em tempo real. Como o ângulo de elevação (ϕ) é limitado ao intervalo de -90° a +90°, foi implementada uma correção para garantir que o valor calculado permanecesse dentro desses limites, evitando comportamentos inválidos ou movimentos abruptos da câmera.

Adicionalmente, foram realizados cálculos com a matriz de MVP do jogo para verificar se o inimigo encontrava-se dentro do campo de visão antes de aplicar o ajuste. Essa verificação impede que a câmera se mova em direção a inimigos fora da tela, o que comprometeria a naturalidade do movimento e poderia evidenciar a automação do sistema.

4.3 Prova de conceito

O software criado permitiu a modificação de variáveis críticas do jogo, como pontos de vida, munição e a quantidade de granadas, além da implementação de um sistema de Extra Sensory Perception (ESP) responsável por exibir a posição dos inimigos em tempo real, e também um sistema de auxilio de mira que ajusta a mira automaticamente para os inimigos. A Figura 15 apresenta a interface do software desenvolvido, exibindo as funcionalidades implementadas.

Figura 15 – O software desenvolvido como prova de conceito.



Fonte: Elaborado pelo autor, 2025.

A prova de conceito desenvolvida conta com as seguintes funcionalidades principais:

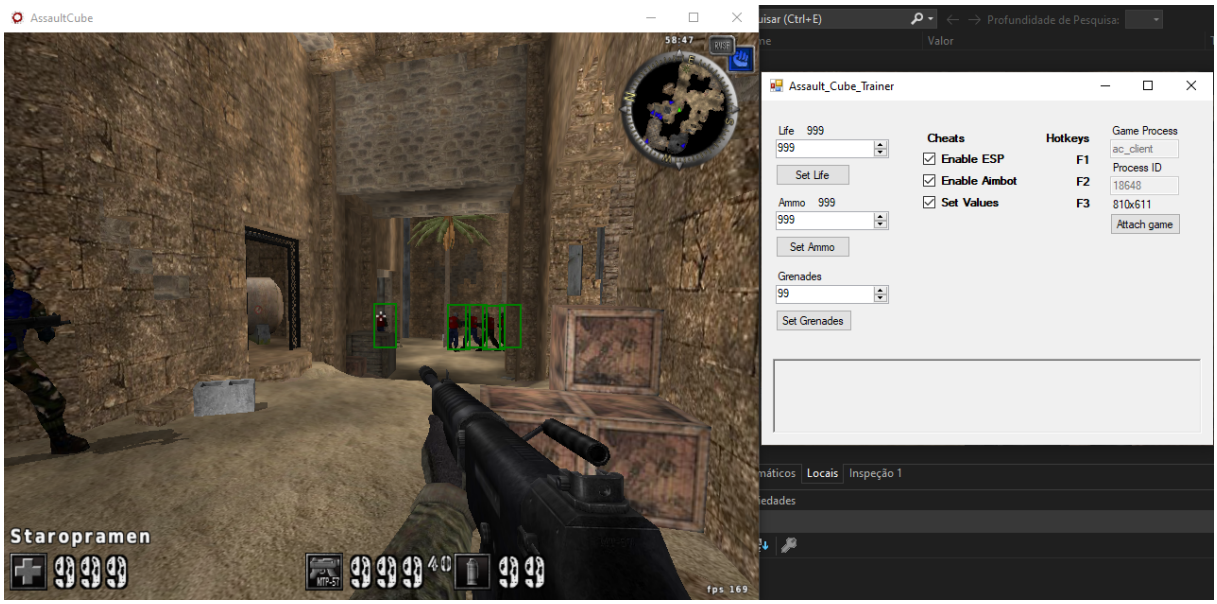
- **Extra Sensory Perception (ESP):** Exibe a posição dos inimigos em tempo real, permitindo que o jogador visualize adversários mesmo através de obstáculos.
- **Aimbot:** Ajusta automaticamente a mira do jogador para a cabeça dos inimigos, calculando os ângulos corretos para alinhar a visão do jogador ao alvo.

- **Set Values (Atualização Contínua de Valores):** Mantém os valores de vida, munição e granadas constantemente atualizados, impedindo que o jogo reduza esses atributos ao longo da partida.

