



# Shrapnel 游戏分析报告

2024.05.31

Senna

DAMOCLES LABS

# 目录

- 概要
- 游戏背景
  - ◆ 游戏版本
  - ◆ 游戏类型&游戏引擎
  - ◆ 游戏玩法可能存在的问题
- 游戏安全分析
  - ◆ 游戏代码保护
  - ◆ 游戏基础反作弊
  - ◆ 游戏逻辑问题&外挂原理分析
  - ◆ 游戏协议&Server 安全性分析
- Web3 安全分析
  - ◆ 代币合约安全
  - ◆ 游戏内经济系统安全
- 关于 Damocles

## 一、 概要

作为一款 FPS 品类的游戏 Shrapnel 在其客户端的安全性为 0，由于 STG 品类的游戏对公平性要求极高，如果公平性丧失，对于普通用户的游戏体验，整体的 Sigma 物质产出将会失衡，造成赢家通吃的场面。通过开放信息来看该游戏接入某 AI 反作弊方案，但是经 Damocles 测试以后发现当前方案并未能检测出自瞄作弊，因此 Damocles 判定其安全评分为 1 星。

安全性评分： ★ ☆ ☆ ☆ ☆

## 二、 游戏背景

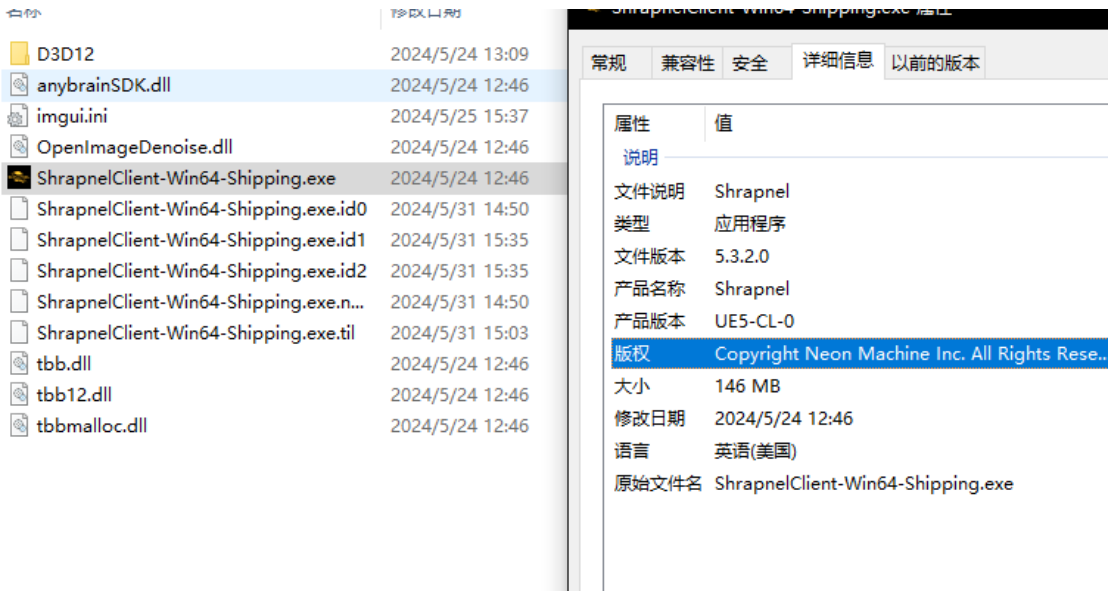
- 进行评估的游戏版本：STX3
- 游戏类型&游戏引擎：STG&FPS，UE5.3
- 游戏玩法可能存在的问题：
  - 非法移动(修改本地人物属性进行加速)
  - 无后坐力
  - 自瞄
  - 瞬移
  - 结算重放攻击
  - 跳跃等一些本地人物属性的修改
  - 透视
  - 秒杀

### 三、 游戏安全性分析

#### 游戏代码保护：

##### 分析过程：

1. 由于不同的引擎有不同的分析模式, 所以在获取到游戏 EXE 后首先需要确定游戏使用的引擎, 通过对游戏基础信息识别我们可以确定该游戏是使用 UE5 进行开发。



