Damocles

Shrapnel Security Analysis

2024.05.21

Senna

DAMOCLES LABS

Damocles

Contents

一、 Summary

As an FPS game, Shrapnel has a security level of 0 in its client. Due to the high demand for fairness in the STG genre, the loss of fairness can disrupt the overall balance of Sigma's material output, leading to a situation where winners dominate. Based on the available information, the game appears to have integrated an AI anti-cheat solution. However, after testing conducted by Damocles, it was found that the current solution fails to detect aimbot cheating. Therefore, Damocles has assigned a safety rating of 1 star to the game..

Security Rating： ★☆☆☆☆

二、 Game Background

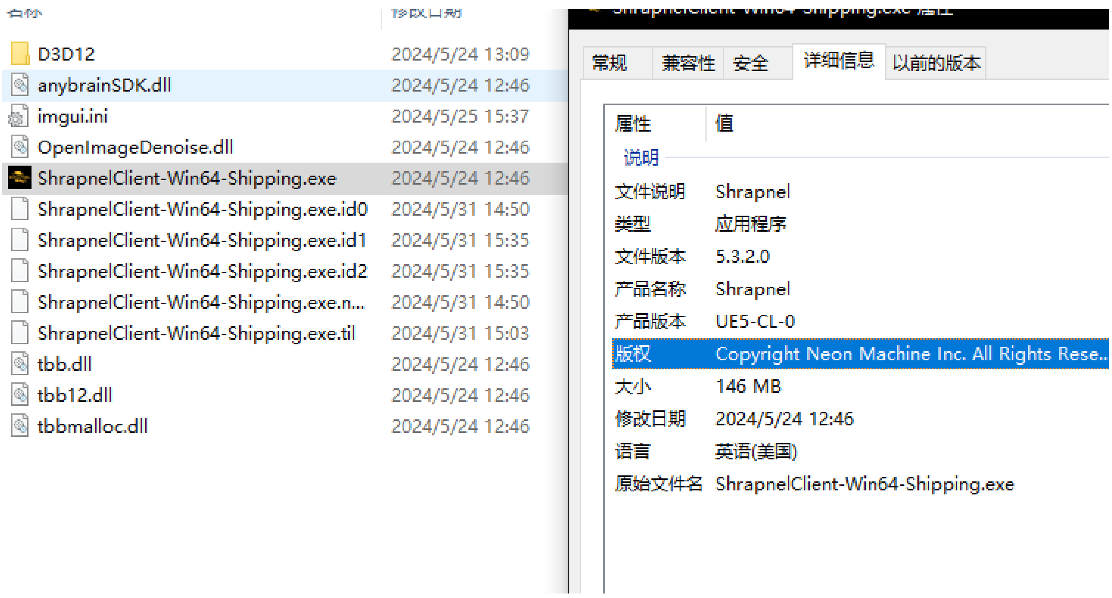
- Game Version：ST3
- Genres & Engine：FPS，UE5
- Possible Issues in Gameplay：
  - Unauthorized movement (modifying local character attributes for speed enhancement).
  - No recoil
  - Aimbot
  - Teleportation
  - Settlement replay attack
  - Modification of local character attributes, such as jumping
  - Wallhacks (ability to see through walls)