Para garantir usabilidade e controle rápido, o software foi desenvolvido com teclas de atalho que permitem ativar e desativar cada funcionalidade de maneira prática:

- **F1:** Ativa/Desativa o ESP.
- **F2:** Ativa/Desativa o Aimbot.
- **F3:** Ativa/Desativa o Set Values (atualização contínua de valores).

Figura 16 – Teste da prova de conceito.



Fonte: Elaborado pelo autor, 2025.

Na Figura 16 é ilustrado o teste prático da ferramenta em execução, evidenciando as modificações realizadas no jogo, sendo possível observar o ESP ativado, a modificação dos atributos do jogador (vida, munição e quantidade de granadas), e a mira do jogador posicionada no inimigo.

Os resultados obtidos pela prova de conceito, que possibilitou o acesso e a modificação de variáveis críticas sem acesso ao código-fonte, evidenciam o potencial das técnicas de engenharia reversa e manipulação de memória tanto para fins educacionais quanto para a identificação e exploração de vulnerabilidades em sistemas.

4.4 Limitações do Estudo

Apesar dos resultados positivos, o estudo apresenta algumas limitações que devem ser consideradas. O jogo utilizado como objeto de estudo, o Assault Cube, é de código aberto e não possui sistemas anti-trapaça que realizam verificações de integridade, monitoramento comportamental ou interceptação de chamadas de sistemas. Essa ausência de barreiras reduziu significativamente a complexidade das manipulações, facilitando a aplicação das técnicas apresentadas neste trabalho, tanto no processo da análise dinâmica do Jogo com o software Cheat Engine, quanto no desenvolvimento da prova de conceito que usou a biblioteca Memory.DLL, que simplifica as interações com a API do Sistema Operacional Windows.

A prova de conceito foi validada em um ambiente controlado, utilizando a versão 1.3.0.2 do Assault Cube e executada no sistema operacional Windows 10. Mudanças nessas condições, como atualizações do jogo, variações de compilação do jogo ou a execução em outras sistemas operacionais, podem comprometer a validade dos endereços de memória mapeados, exigindo uma nova análise dinâmica e remapeamento da memória para restaurar as funcionalidades da ferramenta.

O desenvolvimento da aplicação foi realizado com o .NET Framework 4.7.2, versão compatível com edições do Windows a partir do Windows 7 Service Pack 1. Apesar de garantir portabilidade para sistemas mais antigos, restrições de permissões ou configurações de segurança podem impactar a execução em ambientes corporativos ou sistemas mais recentes com políticas restritivas.

Além disso, a biblioteca *Memory.dll* oferece uma camada de abstração simplificada para chamadas como *ReadProcessMemory* e *WriteProcessMemory*, porém não implementa mecanismos avançados de ofuscação, como manipulação de memória via drivers em modo núcleo que evitam a interceptação de chamadas de sistema (*syscalls hooking*). Essa limitação torna a solução mais suscetível à detecção por sistemas anti-trapaça amplamente utilizados, como Easy Anti-Cheat ou BattlEye.

Por fim, este estudo não abordou técnicas avançadas de manipulação de memória realizadas em modo núcleo ou por meio de hardware com acesso direto à memória (DMA), métodos normalmente empregados em cenários que demandam maior furtividade para evasão de mecanismos de detecção.

5 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho demonstrou, na prática, a aplicação de técnicas de engenharia reversa em jogos digitais, com foco na manipulação direta de valores sensíveis armazenados na memória, como pontos de vida, munição e quantidade de granadas. A implementação em C# com Framework .NET e interface gráfica em Windows Forms utilizou a biblioteca *Memory.dll*, permitindo a leitura e escrita em tempo de execução com respostas visíveis e imediatas no ambiente do jogo.

Os testes realizados confirmaram que, em softwares sem mecanismos robustos de proteção, a engenharia reversa pode ser utilizada com relativa facilidade para mapear e modificar estruturas internas. A implementação do recurso de percepção extra sensorial (ESP) evidenciou a viabilidade de extrair e exibir informações do jogo em tempo real, o que pode ser aplicado em contextos de estudo, auditoria e identificação de falhas de segurança.

