Using VBS enclaves for anti-cheat purposes

tulach.cc/using-vbs-enclaves-for-anti-cheat-purposes

Nov 09, 2024 Samuel Tulach

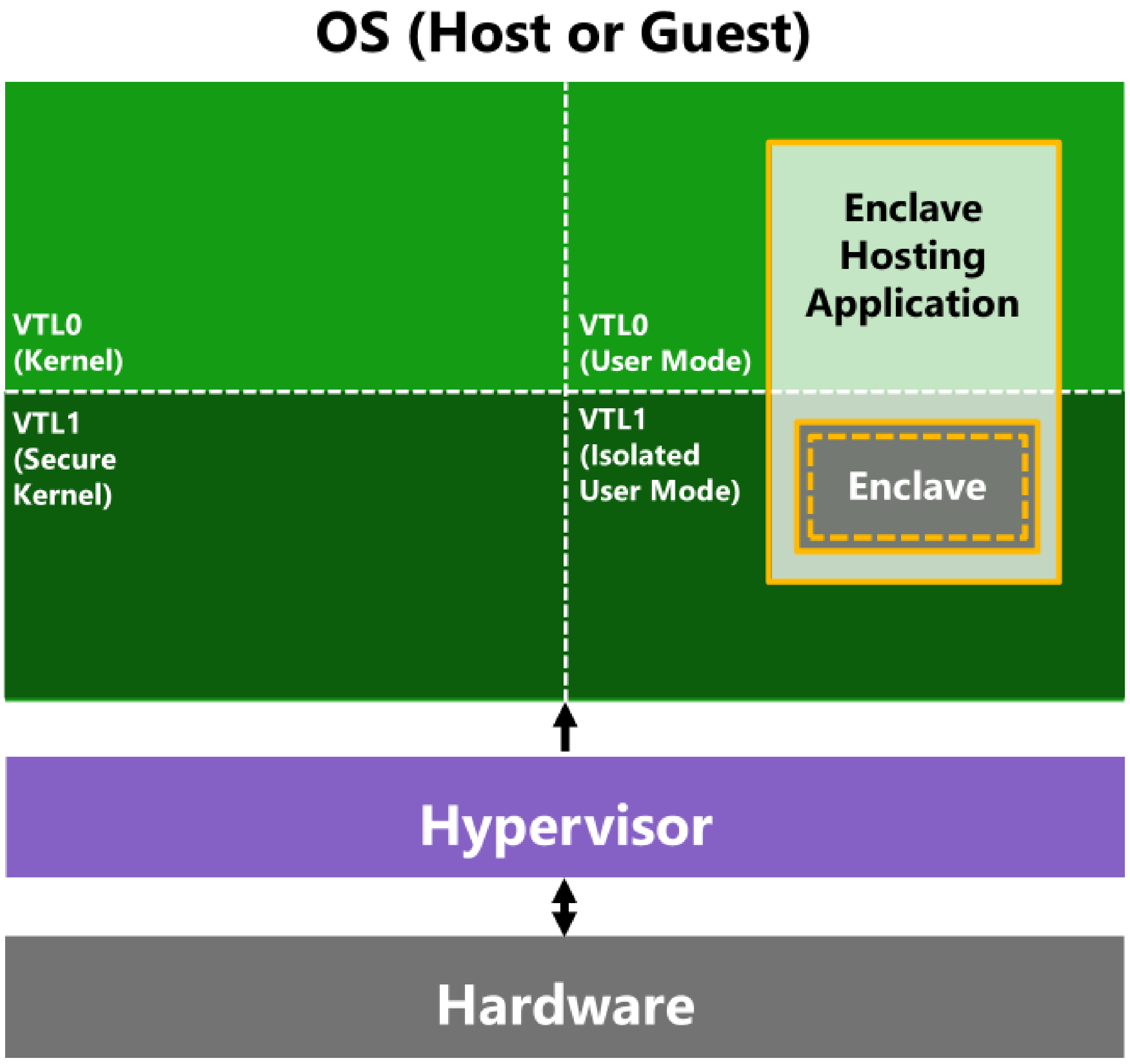
A few months ago, when Microsoft announced VBS (virtualization-based security) enclave functionality, I started wondering whether it could be used for game anti-cheating purposes. While I was skeptical (later I will explain why), I decided to look into it and write a simple Pong-inspired game which handles its entire game logic in such enclave.

What are VBS enclaves?

Hyper-V is a type-1 hypervisor, which means that when enabled, the Windows installation it's running on becomes a guest.

Virtualization-based security (VBS) is built on top of the Hyper-V platform. Since it operates at a higher privilege level than the kernel itself, it can help enforce strict code signing requirements (HVCI) or completely isolate data even from code running at the kernel level.

VBS enclaves allow developers to leverage VBS in their applications to isolate code and data from anything else running on the computer. Think of it as running Windows inside VirtualBox or VMware Player as one VM, with this isolated environment as another. Neither can access the other unless the hosting application explicitly permits it.



Trusted execution enclaves using VBS (source)

Anti-cheat?

Traditionally on Windows, anti-cheat software utilizes kernel-mode drivers to (among other things) prevent any user-mode programs from accessing the game's memory. I have written about it in more detail here.

The idea of using VBS enclaves is that we could run parts of the game in this isolated environment. That would mean that even if cheat developers somehow got kernel-mode code execution, they would still not be able to manipulate the code and data in this enclave.

A bit of skepticism

While it sounds like a good idea on paper, there are currently a few problems with the use of VBS enclaves:

- **Limited APIs** - You can’t just take an existing program and tell Windows to run it in an enclave. It has to be a library (.DLL) specifically designed to run in such enclave which then needs a host process. There is a very limited set of APIs available (*libvcruntime*, *vertdll*, *UCRT*, *bcrypt*).
- **Performance** - Since virtualization is involved, there will always be some overhead, which in other use-cases is not that significant, but for a game trying to run logic or get data in/out of the enclave on each frame, this overhead is something that developers have to account for.
- **Security** - While in theory, the enclave should be absolutely impenetrable, in practice, cheat developers will always have the edge unless the system becomes completely locked down including the firmware. While it’s true that the enclave is inaccessible to anything running *inside* the OS (since anything running inside the OS will run inside the virtualized environment), nothing is stopping cheat developers from writing a bootkit that will load before the OS even starts, therefore before any virtualization takes place. With a clever hook chain, they can gain complete control over the Hyper-V/VBS architecture itself (something I will be exploring in future blog posts).

Regardless of these issues, I decided to write a simple proof-of-concept project to test this idea out and to have something to experiment with in the future.

Getting started

Since VBS enclave functionality is quite new, the available sample projects and documentation are very sparse. In fact, I was not able to find any open-source project utilizing VBS enclaves, so the only starting point I could use was the official documentation and sample project.

Development tools required:

- Visual Studio 2022 version 17.9 or later
- Windows Software Development Kit (SDK) version 10.0.22621.3233 or later

Device/OS requirements:

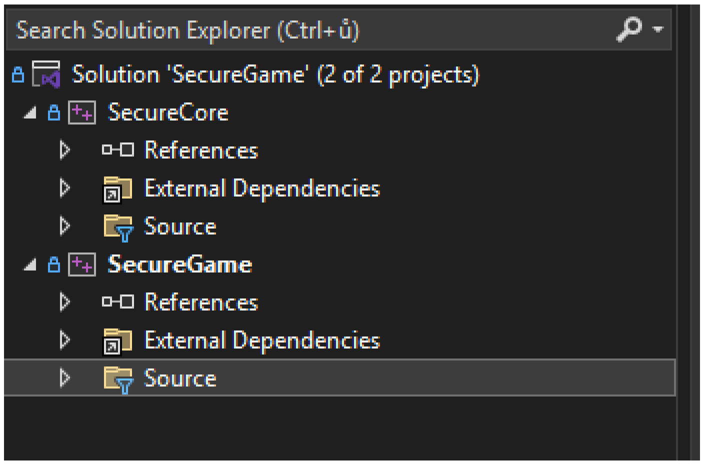
- Windows 11 or later or Windows Server 2019 or later
- VBS/HVCI must be enabled
- For debugging and running enclaves without production signing, test-signing must be enabled

I used VMware Workstation to run the latest version of Windows 11. Don’t forget to enable nested virtualization in setting if you also do so (Virtualize Intel VT-x/EPT or AMD-V/RVI).

