Damocles

LastRemains 游戏分析报告

2024.06.19

Senna

DAMOCLES LABS

Damocles

目录

Damocles

一、 概要

Last Remains 作为一款僵尸大逃杀的游戏，其在保证合约安全与游戏性的前提下应该同样重视游戏本身的安全与公平性，但是目前来看 Last Remains 存在 PDB 泄露的问题，该问题后续会导致很多的风险并且当前代买来看，本地疑似存在爆率操纵因子可以主动修改，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：05_29_2024
- 游戏类型&游戏引擎：STG&FPS，UE5.3.2
- 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
  - 无限体力
  - 僵尸秒杀
  - 子弹追踪

三、 游戏安全分析 | PDB 泄露带来的风险

什么是 PDB:

PDB 文件是指 “Program Database” 文件。这种文件由微软的编译器和调试器生成和使用，主要目的是在程序调试期间提供符号信息。PDB 文件通常具有 .pdb 扩展名。

PDB 文件的作用:

符号信息存储:

PDB 文件包含了程序中的调试符号信息，包括变量名、函数名、类名、源文件名、行号等。这些信息对于调试器来说是至关重要的，因为它们可以将二进制代码映射回源代码。

调试支持:

在调试过程中，调试器使用 PDB 文件来提供详细的调试信息，如变量的值、调用堆栈、断点的设置等。通过这些信息，开发者可以更容易地定位和修复程序中的错误。

性能分析:

PDB 文件也可以用于性能分析工具，以提供符号解析功能，帮助开发者识别性能瓶颈，并进行代码优化。

崩溃分析:

当程序崩溃时，生成的崩溃转储文件（minidump）可以结合 PDB 文件来分析崩溃原因。PDB 文件提供了符号信息，使得开发者可以查看崩溃发生时的调用堆栈、变量状态等详细信息。

PDB 文件的生成

PDB 文件通常在程序编译时生成。开发者可以通过在编译器选项中启用调试信息生成来创建 PDB 文件。例如，在 Microsoft Visual Studio 中，开发者可以在项目属性中启用生成调试信息的选项。

PDB 文件的内容

PDB 文件包含以下类型的信息：

- **全局变量和局部变量的名称和类型**。
- **函数的名称、参数和返回类型**。
- **类和结构的定义**。
- **源文件和行号信息**。
- **调试符号，如断点和数据断点。**

PDB 泄露带来的危害：

官方解释：（来自于 GPT4）

如果一个游戏的 PDB 文件泄露了，可能会带来一系列的安全和隐私风险。这些风险主要包括以下几个方面：

1. **逆向工程和代码分析**

PDB 文件包含了丰富的调试符号信息，如变量名、函数名、类名、以及源代码的行号等。这些信息可以大大简化逆向工程的过程，使得攻击者能够更容易地理解游戏的内部逻辑和结构。

示例：攻击者可以使用 PDB 文件将编译后的二进制代码映射回源代码，从而分析游戏的逻辑、算法和数据结构。这有助于他们找到漏洞或设计破解手段，如作弊工具或未授权的修改。