三、 Game Security Analysis

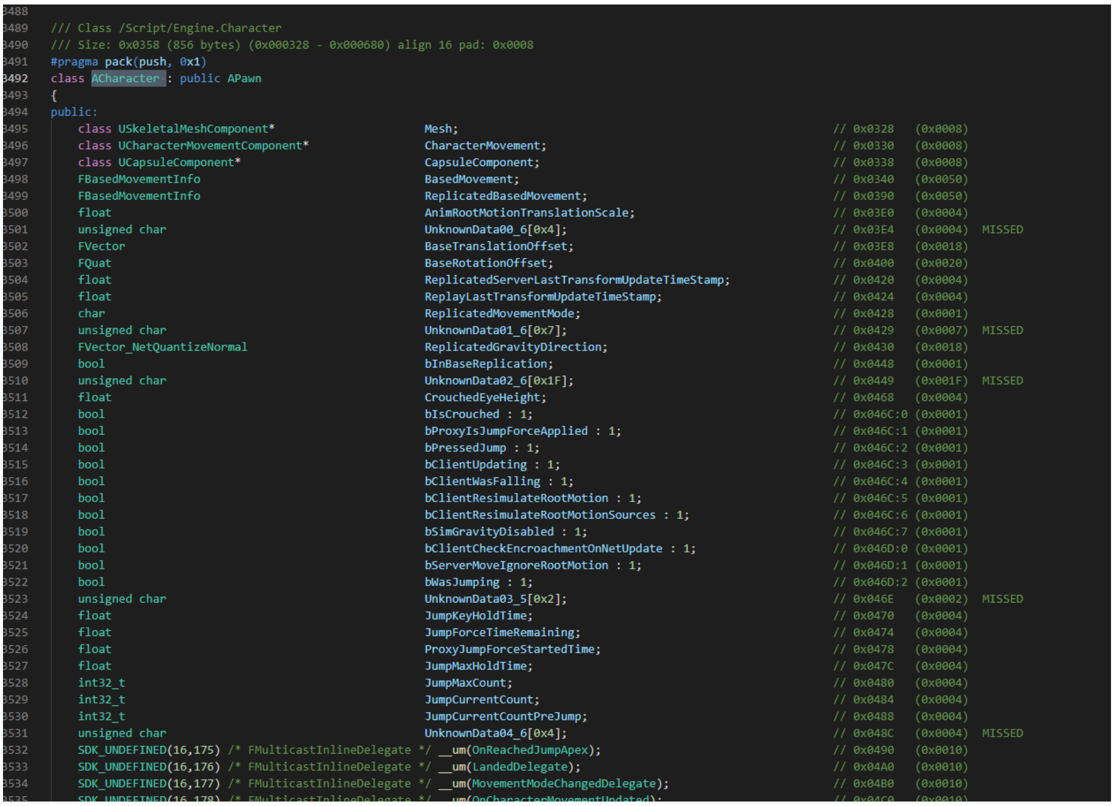
Game Code Protection：

Analysis Process：

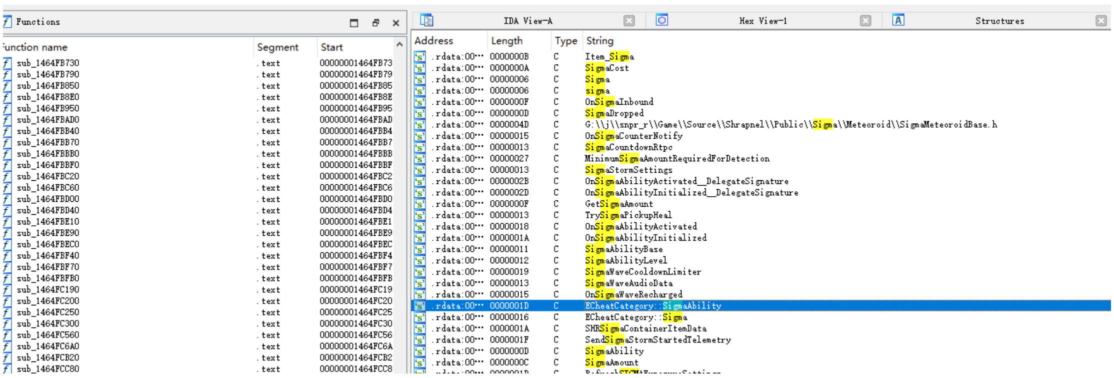
1. Since different engines have different analysis modes, it is important to determine the game engine used after obtaining the game EXE. By analyzing the basic information of the game, we can determine that this game was developed using UE5。



2. Using tools to dump the structure of UE (Unreal Engine) characters for fast positioning. Once located, indexing and modification can be done through UE's unique linked list structure.