Since I wanted something extremely simple, I decided to write a game inspired by the classic Pong. The idea was to have a host process that would handle initialization, window creation, keyboard input and rendering, while the enclave would run the actual game logic.

I created a new solution with two projects: SecureGame, which would be the host process, and SecureCore, which would be the enclave itself.

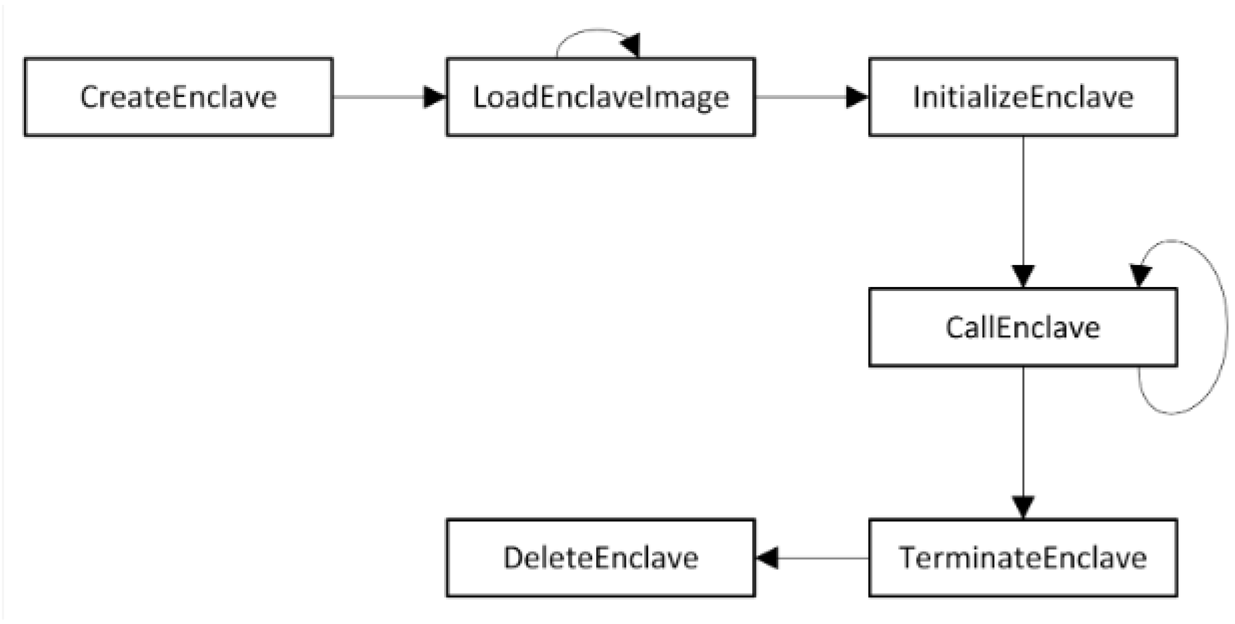
I used vcpkg to install SDL2 and then started implementing the game logic. There's nothing special about it; I ended up with a game loop like this:



```
void Game::Loop()
{
    constexpr int FPS = 240;
    constexpr int frameDelay = 1000 / FPS;

    bool running = true;
    while (running)
    {
        const Uint32 frameStart = SDL_GetTicks();

        SDL_Event event;
        while (SDL_PollEvent(&event))
        {
            if (event.type == SDL_QUIT)
                running = false;
        }

        SDL_SetRenderDrawColor(m_Renderer, 0, 0, 0, 255);
        SDL_RenderClear(m_Renderer);

        Tick();

        SDL_RenderPresent(m_Renderer);

        const int frameTime = SDL_GetTicks() - frameStart;
        if (frameDelay > frameTime)
            SDL_Delay(frameDelay - frameTime);
    }

    SDL_DestroyRenderer(m_Renderer);
    SDL_DestroyWindow(m_Window);
    SDL_Quit();
}
```

The interesting stuff

When the enclave is loaded, the host process can call any exported function from the enclave library using `CallEnclave()`.



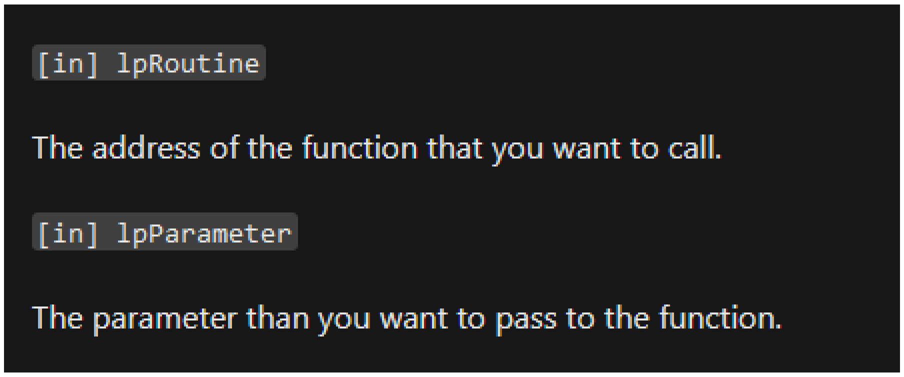
Enclave lifecycle (source)

There are several ways we could approach using it:

1. Use it only for save/load - Store and load data from the enclave, keeping the loaded data in host process memory for the least amount of time possible and only for the time they are needed.
2. Supply all required input and run the entire game logic inside the enclave. The enclave then returns data needed for rendering, which is performed by the host process.
3. Put everything in the enclave - The `CallEnclave()` function can also perform a "reverse call" back to the host process according to the documentation. We could write a function that allows arbitrary function calls in the host process and then call this single function from within the enclave. This would allow us to put the entire game inside the enclave and work around its limitations.

While option 3 seems the most interesting, I went with option 2 to keep things simple.

Initially, I wondered whether you could pass a pointer to host process data into the enclave and access it directly. The documentation doesn't mention whether the host process memory is accessible to the enclave at all, and to make matters worse, the sample code only passes data values, never pointers.


`[in] lpRoutine`

The address of the function that you want to call.

`[in] lpParameter`

The parameter than you want to pass to the function.

`CallEnclave()` *(very helpful) documentation (source)*

By trying it out, I discovered that the enclave **has access to the host process memory**, so you can pass pointers to data structures inside the host process. With this knowledge, I created a structure to be shared between the enclave and the host process.

```
typedef struct _TICK_DATA
{
    float DeltaTime;

    bool KeyW;
    bool KeyS;
    bool KeyUp;
    bool KeyDown;

    struct
    {
        float X;
        float Y;
        float Width;
        float Height;
    } LeftPaddle, RightPaddle, Ball;

    int LeftScore;
    int RightScore;
} TICK_DATA;
```

This structure would hold information about which keys are pressed on the current tick, the time elapsed since the last frame rendered, and the output from the enclave containing game object positions and score.