Durante o processo, também foram identificados desafios como a realocação dinâmica de endereços a cada nova execução do jogo. Para lidar com isso, foi adotada a técnica de escaneamento de ponteiros, garantindo acesso estável às variáveis desejadas. Além disso, a ausência de sistemas anti-trapaça no Assault Cube simplificou a leitura e manipulação da memória, mas, em ambientes mais protegidos, seriam necessárias estratégias adicionais para contornar mecanismos de integridade e verificação.

Embora a pesquisa tenha fins acadêmicos, os resultados reforçam a necessidade de se discutir as implicações éticas e de segurança dessas técnicas. O uso indevido pode comprometer a integridade de jogos e afetar negativamente suas comunidades. Isso destaca a importância do desenvolvimento contínuo de mecanismos de defesa, como criptografia de variáveis críticas, verificações de integridade e detecção de alterações não autorizadas.

Este trabalho, além de contribuir tecnicamente, pode ser utilizado como material introdutório para capacitação prática em engenharia reversa. Para permitir a reprodução dos experimentos e incentivar a continuidade de estudos na área, o código-fonte e os artefatos desenvolvidos estão disponíveis no repositório do projeto no GitHub: **<https://github.com/gabrielgollo/AssaultCubeTrainer>**

5.1 Trabalhos Futuros

Este estudo focou na manipulação de memória com o uso da biblioteca *Memory.dll*, que abstrai a comunicação com a API do sistema operacional. Pesquisas futuras podem avançar para abordagens mais sofisticadas, tanto em nível de software quanto de hardware.

Entre as possíveis extensões deste trabalho, destacam-se:

- **Acesso direto à memória (DMA):** Técnicas baseadas em hardware permitiriam interagir com a memória do jogo de forma mais eficiente e difícil de ser detectada por operar fora da supervisão do sistema operacional.

- **Implementação de drivers próprios para manipulação de memória:** O desenvolvimento de drivers próprios que sejam executados em modo kernel possibilitaria a leitura e escrita direta na memória física, contornando as restrições impostas em modo usuário e evitando interceptações por mecanismos de segurança convencionais.
- **Análise de sistemas anti-trapaça:** Estudar os principais mecanismos de proteção do mercado pode revelar vulnerabilidades e ajudar no desenvolvimento de contramedidas mais eficazes.
- **Aplicação em outros jogos:** Adaptar a metodologia deste trabalho para jogos comerciais com diferentes motores gráficos e níveis de proteção, permitindo comparações sobre a eficácia das técnicas empregadas.

Este trabalho não apenas cumpriu seus objetivos iniciais, como também abriu espaço para investigações futuras sobre engenharia reversa e segurança em jogos digitais, incentivando o estudo de práticas de defesa e ataque em ambientes computacionais modernos.

REFERÊNCIAS

BLIX.GG. **CS2 Cheating Problem – An In-Depth Look**. 2023. Acesso em: 29 jun. 2025. Disponível em: <https://blix.gg/news/cs-2/cs2-cheating-problem-an-in-depth-look>. Citado na página 23.

CVE Program. **Genshin Impact mhyprot2.sys Kernel Driver Privilege Escalation Vulnerability**. [S.l.], 2020. Acesso em: 2025-05-10. Disponível em: <https://www.cve.org/CVERecord?id=CVE-2020-36603>. Citado na página 26.

EILAM, E. **Reversing: Secrets of Reverse Engineering**. Wiley, 2011. ISBN 9781118079768. Disponível em: <https://books.google.com.br/books?id=_78HnPPRU_oC>. Citado 2 vezes nas páginas 17 e 18.

European Union Agency for Cybersecurity. **ENISA Threat Landscape 2023**. 2023. Relatório técnico. Acesso em: 2025-05-07. Disponível em: <https://www.enisa.europa.eu/publications/enisa-threat-landscape-2023>. Citado na página 14.

HEIJNEN, E. **Cheat Engine**. 2000. <https://www.cheatengine.org/>. Open-source memory scanner and debugger for Windows. Citado 2 vezes nas páginas 15 e 19.

HEX-RAYS. **IDA Pro**. 1996. <https://hex-rays.com/ida-pro/>. Interactive Disassembler for reverse engineering. Citado na página 18.

