Damocles

MetalCore 游戏分析报告

2024.07.05

Senna

DAMOCLES LABS

Damocles

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

Damocles

一、 概要

作为一款 STG 品类的游戏 MetalCore 在其客户端的安全性为 0，客户端的保护缺失导致攻击者如果想进行恶意分析的话会很简单，并且对游戏内的一些逻辑的分析利用也会造成产出失衡，不过 MetalCore 在 DS 校验上相对于同期的 STG 游戏校验比较严谨，因此 Damocles 判定其安全评分为 2 星。

安全性评分： 

二、 游戏背景

- 进行评估的游戏版本：1.0.0.38727
- 游戏类型&游戏引擎：STG，UE4.26
- 游戏玩法可能存在的问题：
  - 加速移动（修改本地时间速率）
  - 无后坐力
  - 无限子弹
  - 物品透视
  - 本地人物属性的修改例如出刀加速
  - 人物透视
  - 子弹追踪
  - 自瞄

三、 游戏安全性分析

游戏代码保护：

分析过程：

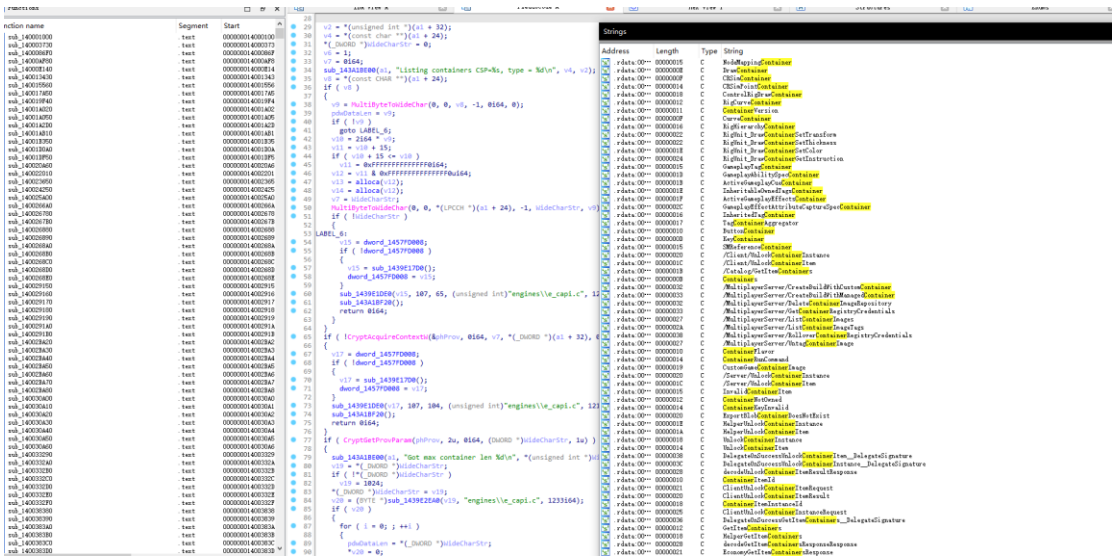
1. 由于不同的引擎有不同的分析模式，所以在获取到游戏 EXE 后首先需要确定游戏使用的引擎，通过对游戏基础信息识别我们可以确定该游戏是使用 UE4 进行开发。