The enclave would then implement the game logic like this (peak coding performance, don't judge):

```
void Reset()
{
    Data::BallPositionX = static_cast<float>(WINDOW_WIDTH) / 2 - BALL_SIZE / 2;
    Data::BallPositionY = static_cast<float>(WINDOW_HEIGHT) / 2 - BALL_SIZE / 2;

    Data::BallVelocityX = (rand() % 2 == 0) ? BALL_SPEED : -BALL_SPEED;
    Data::BallVelocityY = (rand() % 2 == 0) ? BALL_SPEED : -BALL_SPEED;

    Data::State = Data::StateId::Running;
}

void Run(TICK_DATA* currentTick)
{
    const float deltaTime = currentTick->DeltaTime;

    if (currentTick->KeyW && Data::LeftPaddleY > 0)
        Data::LeftPaddleY -= PADDLE_SPEED * deltaTime;
    if (currentTick->KeyS && Data::LeftPaddleY < WINDOW_HEIGHT - PADDLE_HEIGHT)
        Data::LeftPaddleY += PADDLE_SPEED * deltaTime;
    if (currentTick->KeyUp && Data::RightPaddleY > 0)
        Data::RightPaddleY -= PADDLE_SPEED * deltaTime;
    if (currentTick->KeyDown && Data::RightPaddleY < WINDOW_HEIGHT - PADDLE_HEIGHT)
        Data::RightPaddleY += PADDLE_SPEED * deltaTime;

    Data::BallPositionX += Data::BallVelocityX * deltaTime;
    Data::BallPositionY += Data::BallVelocityY * deltaTime;

    if (Data::BallPositionY <= 0 || Data::BallPositionY + BALL_SIZE >= WINDOW_HEIGHT)
        Data::BallVelocityY = -Data::BallVelocityY;

    const bool ballInLeftPaddleYRange = Data::BallPositionY + BALL_SIZE >=
Data::LeftPaddleY &&
        Data::BallPositionY <= Data::LeftPaddleY + PADDLE_HEIGHT;
    const bool ballInRightPaddleYRange = Data::BallPositionY + BALL_SIZE >=
Data::RightPaddleY &&
        Data::BallPositionY <= Data::RightPaddleY + PADDLE_HEIGHT;

    if (Data::BallPositionX <= 20 + PADDLE_WIDTH &&
        Data::BallPositionX >= 20 &&
        ballInLeftPaddleYRange)
    {
        Data::BallPositionX = 20 + PADDLE_WIDTH;
        Data::BallVelocityX = -Data::BallVelocityX;
    }

    if (Data::BallPositionX + BALL_SIZE >= WINDOW_WIDTH - 20 - PADDLE_WIDTH &&
        Data::BallPositionX <= WINDOW_WIDTH - 20 &&
        ballInRightPaddleYRange)
    {
        Data::BallPositionX = WINDOW_WIDTH - 20 - PADDLE_WIDTH - BALL_SIZE;
        Data::BallVelocityX = -Data::BallVelocityX;
    }
    if (Data::BallPositionX <= 0)
    {
        Data::RightScore++;
        Data::State = Data::StateId::Reset;
    }
    if (Data::BallPositionX + BALL_SIZE >= WINDOW_WIDTH)
    {
        Data::LeftScore++;
        Data::State = Data::StateId::Reset;
    }
}

extern "C" __declspec(dllexport) void* CALLBACK GameTick(PVOID context)
{
    TICK_DATA* currentTick = static_cast<TICK_DATA*>(context);

    switch (Data::State)
    {
    case Data::StateId::Reset:
        Reset();
        break;
    case Data::StateId::Running:
        Run(currentTick);
        break;
    }

    currentTick->LeftPaddle.X = 20;
    currentTick->LeftPaddle.Y = Data::LeftPaddleY;
    currentTick->LeftPaddle.Width = PADDLE_WIDTH;
    currentTick->LeftPaddle.Height = PADDLE_HEIGHT;

    currentTick->RightPaddle.X = WINDOW_WIDTH - 20 - PADDLE_WIDTH;
    currentTick->RightPaddle.Y = Data::RightPaddleY;
    currentTick->RightPaddle.Width = PADDLE_WIDTH;
    currentTick->RightPaddle.Height = PADDLE_HEIGHT;

    currentTick->Ball.X = Data::BallPositionX;
    currentTick->Ball.Y = Data::BallPositionY;
    currentTick->Ball.Width = BALL_SIZE;
    currentTick->Ball.Height = BALL_SIZE;

    currentTick->LeftScore = Data::LeftScore;
    currentTick->RightScore = Data::RightScore;

    return nullptr;
}
```

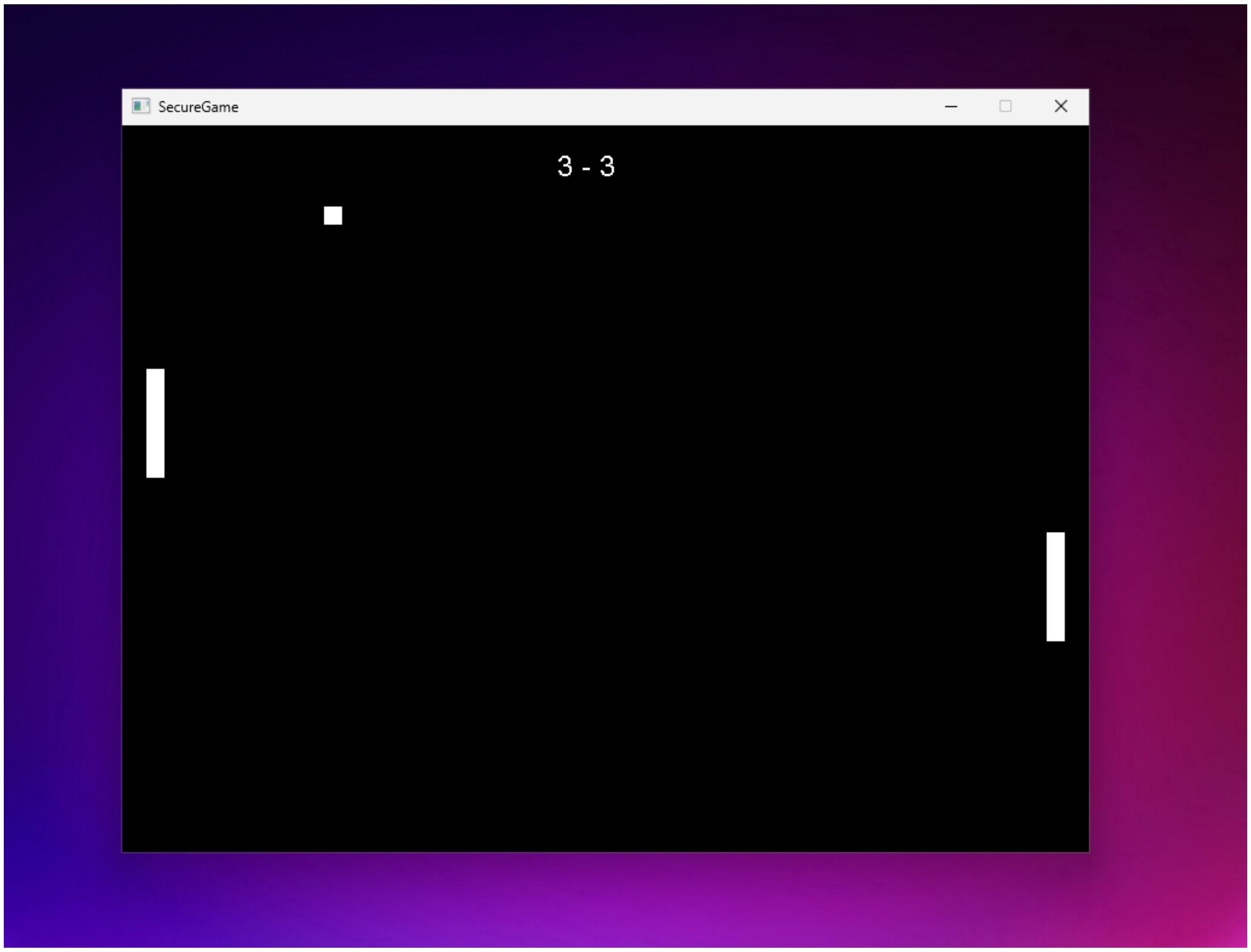
And then the enclave function is called from within the game tick in the host process, and then the resulting game objects are rendered:

```
void Game::Tick()
{
    const Uint8* keystates = SDL_GetKeyboardState(nullptr);

    static Uint32 lastTime = SDL_GetTicks();
    const Uint32 currentTime = SDL_GetTicks();
    const float deltaTime = (currentTime - lastTime) / 1000.0f;
    lastTime = currentTime;

    TICK_DATA data;
    data.DeltaTime = deltaTime;
    data.KeyW = keystates[SDL_SCANCODE_W];
    data.KeyS = keystates[SDL_SCANCODE_S];
    data.KeyUp = keystates[SDL_SCANCODE_UP];
    data.KeyDown = keystates[SDL_SCANCODE_DOWN];

    PVOID returnValue = nullptr;
    if (!CallEnclave(Global::TickRoutine, &data, true, &returnValue))
    {
        char buffer[256];
        sprintf_s(buffer, "Failed to call enclave routine: %d", GetLastError());
        MessageBoxA(nullptr, buffer, "Error", MB_OK | MB_ICONERROR);
        return;
    }

    SDL_SetRenderDrawColor(m_Renderer, 255, 255, 255, 255);

    const SDL_Rect leftPaddle =
    {
        static_cast<int>(data.LeftPaddle.X),
        static_cast<int>(data.LeftPaddle.Y),
        static_cast<int>(data.LeftPaddle.Width),
        static_cast<int>(data.LeftPaddle.Height)
    };
    SDL_RenderFillRect(m_Renderer, &leftPaddle);

    const SDL_Rect rightPaddle =
    {
        static_cast<int>(data.RightPaddle.X),
        static_cast<int>(data.RightPaddle.Y),
        static_cast<int>(data.RightPaddle.Width),
        static_cast<int>(data.RightPaddle.Height)
    };
    SDL_RenderFillRect(m_Renderer, &rightPaddle);

    const SDL_Rect ball =
    {
        static_cast<int>(data.Ball.X),
        static_cast<int>(data.Ball.Y),
        static_cast<int>(data.Ball.Width),
        static_cast<int>(data.Ball.Height)
    };
    SDL_RenderFillRect(m_Renderer, &ball);

    char scoreText[32];
    sprintf_s(scoreText, "%d - %d", data.LeftScore, data.RightScore);
    RenderText(scoreText, WINDOW_WIDTH / 2 - 40, 20);
}
```