2. 漏洞发现

通过分析 PDB 文件，攻击者可以更容易地发现游戏中的安全漏洞和缺陷，例如缓冲区溢出、未处理的异常或其他潜在的安全问题。

示例：如果 PDB 文件暴露了某个函数处理用户输入的具体实现，攻击者可能会发现并利用其中的输入验证漏洞，进行缓冲区溢出攻击，从而执行任意代码。

3. 作弊和黑客攻击

PDB 文件可以帮助攻击者制作更有效的作弊工具。通过了解游戏的内部机制，他们可以设计出更隐蔽和高效的作弊方法，影响游戏的公平性和玩家体验。

示例：在多人在线游戏中，攻击者可以利用 PDB 文件编写自动瞄准（aimbot）、透视（wallhack）等作弊工具，严重影响游戏的公平性和玩家体验。

4. 知识产权和商业秘密泄露

游戏开发中的独特算法、创新设计和其他知识产权信息可能会被暴露。如果竞争对手获得这些信息，他们可能会复制或模仿这些创新，从而损害原开发公司的竞争优势。

示例：如果一个游戏引擎的独特优化算法被泄露，竞争对手可以分析并实现类似的优化，甚至可能改进并超越原作，从而抢占市场份额。

5. 玩家数据和隐私风险

虽然 PDB 文件主要包含调试信息，但如果游戏中处理了敏感的玩家数据（如登录信息、支付信息等），这些数据的处理逻辑可能也会被暴露，增加数据泄露的风险。

示例：如果 PDB 文件暴露了处理玩家支付信息的代码逻辑，攻击者可以分析并找到绕过支付验证的方法，进行欺诈和盗窃。

PDB 泄露问题在 Last Remains 上的示例：

以可实现的外挂功能为例

示例一：子弹追踪

在 UE 中开发射击类游戏不可避免的会使用一种名为射线检测的机制来判断玩家开枪时是否击中目标。射线检测（Raycast）是一种应用广泛的检测方式，类似碰撞检测，也能用于判断是否指向了某个物件。以游戏 Apex 为例：恶灵被瞄准时会得到提示；视线对准的物件会被高亮；鼠标指向的物体能被选中，等等。

在 Cocos 引擎中，射线检测被定义为：对一条射线和另一个形状进行**相交性判断**。而虚幻引擎则认为：**射线检测是获取有关线段上存在内容的反馈的方法。**

综合来讲，射线检测会发射一条由 A -> B 的射线，如果射线命中一个游戏对象，便可以获取到对象的引用、角度、位置等信息。游戏开发商可以利用这些信息来进行逻辑判断，实现 Gameplay 功能。

正常游戏中定位射线检测函数需要的工作量庞大，由于函数名、数据结构完全隐藏，黑客想要定位需要深入的理解引擎原理。但是如果游戏存在 PDB，由于 PDB 中存在所有的游戏结构体、变量名信息，所以定位迅速。以下便是 Last Remains 与射线检测相关的函数。

如果黑客想要进行恶意操作，例如实现原地开枪同时子弹可以射击到其他玩家的功能，则可以对函数内的 StartTrace 数据进行本地计算后修改。

Damocles

```
if ( BulletsPerCartridge > 0 )
{
  do
  {
    CalculatedSpreadAngleMultiplier = ULyraRangedWeaponInstance::GetCalculatedSpreadAngleMultiplier(WeaponData);
    VRandConeNormalDistribution(
      (UE::Math::TVector<double> *)&BulletDir.XYZ[1],
      &v6->AimDir,
      (float)(CalculatedSpreadAngleMultiplier * WeaponData->CurrentSpreadAngle) * 0.0087266462,
      WeaponData->SpreadExponent);
    v11 = _mm_cvtps_pd((__m128)LODWORD(WeaponData->MaxDamageRange));
    *(_QWORD *)&AllImpacts.ArrayNum = 0i64;
    v41 = 0i64;
    v12.m128d_f64[1] = v11.m128d_f64[1];
    v13.m128d_f64[1] = v11.m128d_f64[1];
    v12.m128d_f64[0] = v11.m128d_f64[0] * BulletDir.Y + v6->StartTrace.X;
    v13.m128d_f64[0] = v11.m128d_f64[0] * BulletDir.Z + v6->StartTrace.Y;
    v14 = _mm_unpacklo_pd(v12, v13);
    SweepRadius = WeaponData->BulletTraceSweepRadius;
    v15 = v11.m128d_f64[0] * *(double *)&Impact.FaceIndex + v6->StartTrace.Z;
    BulletDir.X = v15;
    *(__m128d *)&EndTrace.XYZ[1] = v14;
    ULyraGameplayAbility_RangedWeapon::DoSingleBulletTrace(
      v7,
      (FHitResult *)&Impact.Distance,
      &v6->StartTrace,
      (UE::Math::TVector<double> *)&EndTrace.XYZ[1],
      SweepRadius,
      0,
      (TArray<FHitResult,TSizedDefaultAllocator<32> > *)&AllImpacts.ArrayNum);
    *(_QWORD *)&EndTrace.X = FActorInstanceHandle::FetchActor((FActorInstanceHandle *)&Impact.HitObjectHandle.Manager);
    X = EndTrace.X;
```