During the static code analysis using IDA, it was discovered that the code contains easily identifiable string information. Additionally, the code is not encrypted, allowing for clear analysis.



```
void __fastcall sub_1464E4300(AK::MemoryMgr *a1, _QWORD *a2)
{
  unsigned __int64 v4; // rbx
  __int64 v5; // r15
  unsigned __int64 v6; // rbp
  unsigned __int64 *v7; // rcx
  unsigned __int64 v8; // rdx
  _QWORD *v9; // rsi
  _QWORD *v10; // rax
  unsigned __int64 i; // rdi
  __int64 v12; // rax
  void (__fastcall ***v13)(_QWORD); // rcx
  __int64 v14; // rdx
  __int64 v15; // rax
  __int64 v16; // rcx
  unsigned __int64 v17; // [rsp+60h] [rbp+8h] BYREF
  __int64 v18; // [rsp+70h] [rbp+18h] BYREF

  v4 = 0i64;
  LODWORD(v17) = 0;
  AK::MemoryMgr::StartProfileThreadUsage(a1);
  v5 = *((_QWORD *)a1 - 1);
  if ( *(_QWORD *)v5 >= 0x70ui64 && *(_QWORD *)(v5 + 104) )
  {
    v18 = (__int64)(a2[1] - *a2) >> 3;
    v6 = v18;
    v17 = (*(__int64 (__fastcall **)(AK::MemoryMgr *))(*(_QWORD *)a1 + 56i64))(a1);
    v7 = (unsigned __int64 *)&v18;
    if ( v17 >= v6 )
      v7 = &v17;
    v8 = *v7;
    v17 = v8;
    v9 = 0i64;
    if ( v8 )
    {
      v10 = (_QWORD *)sub_14487FF10(saturated_mul(v8, 8ui64));
      v9 = v10;
      if ( v10 )
      {
        memset(v10, 0, 8 * v17);
        if ( v6 )
        {
          for ( i = 0i64; i < v6; ++i )
          {
            v12 = *(_QWORD *)(*a2 + 8 * i);
```

Therefore, it is possible to combine the dumped UE data structures with the use of decompilation tools to gain a basic understanding of the game's code logic.

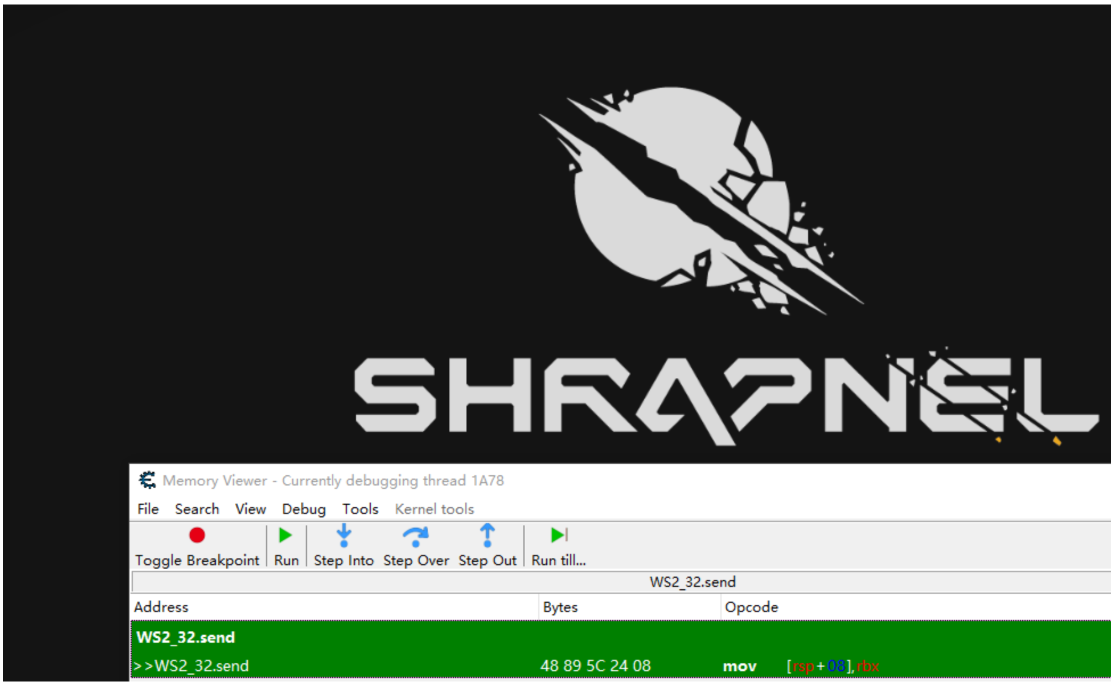
Analysis Conclusion:

Shrapnel scores 0 points in terms of game code protection. The client code lacks any form of protection. Although the game developers have made modifications to the UE engine source code, it only slows down the analysis progress. With the combination of the SDK, it is still possible to analyze the game's combat and Sigma settlement logic.