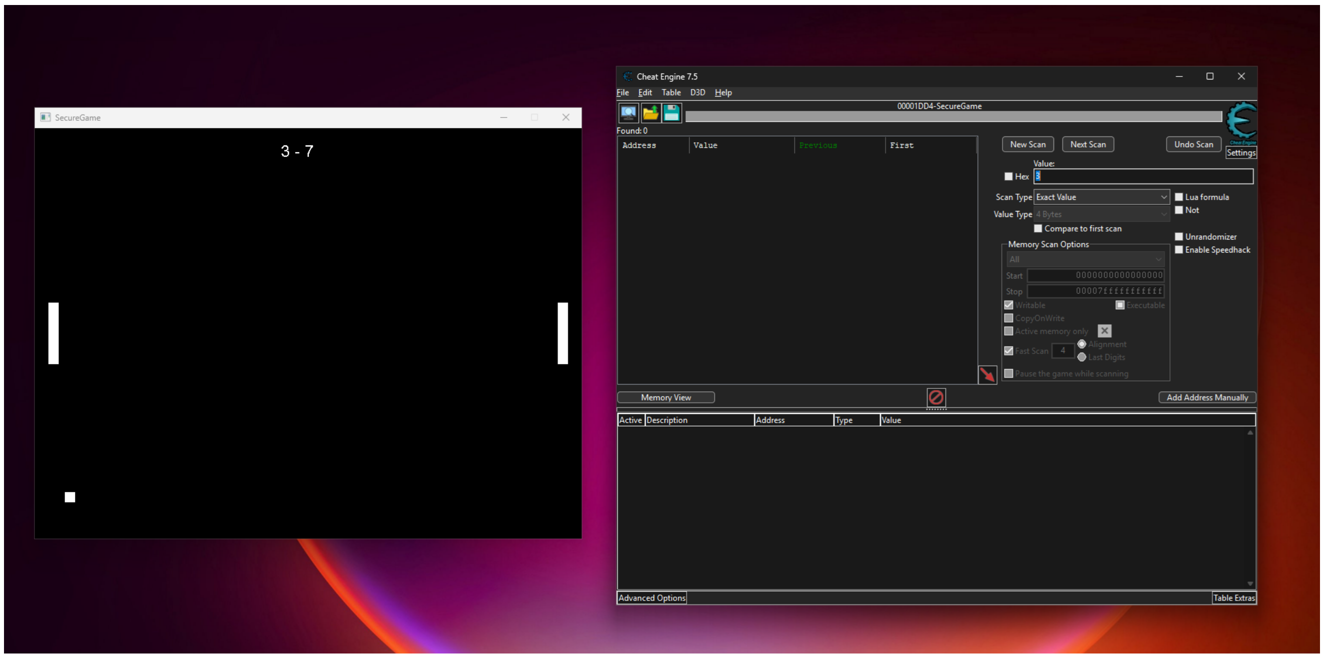
And that's it, now let's go test it out! Full source code available here.

Testing

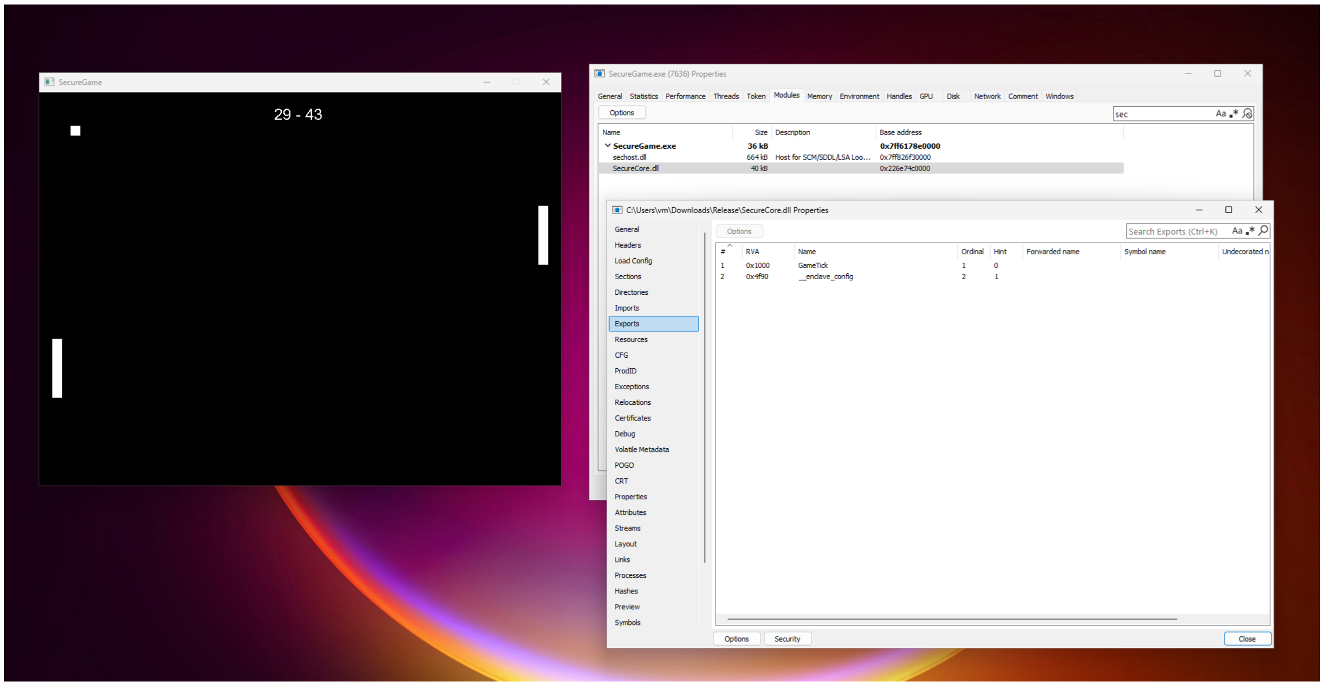
When you start the game, nothing will seem out of the ordinary (apart from the fact that it won't run without VBS enabled).