Damocles

```
FHitResult *__fastcall ULyraGameplayAbility_RangedWeapon::DoSingleBulletTrace(
        ULyraGameplayAbility_RangedWeapon *this,
        FHitResult *result,
        const UE::Math::TVector<double> *StartTrace,
        const UE::Math::TVector<double> *EndTrace,
        float SweepRadius,
        bool bIsSimulated,
        TArray<FHitResult,TSizedDefaultAllocator<32> > *OutHits)
{
  FHitResult *v11; // rax
  __int128 v12; // xmm1
  __int64 v13; // r15
  FHitResult *v14; // rax
  __int128 v15; // xmm1
  int FirstPawnHitResult; // eax
  int ArrayNum; // esi
  __int64 v18; // r13
  __int64 v19; // rbp
  FScriptContainerElement *v20; // rbx
  FScriptContainerElement *v21; // rdi
  FScriptContainerElement *v22; // rsi
  int ArrayMax; // r8d
  FScriptContainerElement *Data; // rdi
  FScriptContainerElement *v25; // rbx
  FScriptContainerElement *v26; // rdi
  FScriptContainerElement v27; // cl
  __int64 v28; // rax
  TArray<FHitResult,TSizedDefaultAllocator<32> > SweepHits; // [rsp+40h] [rbp-138h] BYREF
  FHitResult resulta; // [rsp+50h] [rbp-128h] BYREF

  FHitResult::FHitResult(result, (EForceInit)result);
  if ( UGameplayAbility_RangedWeapon::FindFirstPawnHitResult(OutHits) == -1 )
  {
    v11 = ULyraGameplayAbility_RangedWeapon::WeaponTrace(
            this,
            &resulta,
            StartTrace,
            EndTrace,
            0.0,
            bIsSimulated,
            OutHits);
    *(_OWORD *)&result->FaceIndex = *(_OWORD *)&v11->FaceIndex;
    *(_OWORD *)&result->Location.X = *(_OWORD *)&v11->Location.X;
    *(_OWORD *)&result->Location.XYZ[2] = *(_OWORD *)&v11->Location.XYZ[2];
    *(_OWORD *)&result->ImpactPoint.XYZ[1] = *(_OWORD *)&v11->ImpactPoint.XYZ[1];
    *(_OWORD *)&result->Normal.X = *(_OWORD *)&v11->Normal.X;
    *(_OWORD *)&result->Normal.XYZ[2] = *(_OWORD *)&v11->Normal.XYZ[2];
    *(_OWORD *)&result->ImpactNormal.XYZ[1] = *(_OWORD *)&v11->ImpactNormal.XYZ[1];
    v12 = *(_OWORD *)&v11->TraceStart.X;
    v11 = (FHitResult *)((char *)v11 + 128);
    *(_OWORD *)&result->TraceStart.X = v12;
    *(_OWORD *)&result->TraceStart.XYZ[2] = *(_OWORD *)&v11->FaceIndex;
    *(_OWORD *)&result->TraceEnd.XYZ[1] = *(_OWORD *)&v11->Location.X;
    *(_OWORD *)&result->PenetrationDepth = *(_OWORD *)&v11->Location.XYZ[2];