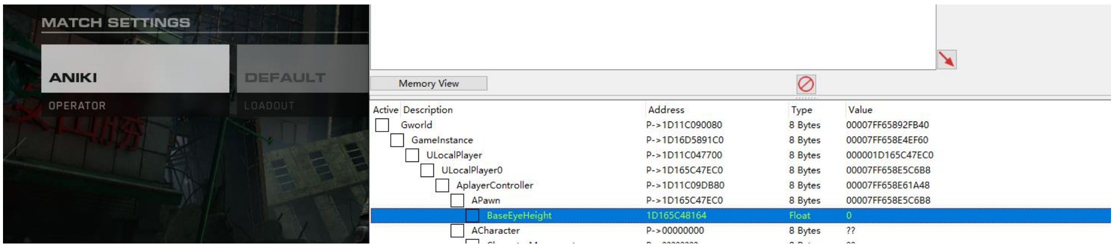
Game Basic Anti-Cheat:

Analysis Process:

1. In terms of basic anti-cheat detection, we primarily determine whether the game loads and executes external logic by replacing Lua files.
2. While attaching with Cheat Engine (CE) in the game's open state and setting breakpoints on common functions, it was observed that the game did not exit or provide any prompts..



3. It is possible to directly modify in-game character attributes such as speed, jump height, and base eye height. These attribute modifications can have a direct impact on shooting judgments during raycasting. Furthermore, it appears that the game server does not initiate any kicking actions in response to these modifications..



Analysis Conclusion:

1. Shrapnel's anti-cheat measures are lacking in terms of countering dynamic debugging and dynamic analysis. This makes it relatively easy for malicious players to engage in cheating activities with low cost. Additionally, the game lacks the capability to detect players who are already cheating. Based on the available public information regarding the AI anti-cheat solution, it appears to rely on KDA (Kill-Death-Assist ratio) for determination. However, for games that have limited player match records and rely on Play to Airdrop to attract players, this anti-cheat solution proves to be ineffective. It is recommended to consider integrating EAC (Easy Anti-Cheat) as a solution.
2. The reason for focusing only on anti-debugging and read/write protection testing is that for a cheat program, finding data and implementing desired functionalities can be achieved through debugging and memory manipulation. If the most fundamental protection measures in these two aspects are missing, other detection methods such as code injection and hooking become meaningless.

Game Logic Issues

Analysis Process:

During our analysis of Shrapnel, we have identified issues with incomplete data synchronization. Additionally, both the client and server lack adequate anti-cheat mechanisms. Based on this, we have expanded our analysis to include an examination of game logic issues and an introduction to cheat analysis principles.

Shrapnel utilizes the Unreal Engine, and the implementation logic for wallhacks (ESP) and aimbots can be templated. The implementation primarily relies on character-related attributes, specifically:

Attributes to modify for aimbot: APlayerController->Apawn->FRotation {Pitch, Yaw, Roll}

Attributes to retrieve for wallhacks: APlayerController->Apawn->FLocation {X, Y, Z}

Once the memory addresses for these data are obtained, the cheat program performs matrix transformations to convert the two-dimensional data into three-dimensional data. After calculations are performed by the cheat program, modifications are made in the game to achieve aimbot or wallhack functionality. It is worth noting that FPS games developed using the Unreal Engine often have templates for bullet tracking. Therefore, we recommend that the project team pays attention to detection in this aspect in the future.

Taking Apex, which uses the Source Engine, as an example, the logic is similar, as shown below:

Damocles

```
static void EspLoop()
{
	esp_t = true;
	while(esp_t)
	{
		std::this_thread::sleep_for(std::chrono::milliseconds(1));
		while(g_Base!=0 && c_Base!=0)
		{
			std::this_thread::sleep_for(std::chrono::milliseconds(1));
			if (esp)
			{
				valid = false;

				uint64_t LocalPlayer = 0;
				apex_mem.Read<uint64_t>(g_Base + OFFSET_LOCAL_ENT, LocalPlayer);
				if (LocalPlayer == 0)
				{
					next = true;
					while(next && g_Base!=0 && c_Base!=0 && esp)
					{
						std::this_thread::sleep_for(std::chrono::milliseconds(1));
					}
					continue;
				}
				Entity LPlayer = getEntity(LocalPlayer);
				int team_player = LPlayer.getTeamId();
				if (team_player < 0 || team_player>50)
				{
					next = true;
					while(next && g_Base!=0 && c_Base!=0 && esp)
					{
						std::this_thread::sleep_for(std::chrono::milliseconds(1));
					}
					continue;
				}
				Vector LocalPlayerPosition = LPlayer.getPosition();

				uint64_t viewRenderer = 0;
				apex_mem.Read<uint64_t>(g_Base + OFFSET_RENDER, viewRenderer);
				uint64_t viewMatrix = 0;
				apex_mem.Read<uint64_t>(viewRenderer + OFFSET_MATRIX, viewMatrix);
				Matrix m = {};
				apex_mem.Read<Matrix>(viewMatrix, m);

				uint64_t entitylist = g_Base + OFFSET_ENTITYLIST;

				memset(players,0,sizeof(players));
				if(firing_range)
				{
					int c=0;
					for (int i = 0; i < 10000; i++)
					{
						uint64_t centity = 0;
```