Let's try to mess with it. Trying to change the score values using Cheat Engine won't work. The score isn't in the host process memory at all (or it's there only for a very brief moment when it's rendered on screen, but changing it won't affect the actual score counter).



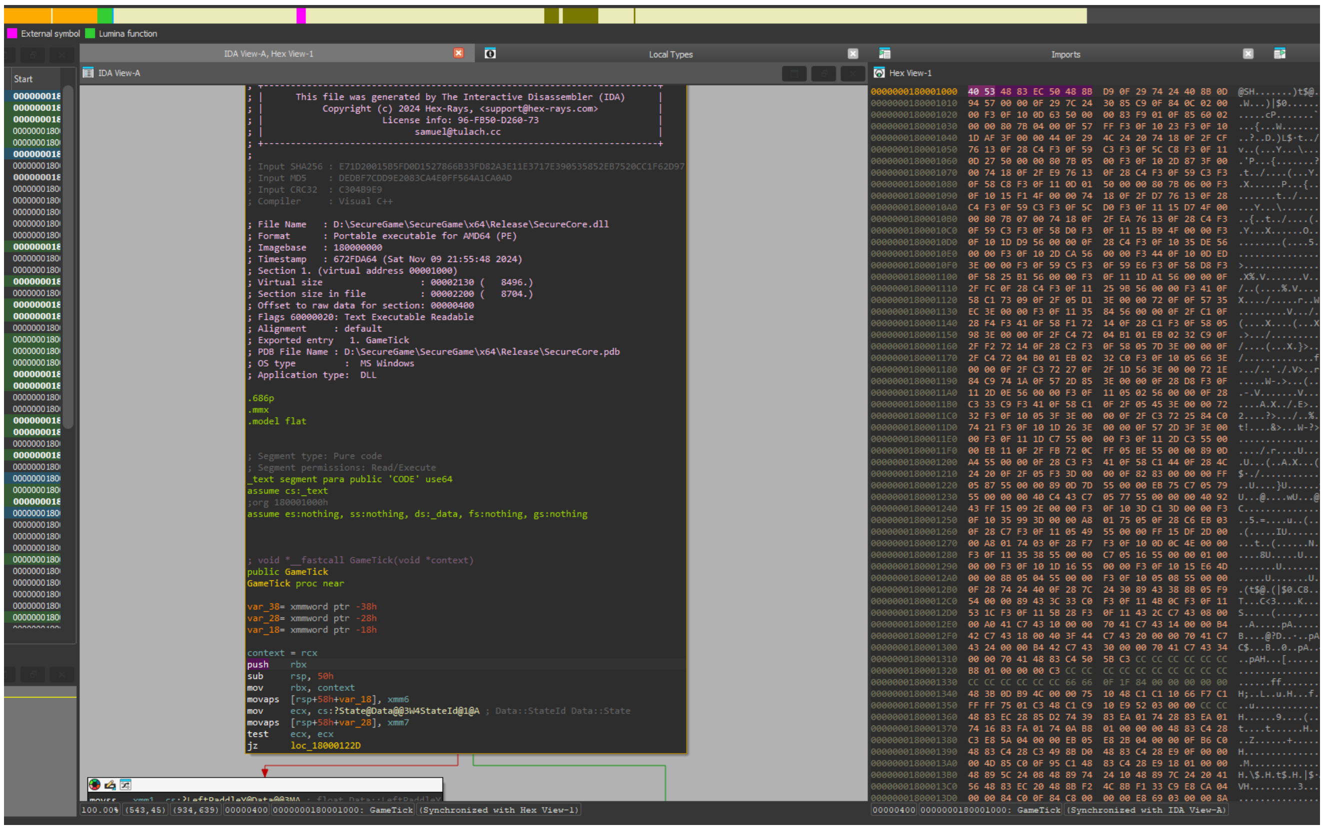
Ok, so what about trying to edit the code directly? When we look at the loaded modules of the process, we can actually see the loaded enclave DLL.



Name	Size	Description	Base address
SecureGame.exe	**36 kB**		**0x7ff6178e0000**
sechost.dll	664 kB	Host for SCM/SDDL/LSA Loo…	0x7ff826f30000
SecureCore.dll	40 kB		0x226e74c0000

#	RVA	Name	Ordinal	Hint	Forwarded name	Symbol name	Undecorated n
1	0x1000	GameTick	1	0			
2	0x4f90	__enclave_config	2	1			

Does that mean we can just patch the module? Let's open `SecureCore.dll` in IDA and copy over the first bytes of the `GameTick()` function.