    *(_OWORD *)&result->PhysMaterial.ObjectIndex = *(_OWORD *)&v11->ImpactPoint.XYZ[1];
    *(_OWORD *)&result->HitObjectHandle.Manager.ObjectIndex = *(_OWORD *)&v11->Normal.X;
    *(_OWORD *)&result->Component.ObjectIndex = *(_OWORD *)&v11->Normal.XYZ[2];
    result->MyBoneName = *(FName *)&v11->ImpactNormal.Y;
  }
  if ( UGameplayAbility_RangedWeapon::FindFirstPawnHitResult(OutHits) == -1 && SweepRadius > 0.0 )
  {
    v13 = 0i64;
    SweepHits.AllocatorInstance.Data = 0i64;
    *(_QWORD *)&SweepHits.ArrayNum = 0i64;
    v14 = ULyraGameplayAbility_RangedWeapon::WeaponTrace(
            this,
            &resulta,
            StartTrace,
            EndTrace,
            SweepRadius,
            bIsSimulated,
            &SweepHits);
    *(_OWORD *)&result->FaceIndex = *(_OWORD *)&v14->FaceIndex;
    *(_OWORD *)&result->Location.X = *(_OWORD *)&v14->Location.X;
    *(_OWORD *)&result->Location.XYZ[2] = *(_OWORD *)&v14->Location.XYZ[2];
    *(_OWORD *)&result->ImpactPoint.XYZ[1] = *(_OWORD *)&v14->ImpactPoint.XYZ[1];
    *(_OWORD *)&result->Normal.X = *(_OWORD *)&v14->Normal.X;
    *(_OWORD *)&result->Normal.XYZ[2] = *(_OWORD *)&v14->Normal.XYZ[2];
    *(_OWORD *)&result->ImpactNormal.XYZ[1] = *(_OWORD *)&v14->ImpactNormal.XYZ[1];
    v15 = *(_OWORD *)&v14->TraceStart.X;
    v14 = (FHitResult *)((char *)v14 + 128);
    *(_OWORD *)&result->TraceStart.X = v15;
    *(_OWORD *)&result->TraceStart.XYZ[2] = *(_OWORD *)&v14->FaceIndex;
    *(_OWORD *)&result->TraceEnd.XYZ[1] = *(_OWORD *)&v14->Location.X;
    *(_OWORD *)&result->PenetrationDepth = *(_OWORD *)&v14->Location.XYZ[2];
    *(_OWORD *)&result->PhysMaterial.ObjectIndex = *(_OWORD *)&v14->ImpactPoint.XYZ[1];
    *(_OWORD *)&result->HitObjectHandle.Manager.ObjectIndex = *(_OWORD *)&v14->Normal.X;
    *(_OWORD *)&result->Component.ObjectIndex = *(_OWORD *)&v14->Normal.XYZ[2];
    result->MyBoneName = *(FName *)&v14->ImpactNormal.Y;
    FirstPawnHitResult = UGameplayAbility_RangedWeapon::FindFirstPawnHitResult(&SweepHits);
    if ( FirstPawnHitResult >= 0 )
    {
      ArrayNum = SweepHits.ArrayNum;
      if ( FirstPawnHitResult < SweepHits.ArrayNum )
      {
        if ( FirstPawnHitResult <= 0 )
        {
LABEL_18:
          if ( OutHits != &SweepHits )
          {
            ArrayMax = OutHits->ArrayMax;
            Data = SweepHits.AllocatorInstance.Data;
```