Analysis Conclusion:

1. For a game, the security of its local logic is closely related to the judgment and local security measures of the game server (GS). Based on the current performance of the game, the GS lacks control over synchronized data, and the synchronization of data is incomplete.

Additionally, there are missing user data reports. Therefore, the security rating in this aspect is 0.

2. Due to the similarity in raycasting logic used by Unreal Engine, malicious users can easily implement features such as bullet tracking, which enable them to achieve instant kill functionality

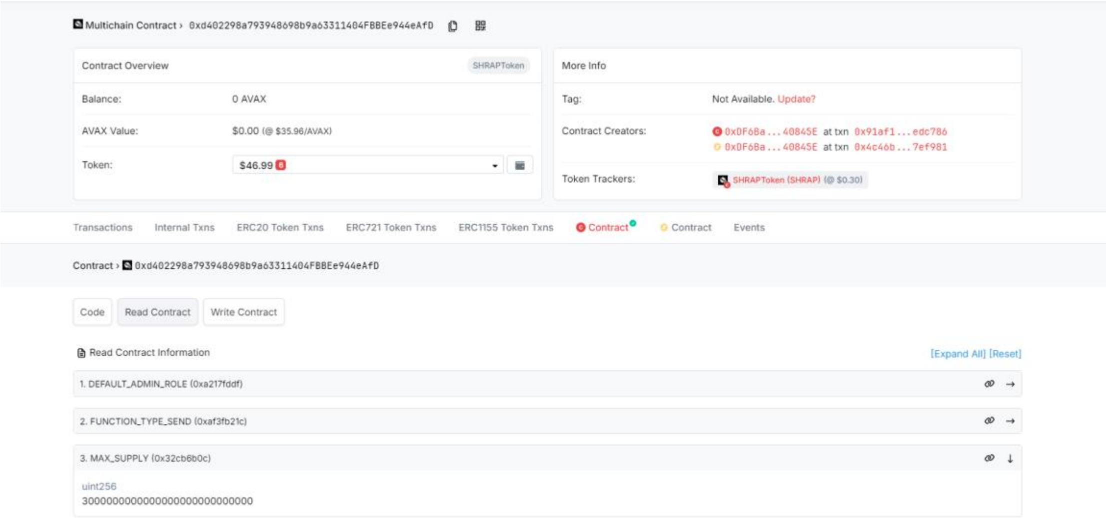
Game Protocol & Server Security Analysis

The current game protocol is relatively limited, with the main focus being on match settlement. Therefore, we will temporarily exclude the analysis of this particular aspect

WEB3 Security Analysis：

Shrap is an ERC20 token issued on the Avax network with a total supply of 3 billion tokens. The token contract code is relatively simple, and in addition to Avax, Shrap tokens also have support for Ethereum (ETH) and Binance Smart Chain (BSC).

The current token contract has limited functionality, and there is a restriction on the total token supply, which has already been fully minted. As a result, the contract carries a relatively low security risk.



About Damocles

Damocles Labs is a security team established in 2023, specializing in security for the Web3 industry. Their services include contract code auditing, business code auditing, penetration testing, GameFi code auditing, GameFi vulnerability discovery, GameFi cheat analysis, and GameFi anti-cheat measures. They are committed to making continuous efforts in the Web3 security industry, producing as many analysis reports as possible, raising awareness among project owners and users about GameFi security, and promoting the overall security development of the industry.。

Twitter: https://twitter.com/DamoclesLabs

Discord: https://discord.gg/xd6H6eqFHz