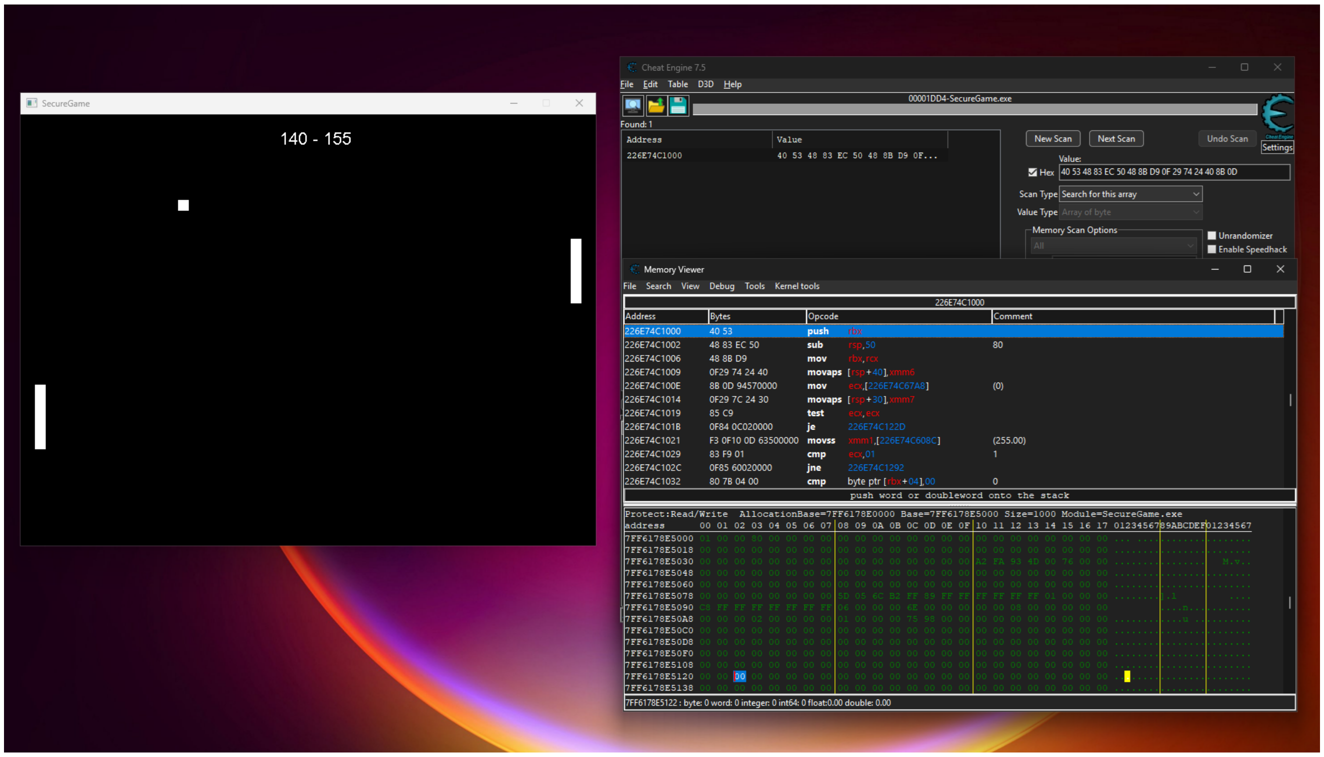
```
; +-------------------------------------------------------------------------+
; |      This file was generated by The Interactive Disassembler (IDA)      |
; |           Copyright (c) 2024 Hex-Rays, <support@hex-rays.com>           |
; |                      License info: 96-FB50-D260-73                      |
; |                            samuel@tulach.cc                             |
; +-------------------------------------------------------------------------+
;
; Input SHA256 : E71D20015B5FD0D1527866B33FD82A3E11E3717E390535852EB7520CC1F62D97
; Input MD5    : DEDBF7CDD9E2083CA4E0FF564A1CA0AD
; Input CRC32  : C304B9E9
; Compiler     : Visual C++

; File Name   : D:\SecureGame\SecureGame\x64\Release\SecureCore.dll
; Format      : Portable executable for AMD64 (PE)
; Imagebase   : 180000000
; Timestamp   : 672FDA64 (Sat Nov 09 21:55:48 2024)
; Section 1. (virtual address 00001000)
; Virtual size                  : 00002130 (   8496.)
; Section size in file          : 00002200 (   8704.)
; Offset to raw data for section: 00000400
; Flags 60000020: Text Executable Readable
; Alignment     : default
; Exported entry   1. GameTick
; PDB File Name : D:\SecureGame\SecureGame\x64\Release\SecureCore.pdb
; OS type         :  MS Windows
; Application type:  DLL

.686p
.mmx
.model flat


; Segment type: Pure code
; Segment permissions: Read/Execute
_text segment para public 'CODE' use64
assume cs:_text
;org 180001000h
assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing



; void *__fastcall GameTick(void *context)
public GameTick
GameTick proc near

var_38= xmmword ptr -38h
var_28= xmmword ptr -28h
var_18= xmmword ptr -18h

context = rcx
push    rbx
sub     rsp, 50h
mov     rbx, context
movaps  [rsp+58h+var_18], xmm6
mov     ecx, cs:?State@Data@@3W4StateId@1@A ; Data::StateId Data::State
movaps  [rsp+58h+var_28], xmm7
test    ecx, ecx
jz      loc_18000122D
```

```
100.00% (543,45) (934,639) 00000400 0000000180001000: GameTick (Synchronized with Hex View-1)
```

```
0000000180001000  40 53 48 83 EC 50 48 8B  D9 0F 29 74 24 40 8B 0D  @SH.......)t$@.
0000000180001010  94 57 00 00 0F 29 7C 24  30 85 C9 0F 84 0C 02 00  .W...)|$0......
0000000180001020  00 F3 0F 10 0D 63 50 00  00 83 F9 01 0F 85 60 02  .....cP.......`
0000000180001030  00 00 80 7B 04 00 0F 57  FF F3 0F 10 23 F3 0F 10  ...{...W.......
0000000180001040  1D AF 3F 00 00 44 0F 29  4C 24 20 74 18 0F 2F CF  ..?..D.)L$·t../
0000000180001050  76 13 0F 28 C4 F3 0F 59  C3 F3 0F 5C C8 F3 0F 11  v..(...Y...\...
0000000180001060  0D 27 50 00 00 80 7B 05  00 F3 0F 10 2D 87 3F 00  .'P...{.......?
0000000180001070  00 74 18 0F 2F E9 76 13  0F 28 C4 F3 0F 59 C3 F3  .t../....(...Y.
0000000180001080  0F 58 C8 F3 0F 11 0D 01  50 00 00 80 7B 06 00 F3  .X......P...{..
0000000180001090  0F 10 15 F1 4F 00 00 74  18 0F 2F D7 76 13 0F 28  .......t../....
00000001800010A0  C4 F3 0F 59 C3 F3 0F 5C  D0 F3 0F 11 15 D7 4F 00  ...Y...\.......
00000001800010B0  00 80 7B 07 00 74 18 0F  2F EA 76 13 0F 28 C4 F3  ..{..t../....(.
00000001800010C0  0F 59 C3 F3 0F 58 D0 F3  0F 11 15 B9 4F 00 00 F3  .Y...X......O..
00000001800010D0  0F 10 1D D9 56 00 00 0F  28 C4 F3 0F 10 35 DE 56  ........(....5.
00000001800010E0  00 00 F3 0F 10 2D CA 56  00 00 F3 44 0F 10 0D ED  ...............
00000001800010F0  3E 00 00 F3 0F 59 C5 F3  0F 59 E6 F3 0F 58 D8 F3  >..............
0000000180001100  0F 58 25 B1 56 00 00 F3  0F 11 1D A1 56 00 00 0F  .X%.V.......V..
0000000180001110  2F FC 0F 28 C4 F3 0F 11  25 9B 56 00 00 F3 41 0F  /..(....%.V....
0000000180001120  58 C1 73 09 0F 2F 05 D1  3E 00 00 72 0F 0F 57 35  X..../.....r..W
0000000180001130  EC 3E 00 00 F3 0F 11 35  84 56 00 00 0F 2F C1 0F  .........V.../.
0000000180001140  28 F4 F3 41 0F 58 F1 72  14 0F 28 C1 F3 0F 58 05  (....X....(...X
0000000180001150  98 3E 00 00 0F 2F C4 72  04 B1 01 EB 02 32 C9 0F  .>.../.........
0000000180001160  2F F2 72 14 0F 28 C2 F3  0F 58 05 7D 3E 00 00 0F  /....(...X.}>..
0000000180001170  2F C4 72 04 B0 01 EB 02  32 C0 F3 0F 10 05 66 3E  /.............f
0000000180001180  00 00 0F 2F C3 72 27 0F  2F 1D 56 3E 00 00 72 1E  .../..'./.V>..r
0000000180001190  84 C9 74 1A 0F 57 2D 85  3E 00 00 0F 28 D8 F3 0F  .....W-.>...(..
00000001800011A0  11 2D 0E 56 00 00 F3 0F  11 05 02 56 00 00 0F 28  .-.V.......V...
00000001800011B0  C3 33 C9 F3 41 0F 58 C1  0F 2F 05 45 3E 00 00 72  ....A.X../.E>..
00000001800011C0  32 F3 0F 10 05 3F 3E 00  00 0F 2F C3 72 25 84 C0  2....?>.../..%.
00000001800011D0  74 21 F3 0F 10 1D 26 3E  00 00 0F 57 2D 3F 3E 00  t!....&>...W-?>
00000001800011E0  00 F3 0F 11 1D C7 55 00  00 F3 0F 11 2D C3 55 00  ...............
00000001800011F0  00 EB 11 0F 2F FB 72 0C  FF 05 BE 55 00 00 89 0D  ..../.r....U...
0000000180001200  A4 55 00 00 0F 28 C3 F3  41 0F 58 C1 44 0F 28 4C  .U...(..A.X...(
0000000180001210  24 20 0F 2F 05 F3 3D 00  00 0F 82 83 00 00 00 FF  $·./...........
0000000180001220  05 87 55 00 00 89 0D 7D  55 00 00 EB 75 C7 05 79  ..U....}U......
0000000180001230  55 00 00 00 40 C4 43 C7  05 77 55 00 00 00 40 92  U...@....wU...@
0000000180001240  43 FF 15 09 2E 00 00 F3  0F 10 3D C1 3D 00 00 F3  C..............
0000000180001250  0F 10 35 99 3D 00 00 A8  01 75 05 0F 28 C6 EB 03  ..5.=....u..(..
0000000180001260  0F 28 C7 F3 0F 11 05 49  55 00 00 FF 15 DF 2D 00  .(.....IU......
0000000180001270  00 A8 01 74 03 0F 28 F7  F3 0F 10 0D 0C 4E 00 00  ...t..(......N.
0000000180001280  F3 0F 11 35 38 55 00 00  C7 05 16 55 00 00 01 00  ....8U.....U...
0000000180001290  00 00 F3 0F 10 1D 16 55  00 00 F3 0F 10 15 E6 4D  .......U.......
00000001800012A0  00 00 8B 05 04 55 00 00  F3 0F 10 05 08 55 00 00  .....U.......U.
00000001800012B0  0F 28 74 24 40 0F 28 7C  24 30 89 43 38 8B 05 F9  .(t$@.(|$0.C8..
00000001800012C0  54 00 00 89 43 3C 33 C0  F3 0F 11 4B 0C F3 0F 11  T...C<3....K...
00000001800012D0  53 1C F3 0F 11 5B 28 F3  0F 11 43 2C C7 43 08 00  S.....(....,...
00000001800012E0  00 A0 41 C7 43 10 00 00  70 41 C7 43 14 00 00 B4  ..A.....pA.....
00000001800012F0  42 C7 43 18 00 40 3F 44  C7 43 20 00 00 70 41 C7  B....@?D..·..pA
0000000180001300  43 24 00 00 B4 42 C7 43  30 00 00 70 41 C7 43 34  C$...B..0..pA..
0000000180001310  00 00 70 41 48 83 C4 50  5B C3 CC CC CC CC CC CC  ..pAH...[......
0000000180001320  B8 01 00 00 00 C3 CC CC  CC CC CC CC CC CC CC CC  ...............
0000000180001330  CC CC CC CC CC CC 66 66  0F 1F 84 00 00 00 00 00  ......ff.......
0000000180001340  48 3B 0D B9 4C 00 00 75  10 48 C1 C1 10 66 F7 C1  H;..L..u.H...f.
0000000180001350  FF FF 75 01 C3 48 C1 C9  10 E9 52 03 00 00 CC CC  ..u............
0000000180001360  48 83 EC 28 85 D2 74 39  83 EA 01 74 28 83 EA 01  H......9....(..
0000000180001370  74 16 83 FA 01 74 0A B8  01 00 00 00 48 83 C4 28  t....t......H..
0000000180001380  C3 E8 5A 04 00 00 EB 05  E8 2B 04 00 00 0F B6 C0  ..Z......+.....
0000000180001390  48 83 C4 28 C3 49 8B D0  48 83 C4 28 E9 0F 00 00  H..............
00000001800013A0  00 4D 85 C0 0F 95 C1 48  83 C4 28 E9 18 01 00 00  .M.............
00000001800013B0  48 89 5C 24 08 48 89 74  24 10 48 89 7C 24 20 41  H.\$.H.t$.H.|$·
00000001800013C0  56 48 83 EC 20 48 8B F2  4C 8B F1 33 C9 E8 CA 04  VH.........3...
00000001800013D0  00 00 84 C0 0F 84 C8 00  00 00 E8 69 03 00 00 8A  ...............
```

```
00000400 0000000180001000: GameTick (Synchronized with IDA View-A)
```