示例二：游戏逻辑接管

大部分 GameFi 的玩家均是为了获利才会选择花费精力与时间来参与一款游戏，所以在游戏中如何获利、如何快速获利备受玩家/工作室青睐。正常游戏来说，如果想要获取到代币/NFT 的刷新点，其分析难度很大，投入与产出不成正比，但是在 PDB 泄露的游戏上快速获利却变得异常简单。

在 Last Remains 中地图存在 NFT 刷新点，正常来说 NFT 刷新是随机的，但是可以通过 hook 的方式来获取当前游戏所有的物资，再结合图形绘制进行标记。

通过标记的方式可以轻松找到所有的物资点，然后进行针对性提取。

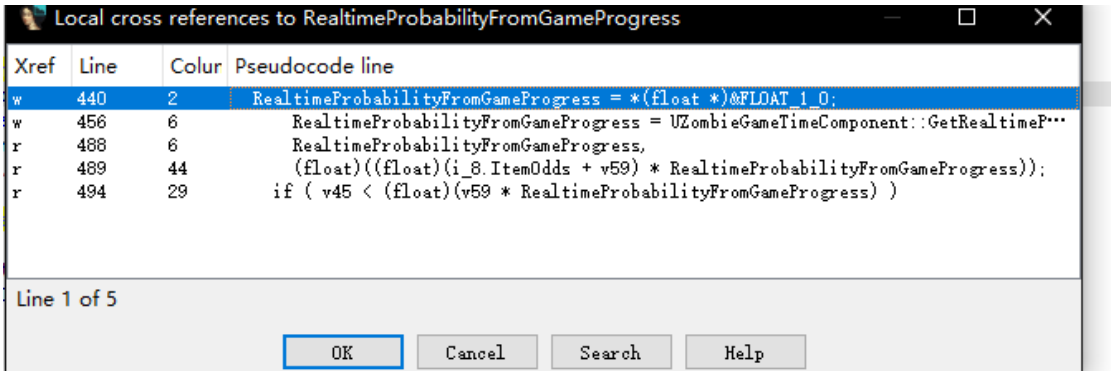
以下为 Last Remains 创建物资的函数：

```
void __fastcall UContainerComponent::BeginPlay(UContainerComponent *this, __int64 a2, float a3)
{
  AActor *OwnerPrivate; // rbx
  UClass *PrivateStaticClass; // rax
  UClass *ClassPrivate; // rdx
  FStructBaseChain *v7; // r8
  __int64 NumStructBasesInChainMinusOne; // rax
  bool v9; // zf
  AActor *v10; // rdi
  UActorComponent *(__fastcall *FindComponentByClass)(AActor *, const TSubclassOf<UActorComponent>); // rbx
  UClass *v12; // rax
  __int64 v13; // rax

  UActorComponent::BeginPlay(this);
  this->SetComponentTickEnabled(this, 0);
  if ( (*((_BYTE *)&this->UActorComponent + 136) & 0x20) != 0 && !this->bCreated )
  {
    this->bCreated = 1;
    if ( UKismetSystemLibrary::IsDedicatedServer(this) )
    {
      UContainerComponent::CreateItem(this, 0, a3);
      if ( this->bInEnmtyDes )
      {
        OwnerPrivate = this->OwnerPrivate;
        if ( OwnerPrivate )
        {
          PrivateStaticClass = AContainer::GetPrivateStaticClass();
          ClassPrivate = OwnerPrivate->ClassPrivate;
          v7 = &PrivateStaticClass->FStructBaseChain;
          NumStructBasesInChainMinusOne = PrivateStaticClass->NumStructBasesInChainMinusOne;
          if ( (int)NumStructBasesInChainMinusOne <= ClassPrivate->NumStructBasesInChainMinusOne
            && ClassPrivate->StructBaseChainArray[NumStructBasesInChainMinusOne] == v7 )
          {
            if ( LOBYTE(OwnerPrivate[1].ReplicatedComponents.ArrayNum) )
            {
              if ( LODWORD(OwnerPrivate[1].ReplicatedComponentsInfo.AllocatorInstance.Data) )
                return;
              v9 = LODWORD(OwnerPrivate[1].ReplicatedMovement.LinearVelocity.XYZ[2]) == 0;
              goto LABEL_14;
            }
            v10 = this->OwnerPrivate;
            FindComponentByClass = v10->FindComponentByClass;
            v12 = UInventoryComponent::StaticClass();
            v13 = ((__int64 (__fastcall *)(AActor *, UClass *))FindComponentByClass)(v10, v12);
            if ( v13 && !*(_DWORD *)(v13 + 800) )
            {
              v9 = *(_DWORD *)(v13 + 512) == 0;
LABEL_14:
              if ( v9 )
                AActor::Destroy(this->OwnerPrivate, 0, 1);
            }
          }
        }
      }
    }
  }
}
```