2. 通过工具进行 dump UE 的人物结构进行快速定位, 定位以后通过 UE 特有的链表结构进行索引与修改

```

3488
3489 /// Class /Script/Engine.Character
3490 /// Size: 0x0358 (856 bytes) (0x000328 - 0x000680) align 16 pad: 0x0008
3491 #pragma pack(push, 0x1)
3492 class ACharacter : public APawn
3493 {
3494 public:
3495     class USkeletalMeshComponent* Mesh; // 0x0328 (0x0008)
3496     class UCharacterMovementComponent* CharacterMovement; // 0x0330 (0x0008)
3497     class UCapsuleComponent* CapsuleComponent; // 0x0338 (0x0008)
3498     FBasedMovementInfo BasedMovement; // 0x0340 (0x0050)
3499     FBasedMovementInfo ReplicatedBasedMovement; // 0x0390 (0x0050)
3500     float AnimRootMotionTranslationScale; // 0x03E0 (0x0004)
3501     unsigned char UnknownData00_6[0x4]; // 0x03E4 (0x0004) MISSED
3502     FVector BaseTranslationOffset; // 0x03E8 (0x0018)
3503     FQuat BaseRotationOffset; // 0x0400 (0x0020)
3504     float ReplicatedServerLastTransformUpdateTimeStamp; // 0x0420 (0x0004)
3505     float ReplayLastTransformUpdateTimeStamp; // 0x0424 (0x0004)
3506     char ReplicatedMovementMode; // 0x0428 (0x0001)
3507     unsigned char UnknownData01_6[0x7]; // 0x0429 (0x0007) MISSED
3508     FVector_NetQuantizeNormal ReplicatedGravityDirection; // 0x0430 (0x0018)
3509     bool bInBaseReplication; // 0x0448 (0x0001)
3510     unsigned char UnknownData02_6[0x1F]; // 0x0449 (0x001F) MISSED
3511     float CrouchedEyeHeight; // 0x0468 (0x0004)
3512     bool bIsCrouched : 1; // 0x046C:0 (0x0001)
3513     bool bProxyIsJumpForceApplied : 1; // 0x046C:1 (0x0001)
3514     bool bPressedJump : 1; // 0x046C:2 (0x0001)
3515     bool bClientUpdating : 1; // 0x046C:3 (0x0001)
3516     bool bClientWasFalling : 1; // 0x046C:4 (0x0001)
3517     bool bClientResimulateRootMotion : 1; // 0x046C:5 (0x0001)
3518     bool bClientResimulateRootMotionSources : 1; // 0x046C:6 (0x0001)
3519     bool bSimGravityDisabled : 1; // 0x046C:7 (0x0001)
3520     bool bClientCheckEncroachmentOnNetUpdate : 1; // 0x046D:0 (0x0001)
3521     bool bServerMoveIgnoreRootMotion : 1; // 0x046D:1 (0x0001)
3522     bool bWasJumping : 1; // 0x046D:2 (0x0001)
3523     unsigned char UnknownData03_5[0x2]; // 0x046E (0x0002) MISSED
3524     float JumpKeyHoldTime; // 0x0470 (0x0004)
3525     float JumpForceTimeRemaining; // 0x0474 (0x0004)
3526     float ProxyJumpForceStartTime; // 0x0478 (0x0004)
3527     float JumpMaxHoldTime; // 0x047C (0x0004)
3528     int32_t JumpMaxCount; // 0x0480 (0x0004)
3529     int32_t JumpCurrentCount; // 0x0484 (0x0004)
3530     int32_t JumpCurrentCountPreJump; // 0x0488 (0x0004)
3531     unsigned char UnknownData04_4[0x4]; // 0x048C (0x0004) MISSED
3532     SDK_UNDEFINED(16,175) /* FMulticastInlineDelegate */ _um(OnReachedJumpApex); // 0x0490 (0x0010)
3533     SDK_UNDEFINED(16,176) /* FMulticastInlineDelegate */ _um(LandedDelegate); // 0x04A0 (0x0010)
3534     SDK_UNDEFINED(16,177) /* FMulticastInlineDelegate */ _um(MovementModeChangedDelegate); // 0x04B0 (0x0010)
3535     SDK_UNDEFINED(16,178) /* FMulticastInlineDelegate */ _um(OnCharacterMovementStarted); // 0x04C0 (0x0010)