Then we can scan for the function using Cheat Engine.



Found: 1

Address	Value
226E74C1000	40 53 48 83 EC 50 48 8B D9 0F...

[✓] Hex Value: 40 53 48 83 EC 50 48 8B D9 0F 29 74 24 40 8B 0D
Scan Type: Search for this array
Value Type: Array of byte
Memory Scan Options: All
Unrandomizer
Enable Speedhack

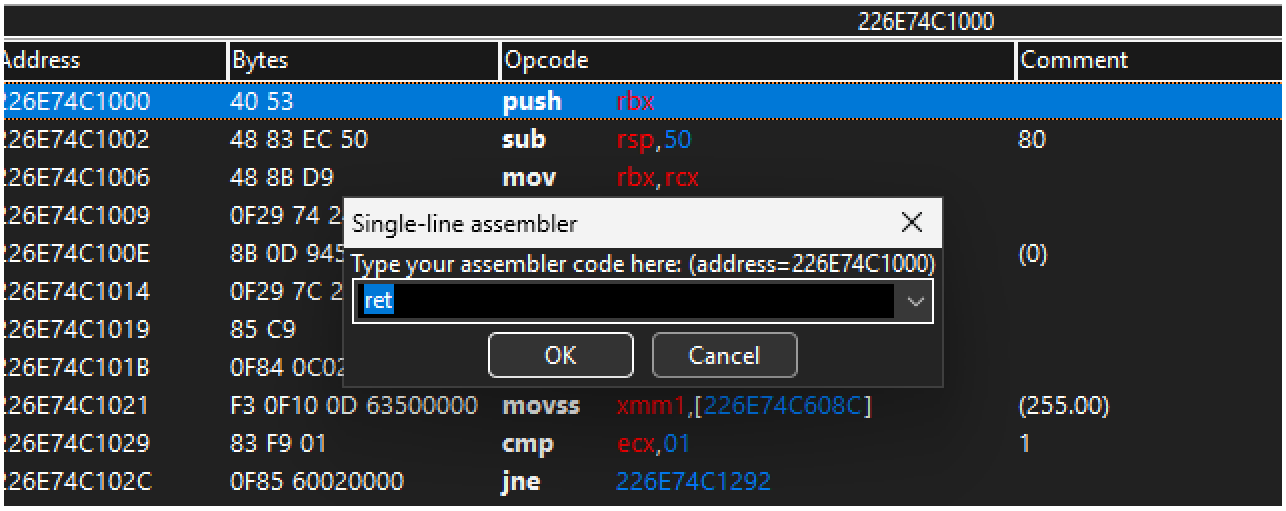
226E74C1000

Address	Bytes	Opcode		Comment
226E74C1000	40 53	push	rbx	
226E74C1002	48 83 EC 50	sub	rsp,50	80
226E74C1006	48 8B D9	mov	rbx,rcx	
226E74C1009	0F29 74 24 40	movaps	[rsp+40],xmm6	
226E74C100E	8B 0D 94570000	mov	ecx,[226E74C67A8]	(0)
226E74C1014	0F29 7C 24 30	movaps	[rsp+30],xmm7	
226E74C1019	85 C9	test	ecx,ecx	
226E74C101B	0F84 0C020000	je	226E74C122D	
226E74C1021	F3 0F10 0D 63500000	movss	xmm1,[226E74C608C]	(255.00)
226E74C1029	83 F9 01	cmp	ecx,01	1
226E74C102C	0F85 60020000	jne	226E74C1292	
226E74C1032	80 7B 04 00	cmp	byte ptr [rbx+04],00	0

push word or doubleword onto the stack

```
Protect:Read/Write  AllocationBase=7FF6178E0000 Base=7FF6178E5000 Size=1000 Module=SecureGame.exe
address      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 0123456789ABCDEF01234567
7FF6178E5000 01 00 00 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ...  ....................
7FF6178E5018 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
7FF6178E5030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 A2 FA 93 4D 00 76 00 00 ................   M.v..
7FF6178E5048 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
7FF6178E5060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
7FF6178E5078 00 00 00 00 00 00 00 00 5D 05 6C B2 FF 89 FF FF FF FF FF FF 01 00 00 00 ........].l         ....
7FF6178E5090 C8 FF FF FF FF FF FF FF 06 00 00 00 6E 00 00 00 00 00 08 00 00 00 00 00         ....n...........
7FF6178E50A8 00 00 00 02 00 00 00 00 01 00 00 00 75 98 00 00 00 00 00 00 00 00 00 00 ............u ..........
7FF6178E50C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
7FF6178E50D8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
7FF6178E50F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
7FF6178E5108 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
7FF6178E5120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
7FF6178E5138 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
7FF6178E5122 : byte: 0 word: 0 integer: 0 int64: 0 float:0.00 double: 0.00
```