同时在对游戏源码的分析过程中我们发现，该游戏的刷新概率疑似是通过本地控制的，由于游戏已经关服无法进行验证，但是该问题如果存在那对游戏/玩家/投资者均是巨大损失。

```
    v63 = FLOAT_2_3;
    goto LABEL_115;
  }
  v65 = &::Data;
  if ( i_8.ItemID.Data.ArrayNum )
    v65 = (const wchar_t *)i_8.ItemID.Data.AllocatorInstance.Data;
  if ( !FGenericPlatformStricmp::Stricmp(v65, "weapon_8001") )
  {
    a3 = FLOAT_4_0;
    v63 = FLOAT_2_5999999;
    goto LABEL_115;
  }
LABEL_116:
  v66 = &::Data;
  if ( i_8.ItemID.Data.ArrayNum )
    v66 = (const wchar_t *)i_8.ItemID.Data.AllocatorInstance.Data;
  FString::PrintfImpl(
    &result,
    L"当前生成%s的修正参数为%f,实际生成概率为%f",
    v66,
    RealtimeProbabilityFromGameProgress,
    (float)((float)(i_8.ItemOdds + v59) * RealtimeProbabilityFromGameProgress));
  UScriptStruct::TCppStructOps<FLyraHUDElementEntry>::GetPreloadDependencies((CAkMidiClipCtx *)&result, v67, a3);
  if ( result.Data.AllocatorInstance.Data )
    FMemory::Free(result.Data.AllocatorInstance.Data);
  v59 = i_8.ItemOdds + v59;
  if ( v45 < (float)(v59 * RealtimeProbabilityFromGameProgress) )
    break;
  if ( i_8.ItemID.Data.AllocatorInstance.Data )
    FMemory::Free(i_8.ItemID.Data.AllocatorInstance.Data);
  v58 += 24;
  if ( v58 == v61 )
    goto LABEL_160;
}
v68 = i_8.ItemID.Data.ArrayNum;
v69 = i_8.ItemID.Data.AllocatorInstance.Data;
result.Data.AllocatorInstance.Data = 0i64;
```



该属性 RealtimeProbabilityFromGameProgress 被强制设置为 1.0

修复建议：

1. 风险评估：分析泄露信息可能带来的具体风险，包括逆向工程、漏洞利用、作弊工具开发等。
2. 安全加固：
   a) 修复已知漏洞：如果泄露的 PDB 文件暴露了代码中的漏洞，迅速修补这些漏

洞，并发布安全更新。

b) 代码审计：进行全面的代码审计，找出可能被泄露的其他安全隐患。

3. 异常行为检测

a) 玩家行为监控：监控游戏中的异常玩家行为，尤其是可能涉及作弊或漏洞利用的行为。

b) 自动化工具：使用自动化工具检测和防止潜在的漏洞利用和作弊行为。

4. 长期改进措施：

安全培训

a) 员工培训：定期开展安全培训，提高员工的安全意识和技能，防止类似事件再次发生。

b) 模拟演练：进行模拟演练，确保员工熟悉应急响应流程。

安全审计

c) 定期审计：定期进行内部和外部安全审计，识别和修复安全漏洞。

d) 第三方评估：邀请第三方安全专家进行独立评估，提供客观的安全建议。

安全开发生命周期（SDL）

e) 安全编码实践：在开发过程中实施安全编码实践，减少代码中的安全漏洞。

f) 持续集成和部署：在持续集成和部署（CI/CD）管道中集成安全测试，确保每次代码更新都经过安全审查。

Damocles

WEB3 安全分析：

目前 Last Remains 的资产合约只有两份 NFT 合约，分 Last Remains Equipment 和 Last Remains Characters 二者均部署在 Polygon 链上。