```

并且使用 IDA 进行静态代码分析时发现，代码字符串信息明显，同时代码未

加密，可以很清晰的分析

Function name	Segment	Start	Address	Length	Type	String
sub_146AFB70	.text	0000000146AFB73	rdata:000000000000000B	C	Item_Sigma	
sub_146AFB70	.text	0000000146AFB70	rdata:000000000000000A	C	SigmaCost	
sub_146AFB80	.text	0000000146AFB80	rdata:0000000000000006	C	Sigma	
sub_146AFB90	.text	0000000146AFB95	rdata:000000000000000F	C	OnSigmaInbound	
sub_146AFBA0	.text	0000000146AFBA0	rdata:000000000000000B	C	SigmaDropped	
sub_146AFBB0	.text	0000000146AFBB0	rdata:0000000000000015	C	G:\V\lmpg_r\Game\Source\Shrapnel\Public\Sigma\Meteoroid\SigmaMeteoroidBase.h	
sub_146AFBB0	.text	0000000146AFBB7	rdata:0000000000000013	C	OnSigmaCounterNotify	
sub_146AFBB0	.text	0000000146AFBB7	rdata:0000000000000013	C	SigmaCountdownType	
sub_146AFBB0	.text	0000000146AFBB7	rdata:0000000000000027	C	MinimumSigmaCountdownRequiredForDetection	
sub_146AFBC0	.text	0000000146AFBC0	rdata:000000000000002B	C	OnSigmaAbilityActivated_DelegateSignature	
sub_146AFBC0	.text	0000000146AFBC0	rdata:000000000000002F	C	OnSigmaAbilityInitialized_DelegateSignature	
sub_146AFBD0	.text	0000000146AFBD0	rdata:0000000000000013	C	TrySigmaPickupMeal	
sub_146AFBE0	.text	0000000146AFBE0	rdata:0000000000000018	C	OnSigmaAbilityActivated	
sub_146AFBF0	.text	0000000146AFBF0	rdata:000000000000001A	C	OnSigmaAbilityInitialized	
sub_146AFD00	.text	0000000146AFD00	rdata:0000000000000011	C	SigmaAbilityBase	
sub_146AFD10	.text	0000000146AFD10	rdata:0000000000000012	C	SigmaAbilityLevel	
sub_146AFD20	.text	0000000146AFD20	rdata:0000000000000019	C	SigmaAwareCooldownLimit	
sub_146AFD30	.text	0000000146AFD30	rdata:0000000000000013	C	SigmaAwareCooldownData	
sub_146AFD40	.text	0000000146AFD40	rdata:0000000000000015	C	OnSigmaAwareRecharged	
sub_146AFD50	.text	0000000146AFD50	rdata:0000000000000011	C	OnSigmaCategory_SigmaAbility	
sub_146AFD60	.text	0000000146AFD60	rdata:0000000000000016	C	OnSigmaCategory_SigmaAbility	
sub_146AFD70	.text	0000000146AFD70	rdata:000000000000001A	C	OnSigmaContainerItemData	
sub_146AFD80	.text	0000000146AFD80	rdata:000000000000001F	C	OnSigmaContainerItemData	
sub_146AFD90	.text	0000000146AFD90	rdata:000000000000000B	C	SigmaAbility	
sub_146AFDA0	.text	0000000146AFDA0	rdata:000000000000000C	C	SigmaAmount	
sub_146AFDB0	.text	0000000146AFDB0	rdata:000000000000000A	C	SigmaAmount	

```

1 void __fastcall sub_1464E4300(AK::MemoryMgr *a1, _QWORD *a2)
2 {
3     unsigned __int64 v4; // rbx
4     __int64 v5; // r15
5     unsigned __int64 v6; // rbp
6     unsigned __int64 *v7; // rcx
7     unsigned __int64 v8; // rdx
8     _QWORD *v9; // rsi
9     _QWORD *v10; // rax
10    unsigned __int64 i; // rdi
11    __int64 v12; // rax
12    void (__fastcall **v13)(_QWORD); // rcx
13    __int64 v14; // rdx
14    __int64 v15; // rax
15    __int64 v16; // rcx
16    unsigned __int64 v17; // [rsp+60h] [rbp+8h] BYREF
17    __int64 v18; // [rsp+70h] [rbp+18h] BYREF
18
19    v4 = 0i64;
20    LODWORD(v17) = 0;
21    AK::MemoryMgr::StartProfileThreadUsage(a1);
22    v5 = *((_QWORD *)a1 - 1);
23    if ( *((_QWORD *)v5) >= 0x70ui64 && *((_QWORD *)v5 + 104) )
24    {
25        v18 = (__int64)(a2[1] - *a2) >> 3;
26        v6 = v18;
27        v17 = (*(__int64 (__fastcall **)(AK::MemoryMgr *)))(*(__QWORD *)a1 + 56i64))(a1);
28        v7 = (unsigned __int64 *)&v18;
29        if ( v17 >= v6 )
30            v7 = &v17;
31        v8 = *v7;
32        v17 = v8;
33        v9 = 0i64;
34        if ( v8 )
35        {
36            v10 = (__QWORD *)sub_14487FF10(saturated_mul(v8, 8ui64));
37            v9 = v10;
38            if ( v10 )
39            {
40                memset(v10, 0, 8 * v17);
41                if ( v6 )
42                {
43                    for ( i = 0i64; i < v6; ++i )
44                    {
45                        v12 = *((_QWORD *)a2 + 8 * i);
46                    }
47                }
48            }
49        }
50    }
51 }