KASPERSKY. **Kaspersky apresenta solução para identificar trapaças em eSports**. 2019. <https://www.kaspersky.com.br/about/press-releases/kaspersky-apresenta-solucao-para-identificar-trapacas-em-esports>. Acesso em: 2024-09-28. Citado na página 23.

KORKIN, I. **Divide et Impera: MemoryRanger Runs Drivers in Isolated Kernel Spaces**. 2018. Disponível em: <https://arxiv.org/abs/1812.09920>. Citado na página 21.

MELO, L. P. de *et al.* Análise de malware: Investigação de códigos maliciosos através de uma abordagem prática. **Sociedade Brasileira de Computação**, 2011. Citado na página 17.

MERCES, F. **Fundamentos da Engenharia Reversa**. 1981. Acesso em: 2024-09-28. Disponível em: <https://mentebinaria.gitbook.io/engenharia-reversa>. Citado na página 20.

NATIONAL SECURITY AGENCY. **Ghidra**. 2019. <https://ghidra-sre.org>. Software reverse engineering framework released by the NSA. Citado na página 18.

OLLYDBG. **OllyDbg**. 2004. <http://www.ollydbg.de/>. 32-bit assembler level debugger for Windows. Citado na página 18.

OPENGLTUTORIAL. **Tutorial 3: Matrices - OpenGL Tutorial**. 2021. Acesso em: 2025-05-03. Disponível em: <https://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/>. Citado na página 36.

RENDENBACH, C. A. **Anti-Cheating Measures in Video Games**. Tese (Doutorado) — Technical University of Munich., 2022. Citado na página 24.

ROBLES, R. J.; YEO, S.-S.; MOON, Y.-D.; PARK, G.; KIM, S. Online games and security issues. In: **2008 Second International Conference on Future Generation Communication and Networking**. [S.l.: s.n.], 2008. v. 2, p. 145–148. Citado na página 14.

ROLIM, D. L. R. VisÃo geral sobre as trapaÇas e mÉtodos antitrapaÇas em jogos digitais: Entendendo a indústria, atores, métodos e preocupações acerca da privacidade na garantia de integridade competitiva entre os jogadores. **Universidade Federal do Rio de Janeiro**, 2023. Citado 2 vezes nas páginas 20 e 26.

T0ST. **How I cut GTA Online loading times by 70%**. 2021. Acesso em: 2024-09-28. Disponível em: <https://nee.lv/2021/02/28/How-I-cut-GTA-Online-loading-times-by-70/>. Citado na página 17.

TANENBAUM, A. S. **Sistemas operacionais modernos**. Prentice-Hall do Brasil, 2010. ISBN 9788576052371. Disponível em: <https://books.google.com.br/books?id=nDatQwAACAAJ>. Citado 3 vezes nas páginas 20, 22 e 25.

TREND MICRO RESEARCH. Ransomware actor abuses genshin impact anti-cheat driver to kill antivirus. **Trend Micro Research Blog**, 2022. Acesso em: 2025-05-10. Disponível em: <https://www.trendmicro.com/en_us/research/22/h/ransomware-actor-abuses-genshin-impact-anti-cheat-driver-to-kill-antivirus.html>. Citado na página 26.

VICENTINE, A. L.; SILVA, G. C. Mendes da; NASCIMENTO, J. P. Domingues Guedes do; NEVES, J. E. D. A. Software anti-cheat: Uma maneira de prevenir trapaças em jogos multiplayer. **Revista Brasileira em Tecnologia da Informação**, v. 4, n. 1, p. 38–47, fev. 2022. Disponível em: <https://www.fateccampinas.com.br/rbti/index.php/fatec/article/view/54>. Citado na página 22.

VRIES, J. de. **Learn OpenGL - Coordinate Systems**. 2020. Acesso em: 2025-05-03. Disponível em: <https://learnopengl.com/Getting-started/Coordinate-Systems>. Citado na página 36.

Wikimedia Commons. **Spherical coordinates**. 2006. Disponível em: Wikimedia Commons. Acesso em: 29 jul. 2025. Disponível em: <https://upload.wikimedia.org/wikipedia/commons/e/e4/Spherical-coordinates.png>. Citado na página 39.

x64dbg Team. **x64dbg**. 2015. <https://x64dbg.com/>. Open-source x64/x32 debugger for Windows. Citado na página 18.