目前来看 Equipment 合约使用的协议为 ERC721 未设置最大供应量，且 mint 权限为项目方官方地址。

```
    }

    function mintMultiple(address _to, uint256 _quantity, string calldata _baseTokenURI) external onlyRole(MINTER_ROLE) {
        require(_quantity <= 1000, "!q");
        require(_quantity > 0, "!q");
        require(_to != address(0), "!a");
        if (bytes(_baseTokenURI).length != 0) {
            _addTokenBatch(_quantity, _baseTokenURI);
        }
        // enforcing a batch of 10 for ERC721A transfer efficiency
        // https://medium.com/@dumbnamenumbers/your-guide-to-erc721a-july-2022-f81f0be84a54
        for (uint256 i = 0; i < _quantity; i += 10) {
            uint256 batch = _quantity - i >= 10 ? 10 : _quantity - i;
            _safeMint(_to, batch);
        }

    }

    function _addTokenBatch(uint256 _quantity, string calldata _baseTokenURI) internal {
        uint256 _next = _nextTokenId();
        batchBaseTokenURI.push(_baseTokenURI);
        uint256[2] memory range;
        range[0] = _next;
        range[1] = _next + _quantity - 1;
        batchRange.push(range);
        nextBatchId++;
    }

    // pricePerToken and currency are ignored for the moment
    function mintWithSignature(RequestWithParam calldata _req, bytes calldata _sig) external {
        require(hasRole(MINTER_ROLE, _recoverAddress(_req, _sig)), "!M");
        _processRequest(_req);
        DefinitiveMintRequests memory p = _decodeMintParam(_req.param);
        require(p.to != address(0), "!r");
        require(p.m.length > 0, "!m.length");
        uint256 startTokenId = _nextTokenId();
        uint256[] memory tokenIds = new uint256[](p.m.length);

        for (uint256 i = 0; i < p.m.length; i++) {
            tokenURIOverride[startTokenId + i] = p.m[i].uri;
            tokenIds[i] = startTokenId + i;
        }
        // reentrancy checked in ERC721A._safeMint
        _safeMint(p.to, p.m.length);
        emit MintWithSignature(p.transactionIdHash, p.userIdHash, p, tokenIds);
    }

    function burnWithSignature(RequestWithParam calldata _req, bytes calldata _sig) external {
        require(hasRole(BURNER_ROLE, _recoverAddress(_req, _sig)), "!M");
        _processRequest(_req);
        BurnRequests memory p = _decodeBurnParam(_req.param);
        uint256 len = p.tokenIds.length;
        require(len > 0, "!len");
        string[] memory uris = new string[](len);

        for (uint256 i = 0; i < len; i++) {
            uris[i] = tokenURI(p.tokenIds[i]);
            // ownership is checked in ERC721A._burn
            _burn(p.tokenIds[i], true);
```

Character 合约使用的协议为 ERC1155 已设置最大供应量，权限已交给黑洞地址不存在增发问题。

Damocles

```
    /// @param _supply What to set the new supply as
    function setMaxSupply(uint256 _id, uint256 _supply) public onlyOwner {
        maxSupply[_id] = _supply;
    }

    /// @notice Owner can set Max Supply for more than one character tokenId
    /// @dev owner only
    /// @param supplies array of struct SupplyInput
    function setMaxSupplies(SupplyInput[] calldata supplies) public onlyOwner {
        for (uint i = 0; i < supplies.length; i++) {
            maxSupply[supplies[i].id] = supplies[i].supply;
        }
    }

    /// @notice setGenesis sets if genesis mints allowed
    /// @dev MUST be set upon each upgrade!
    //// @param mintStatus bool, true means a genesis character can be minted
    function setGenesis(bool mintStatus) public onlyOwner {
        genesisEnabled = mintStatus;
    }

    /// @notice amountMintable utility that checks if amount of tokens is allowable
    /// @param maxCollSupply max collection supply for total 1155
    /// @param collSupply collection supply presently
    /// @param amt amount to mint
    function amountMintable(
        uint256 maxCollSupply,
        uint256 collSupply,
        uint256 amt
    ) internal pure returns (uint256) {
        uint256 supplyLeft = maxCollSupply - collSupply;
        return supplyLeft > amt ? amt : supplyLeft;
    }
```

由于游戏不断产出 Equipment NFT 所以机制导致装备会不断增发。

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：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，GameFi 安全顾问。

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。