```

因此可以结合 dump 的 UE 数据结构使用反编译工具对游戏代码逻辑进行基本的理解。

## 分析结论：

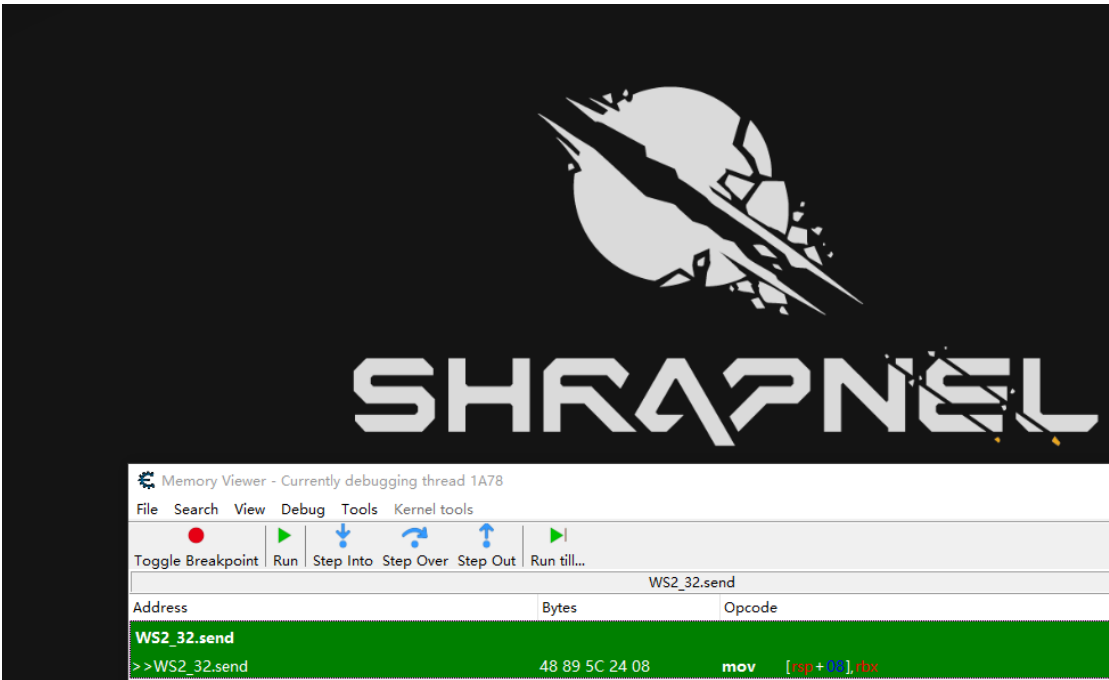
Shrapnel 在游戏代码保护方面得分为 0 分，其 client 代码没有任何保护，虽然游戏开发者针对 UE 引擎源码有过修改，但是只能拖慢分析进度，结合 SDK 依然可以分析出游戏的对战以及 Sigma 结算逻辑。