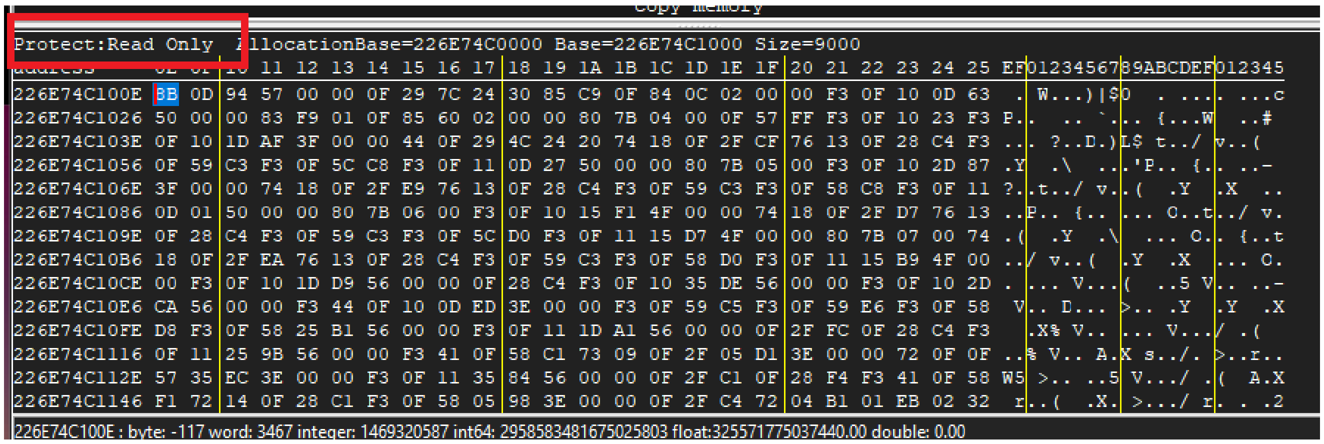
Let’s try to patch it by putting a return (0xC3) at the start of the function.



```
                                                    226E74C1000
Address        Bytes               Opcode                          Comment
226E74C1000    40 53               push    rbx
226E74C1002    48 83 EC 50         sub     rsp,50                  80
226E74C1006    48 8B D9            mov     rbx,rcx
226E74C1009    0F29 74 2
226E74C100E    8B 0D 945                                           (0)
226E74C1014    0F29 7C 2
226E74C1019    85 C9
226E74C101B    0F84 0C02
226E74C1021    F3 0F10 0D 63500000 movss   xmm1,[226E74C608C]      (255.00)
226E74C1029    83 F9 01            cmp     ecx,01                  1
226E74C102C    0F85 60020000       jne     226E74C1292
```

```
Single-line assembler
Type your assembler code here: (address=226E74C1000)
ret
```

Aaaaand nothing. So what’s going on? Well, first of all, let’s check the memory protection.



```
Protect:Read Only  AllocationBase=226E74C0000 Base=226E74C1000 Size=9000
                        11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 EF0123456789ABCDEF012345
226E74C100E BB 0D 94 57 00 00 0F 29 7C 24 30 85 C9 0F 84 0C 02 00 00 F3 0F 10 0D 63   . W...)|$0  . .... ...c
226E74C1026 50 00 00 83 F9 01 0F 85 60 02 00 00 80 7B 04 00 0F 57 FF F3 0F 10 23 F3 P..   .. `... {...W  ..#
226E74C103E 0F 10 1D AF 3F 00 00 44 0F 29 4C 24 20 74 18 0F 2F CF 76 13 0F 28 C4 F3 ... ?..D.)L$ t../ v..(
226E74C1056 0F 59 C3 F3 0F 5C C8 F3 0F 11 0D 27 50 00 00 80 7B 05 00 F3 0F 10 2D 87 .Y  .\  ...'P.. {.. ..-
226E74C106E 3F 00 00 74 18 0F 2F E9 76 13 0F 28 C4 F3 0F 59 C3 F3 0F 58 C8 F3 0F 11 ?..t../ v..(  .Y  .X  ..
226E74C1086 0D 01 50 00 00 80 7B 06 00 F3 0F 10 15 F1 4F 00 00 74 18 0F 2F D7 76 13 ..P.. {.. ... O..t../ v.
226E74C109E 0F 28 C4 F3 0F 59 C3 F3 0F 5C D0 F3 0F 11 15 D7 4F 00 00 80 7B 07 00 74 .(  .Y  .\  ... O.. {..t
226E74C10B6 18 0F 2F EA 76 13 0F 28 C4 F3 0F 59 C3 F3 0F 58 D0 F3 0F 11 15 B9 4F 00 ../ v..(  .Y  .X  ... O.
226E74C10CE 00 F3 0F 10 1D D9 56 00 00 0F 28 C4 F3 0F 10 35 DE 56 00 00 F3 0F 10 2D . .... V...(  ..5 V.. ..-
226E74C10E6 CA 56 00 00 F3 44 0F 10 0D ED 3E 00 00 F3 0F 59 C5 F3 0F 59 E6 F3 0F 58  V.. D... >.. .Y  .Y  .X
226E74C10FE D8 F3 0F 58 25 B1 56 00 00 F3 0F 11 1D A1 56 00 00 0F 2F FC 0F 28 C4 F3   .X% V.. ... V.../ .(
226E74C1116 0F 11 25 9B 56 00 00 F3 41 0F 58 C1 73 09 0F 2F 05 D1 3E 00 00 72 0F 0F ..% V.. A.X s../. >..r..
226E74C112E 57 35 EC 3E 00 00 F3 0F 11 35 84 56 00 00 0F 2F C1 0F 28 F4 F3 41 0F 58 W5 >.. ..5 V.../ .(  A.X
226E74C1146 F1 72 14 0F 28 C1 F3 0F 58 05 98 3E 00 00 0F 2F C4 72 04 B1 01 EB 02 32  r..(  .X. >.../ r. .  .2
226E74C100E : byte: -117 word: 3467 integer: 1469320587 int64: 2958583481675025803 float:325571775037440.00 double: 0.00
```

It’s actually just read-only. No execution permissions. That’s because this is just a dummy image (most likely to prevent memory address conflicts). There’s really no way we could mess around with the code or data in the enclave with anything we launch from within the OS.

Well, is there anything we **can** do then? Yes, obviously. Anything that’s outside of the enclave is easily accessible. We can hook into the host process and edit the data returned by or sent to the enclave. We can also just inject Cheat Engine’s speedhack DLL (which just hooks performance counters) and that will work too, since all the time calculations are done in the host process.

Conclusion

Due to the limitations mentioned above, it would take incredible effort to implement VBS enclaves into an actual game engine in a way that would be meaningful and not completely obliterate the game’s performance. As such, I don’t really see any game developers using them, not even accounting for the system requirements (Windows 11+, VBS enabled) since gamers might not be willing to reconfigure their systems just to play some game.

On top of that, while it would prevent any attempt to manipulate the game from programs or drivers loaded in the OS, experienced developers would have no issues getting around these restrictions by writing firmware apps that would manipulate the entire Windows bootchain.

There are two things that Microsoft could do that would instantly eliminate the vast majority of game cheating:

1. Work with OEMs to enforce strict boot environment code signing policies. Secure Boot is not sufficient (1, 2). The system would have to verify everything from the moment it's turned on until it's turned off. If it allows loading any user-created firmware code, it's over. However, being this strict would essentially mean locking the system down to Windows only, or at least making it very difficult to install alternative OSes (Linux), which no sane person can support.
2. While this would require enormous effort, they could introduce whole "process enclaves" where an entire process could run in a separate virtualized environment while still having access to standard NT APIs. I've read several security-related blog posts from them and I feel this sort of containerization is something they're aiming for, so we'll see.

Thanks for reading and have a nice day.