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

```
static_assert(sizeof(IItemsInterface) == 0x000028, "Wrong size on IItemsInterface");

// Class MetalCore.ShooterCharacter
// 0x1940 (0x1E00 - 0x04C0)
#pragma pack(push, 0x1)
class alignas(0x10) AShooterCharacter : public ACharacter
{
public:
    uint8                                         Pad_211C[0x58];                                    // 0x04B8(0x
    TArray<struct FFireGroupWeapons>              FireGroups;                                        // 0x0510(0x
    FMulticastInlineDelegateProperty_             OnAfterPerformMovement;                            // 0x0520(0x
    uint8                                         Pad_211D[0x8];                                     // 0x0530(0x
    class UAudioComponent*                        DialogueAudioComp;                                 // 0x0538(0x
    FMulticastInlineDelegateProperty_             OnFireActionDown;                                  // 0x0540(0x
    FMulticastInlineDelegateProperty_             OnFireActionUp;                                    // 0x0550(0x
    FMulticastInlineDelegateProperty_             OnTargetActionDown;                                // 0x0560(0x
    FMulticastInlineDelegateProperty_             OnTargetActionUp;                                  // 0x0570(0x
    struct FRotator                               PlayerDeltaRot;                                    // 0x0580(0x
    uint8                                         Pad_211E[0x4];                                     // 0x058C(0x
    TArray<class UActorComponent*>                VFXComponents;                                     // 0x0590(0x
    FMulticastInlineDelegateProperty_             BPNotifyEquipWeapon;                               // 0x05A0(0x
    FMulticastInlineDelegateProperty_             BPNotifyUnEquipWeapon;                             // 0x05B0(0x
    FMulticastInlineDelegateProperty_             NotifyHit;                                         // 0x05C0(0x
    FMulticastInlineDelegateProperty_             OnShooterCharacterFired;                           // 0x05D0(0x
    FMulticastInlineDelegateProperty_             OnShooterCharacterKnockdownChanged;                // 0x05E0(0x
    FMulticastInlineDelegateProperty_             OnShooterCharacterDied;                            // 0x05F0(0x
    FMulticastInlineDelegateProperty_             OnSpawnableCooldown;                               // 0x0600(0x
    FMulticastInlineDelegateProperty_             OnFirstPersonSet;                                  // 0x0610(0x
    FMulticastInlineDelegateProperty_             OnInitalizeUIValues;                               // 0x0620(0x
    FMulticastInlineDelegateProperty_             OnWeaponInventoryChanged;                          // 0x0630(0x
    uint8                                         Pad_211F[0x10];                                    // 0x0640(0x
    FMulticastInlineDelegateProperty_             OnShooterCharacterReadyForPlay;                    // 0x0650(0x
    FMulticastInlineDelegateProperty_             OnPossessionChanged;                               // 0x0660(0x
    bool                                          bIsPossessed;                                      // 0x0670(0x
    uint8                                         bStartInFirstPerson : 1;                           // 0x0671(0x
    uint8                                         Pad_2120[0x6];                                     // 0x0672(0x
    class UCameraComponent*                       FPS_Camera;                                        // 0x0678(0x
    class UCameraComponent*                       TPS_Camera;                                        // 0x0680(0x
    class USceneComponent*                        LOSAnchorSceneComp;                                // 0x0688(0x
    bool                                          bCosmeticsInitialized;                             // 0x0690(0x
    uint8                                         Pad_2121[0x3];                                     // 0x0691(0x
    struct FVector                                CameraTPSTargetOffset;                             // 0x0694(0x
    struct FVector                                CameraTPSSocketOffset;                             // 0x06A0(0x
    struct FVector                                CameraFPSTargetOffset;                             // 0x06AC(0x
    struct FVector                                CameraFPSSocketOffset;                             // 0x06B8(0x
    uint8                                         Pad_2122[0x4];                                     // 0x06C4(0x
    class UMCConsumableComponent*                 ConsumableComp;                                    // 0x06C8(0x
    TArray<struct FDefaultWeapon>                 DefaultInventoryClasses;                           // 0x06D0(0x
    TArray<struct FWeaponPoints>                  WeaponAttachPoints;                                // 0x06E0(0x
    class UInventoryComponent*                    InventoryComp;                                     // 0x06F0(0x
```