## 游戏基础反作弊：

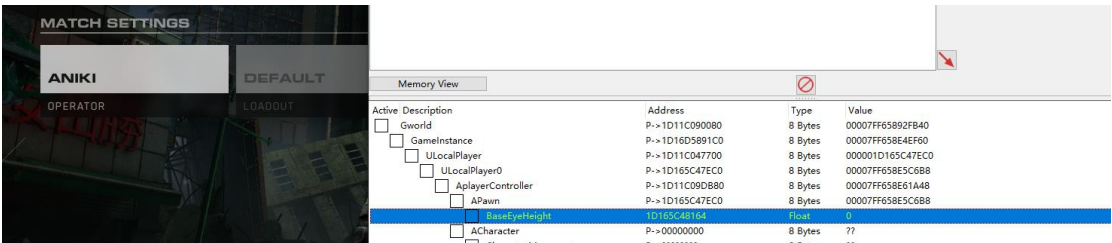
### 分析过程：

1. 在基础反作弊检测方面，我们主要从两个方面进行测试，一个是游戏是否存在反调试，另一个是游戏是否存在读写保护。
2. 在游戏打开状态下使用 CE 进行附加，并且对通用函数进行下断点，发现游戏

并没有退出，或者提示



3. 可以直接修改游戏内的人物属性例如速度、跳跃高度、基础眼睛高度（该属性在射线检测时偶合会用来做射击判断，修改为适当的值则直接影响判定）等属性，并且 GS 不会进行踢出操作。



分析结论：

1. Shrapnel 在反作弊对抗上基本保护为 0，缺少针对动态调试，动态分析的对抗，因为对于想作恶的玩家来说成本很低，并且缺少对已经作弊的玩家的检测能力,根据公开资料中提到的 AI 反作弊方案来看，其原理应该是通过 KDA 来判定，但是对于玩家对局记录少，通过举行 Play to airdrop 来获客的游戏来

说，该反作弊方案无异于 0，建议接入 EAC。

2. 只测试反调试和读写保护两个方面的原因是对于一块外挂来说，找数据与实现功能只需要通过调试和读写就可以实现。如果最基础的两个保护能力都缺失的话，那么一些注入、hook 等检测也毫无意义。

## 游戏逻辑问题&外挂原理分析

### 分析过程：

在对游戏进行分析时我们发现 Shrapnel 存在数据同步不完善的问题，同时本地与服务端均缺少对抗作弊的方案，基于此，我们针对游戏逻辑问题的分析进行扩充，引入外挂原理分析。

Shrapnel 使用的引擎为 Unreal Engine，在该引擎中的透视与自瞄的实现逻辑是可以套用模板的，其实现主要依赖人物相关的属性，具体为：

自瞄需要修改的属性：APlayerController->Apawn->FRotation { Pitch , Yaw, Roll}

透视需要获取的属性：APlayerController->Apawn->FLocation{X, Y ,Z}

得到这些数据的内存地址以后，通过矩阵转换，将二维数据变为三维数据，然后外挂程序进行计算以后，再在游戏中进行修改，之后则可以实现自瞄，或者透视。同时 UE 开发的 FPS 游戏一般都可以套用子弹追踪模板，因此希望项目方后期注意这方面的检测。

以使用起源引擎的 Apex 为例，其逻辑类似，如下：



```
.  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_RENDERER, 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;
```

## 分析结论:

1. 对于一款游戏来说, 其本地逻辑的安全性与 GS 判定和本地的安全手段息息相关, 从当前的游戏表现来看其 GS 对同步的数据缺乏管控, 以及同步的数据并不完善, 同时用户数据上报缺失, 因此其在该处的安全评分为 0。
2. 同时由于 UE 试用的射线计算逻辑相似, 因此对于恶意用户可以很容易的实现如子弹追踪这类秒杀功能。

## WEB3 安全分析:

当前代币合约功能单一，且代币总供应量有限制目前已全部 Mint，其合约安全风险偏小。



Damocles labs 是成立于 2023 年的安全团队,专注于 Web3 行业的安全,业务内容包括:

合约代码审计, 业务代码审计, 渗透测试, GameFi 代码审计, GameFi 漏洞挖掘, GameFi



外挂分析, GameFi 反作弊。

我们会在 Web3 安全行业持续发力, 并且尽可能多的输出分析报告, 提升项目方和用户对于 GameFi 安全的感知度, 以及促进行业的安全发展。