并且使用 IDA 进行静态代码分析时可以看到本地字符串没有加密，代码未加密



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。

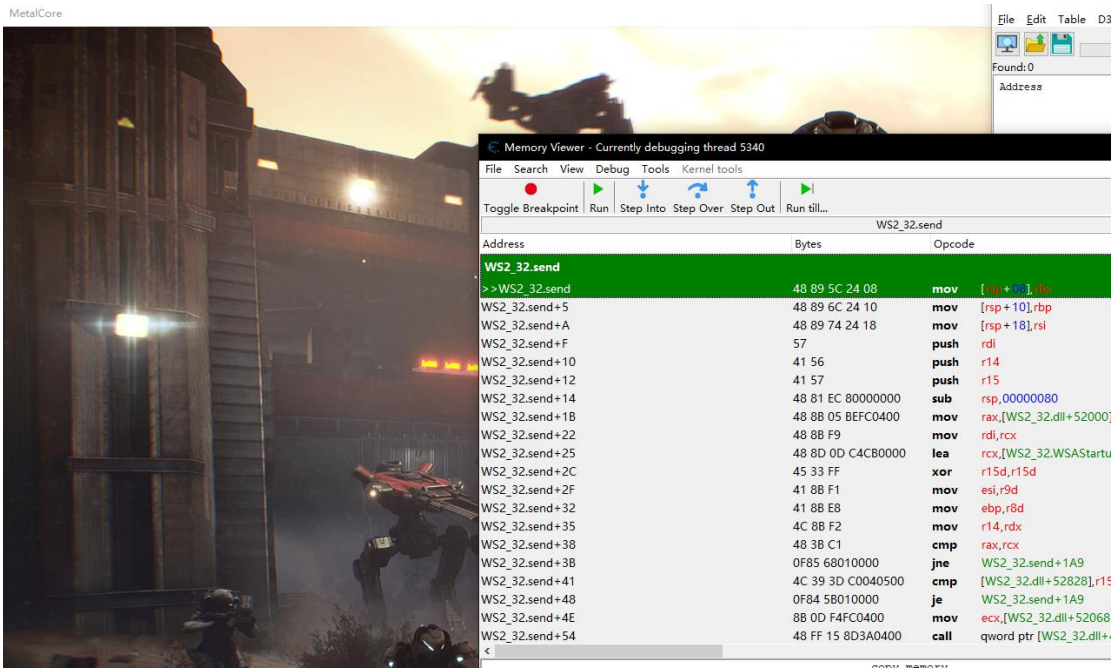
分析结论：

MetalCore 在游戏代码保护方面得分为 0 分，其 client 代码没有任何保护并且字符串也未加密，因为可以结合数据结构、字符串对游戏进行快速的分析。

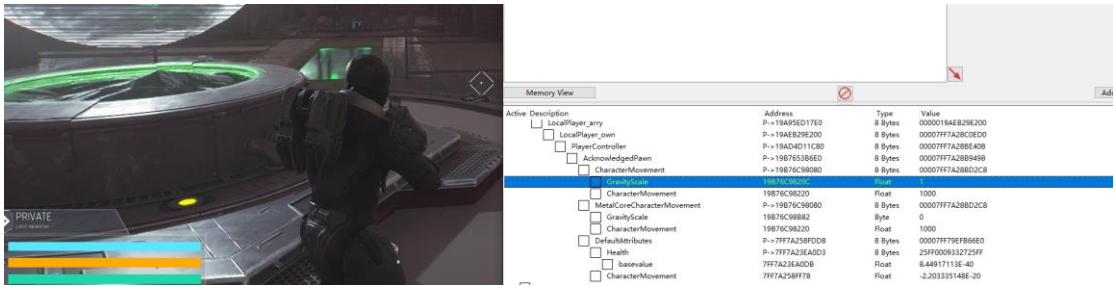
游戏基础反作弊：

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、跳跃高度等属性，并且 DS 不会进行踢出操作,但是对人物的加速修改会进行主动拉回操作。



4.针对武器的修改反而会生效，即本地可以实现无限子弹、无后座力等功能。

分析结论：

1. MetalCore 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。
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 MetalCore 分析时我们发现，当前的玩家的主要盈利模式为：

1. 完成任务获取空投代币。

2. 搜索地图内的盒子，搜集材料进行 NFT 铸造。

因为外挂作者一般都会从利润最高的这两部分出发，其最终的形态则为：

1. 针对 PVPVE 的自瞄、透视、子弹追踪、武器属性修改。

2. 针对快速盈利的高价值物品定位。

同时我们发现，在游戏数据结构中，暴露了针对物品稀有度的枚举类型，即：

Damocles

```
enum class ERarity : uint8
{
    Common                                  = 0,
    Uncommon                                = 1,
    Rare                                    = 2,
    Epic                                    = 3,
    Legendary                               = 4,
    Count                                   = 5,
    ERarity_MAX                             = 6,
};
```

分析结论：

因此，两种影响游戏/经济平衡的手段均可以快速实现，考虑到当前项目方缺少主动发现的安全能力，因此我们判定当前游戏的主要逻辑面临的外挂风险极高，基于此，其安全性评分为 0。

Damocles

游戏协议&Server 安全性分析

当前游戏协议偏少，并不完善，我们只对登录逻辑进行了简略分析。分析时发现项目方在进行设计时，将用户名密码进行明文传输，改行为并不可取，希望项目方可以进行调整。

Damocles

WEB3 安全分析：

MetalCore 代币为发行在 EVM 上的 ERC20 Token.

Symbol	MCG
TotalSupply	**3,000,000,000**
Address	0xc9E503562d0Db0A2629288a5D3b1c94Ea7741869

标准的 ERC20 token 合约，调用 Immutable 封装接口实现，总供应量固定。

File 8 of 8 : MetalCore.sol

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.19;

import "@imtbl/contracts/contracts/token/erc20/preset/ImmutableERC20FixedSupplyNoBurn.sol";

contract MetalCore is ImmutableERC20FixedSupplyNoBurn {
    constructor(
        address _treasurer,
        address _owner
    ) ImmutableERC20FixedSupplyNoBurn("MetalCore", "MCG", 3_000_000_000 ether, _treasurer, _owner) {}
}
```

关于 Damocles

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：GameFi 安全顾问、合约代码审计，业务代码审计，渗透测试，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 GameFi 安全的感知度，以及促进行业的安全发展。