Last Remains Security Analysis

2024.06.19

Senna

DAMOCLES LABS

Contents

一、 Summary

As a zombie battle royale game, Last Remains should prioritize the security and fairness of the game itself while ensuring the safety of contracts. However, it has come to our attention that Last Remains currently has a PDB leakage issue. This problem can lead to various risks in the future. Additionally, based on our preliminary investigation, there seems to be a suspected manipulation factor for drop rates that can be actively modified locally. Therefore, Damocles has determined its security rating to be 1 star.

Security Rating: ★ ☆ ☆ ☆ ☆

二、 Game Background

- Game Version：05_29_2024
- Genres & Engine：FPS，UE5.3.2
- Possible Issues in Gameplay：
  - Unauthorized movement (modifying local character attributes for speed enhancement).
  - No recoil
  - Aimbot
  - Bullet tracking
  - Settlement replay attack
  - Modification of local character attributes, such as jumping
  - Instant zombie kills

三、 Game Security Analysis

What is PDB:

PDB files, short for "Program Database" files, are generated and used by Microsoft compilers and debuggers. Their main purpose is to provide symbol information during program debugging. PDB files typically have a .pdb extension.

The role of PDB files includes:

Symbol Information Storage:

PDB files contain debugging symbol information from the program, including variable names, function names, class names, source file names, line numbers, and more. This information is crucial for the debugger as it allows mapping of the binary code back to the source code.

Debugging Support:

During the debugging process, the debugger uses PDB files to provide detailed debugging information such as variable values, call stacks, breakpoint settings, and more. With this information, developers can more easily locate and fix errors in their program.

Performance Analysis:

PDB files can also be used by performance analysis tools to provide symbol resolution capabilities, helping developers identify performance bottlenecks and optimize their code.

Crash Analysis:

When a program crashes, the generated crash dump file (minidump) can be analyzed in conjunction with the PDB file to determine the cause of the crash. The PDB file provides symbol information, allowing developers to examine detailed information such as the call stack and variable states at the time of the crash.

PDB files are typically generated during program compilation. Developers can create PDB files by enabling the generation of debug information in compiler options. For example, in Microsoft Visual Studio, developers can enable the option to generate debug information in project properties.

The contents of PDB files include:

Names and types of global and local variables.

- Names, parameters, and return types of functions.
- Definitions of classes and structures.
- Source file and line number information.
- Debug symbols such as breakpoints and data breakpoints.

Harm caused by PDB leakage:

Official explanation: (from GPT4)

If a game's PDB file is leaked, it can pose a series of security and privacy risks. These risks mainly include the following aspects:

Reverse engineering and code analysis

PDB files contain rich debugging symbol information, such as variable names,

function names, class names, and source code line numbers. This information can greatly simplify the process of reverse engineering, making it easier for attackers to understand the game's internal logic and structure.

Example: Attackers can use PDB files to map compiled binary code back to source code, allowing them to analyze the game's logic, algorithms, and data structures. This helps them identify vulnerabilities or design cracking methods, such as cheating tools or unauthorized modifications.

Vulnerability discovery

By analyzing PDB files, attackers can more easily discover security vulnerabilities and flaws in the game, such as buffer overflows, unhandled exceptions, or other potential security issues.

Example: If a PDB file exposes the specific implementation of a function that handles user input, attackers may discover and exploit input validation vulnerabilities, leading to buffer overflow attacks and arbitrary code execution.

Cheating and hacker attacks

PDB files can assist attackers in creating more effective cheating tools. By understanding the game's internal mechanisms, they can design more covert and efficient cheating methods, affecting the fairness of the game and player experience.

Example: In multiplayer online games, attackers can use PDB files to develop cheating tools like aimbots or wallhacks, severely impacting the fairness of the game and player experience.

Intellectual property and trade secret leakage

Unique algorithms, innovative designs, and other intellectual property information in game development may be exposed. If competitors obtain this information, they may replicate or imitate these innovations, thereby undermining the original development company's competitive advantage.

Example: If a game engine's unique optimization algorithm is leaked, competitors can analyze and implement similar optimizations, or even improve upon and surpass the original, thereby capturing market share.

Player data and privacy risks

Although PDB files primarily contain debugging information, if the game handles sensitive player data (such as login information, payment information, etc.), the logic for processing this data may also be exposed, increasing the risk of data breaches.

Example: If a PDB file exposes the code logic for handling player payment information, attackers can analyze and find ways to bypass payment verification, leading to fraud and theft.

Examples of PDB leakage issues in Last Remains:

Using achievable cheat functionalities as an example.

Example One: Bullet Tracking

In shooting games developed with Unreal Engine (UE), it is inevitable to use a mechanism called raycasting to determine if the player hits a target when shooting.

Raycasting is a widely used detection method, similar to collision detection, and can also be used to determine if it is pointing towards an object. Taking the game Apex as an example: when aiming at an enemy, there is a prompt; objects in line of sight are highlighted; objects pointed at by the mouse can be selected, and so on.

In the Cocos engine, raycasting is defined as determining the intersection between a ray and another shape. Unreal Engine, on the other hand, sees raycasting as a method of obtaining feedback about the content present on a line segment.

In summary, raycasting involves firing a ray from point A to point B. If the ray hits a game object, it is possible to obtain references, angles, positions, and other information about that object. Game developers can use this information for logical calculations and implement gameplay features.

In a normal game, locating the raycasting function requires a significant amount of work. Due to hidden function names and data structures, hackers would need a deep understanding of the engine's principles to locate them. However, if the game has a PDB file, all game structures and variable names are available in the PDB, making the process of locating functions quick. The following are the functions in Last Remains related to raycasting.

If a hacker intends to perform malicious operations, such as implementing the ability to shoot other players while standing still, they can modify the StartTrace data within the function after performing local calculations.

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;
```

Example Two: Game Logic Takeover

Most GameFi players participate in a game with the goal of making a profit, investing their time and effort. Therefore, how to profit and how to quickly profit in the game is highly sought after by players and studios. In a normal game, analyzing and obtaining the refresh points for tokens/NFTs can be challenging, with the input not always proportional to the output. However, in a game where PDB leakage has occurred, quickly profiting becomes exceptionally simple.

In Last Remains, there are NFT refresh points on the map. Normally, the NFT refresh is random, but it is possible to obtain all the game's resources by hooking into the game and marking them using graphics rendering.

By marking the resources, it becomes easy to locate all the resource points and then extract them selectively.

The following is the function in Last Remains for creating resources.

Damocles

```
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);
            }
          }
        }
      }
    }
  }
}
```

During the analysis of the game's source code, we discovered that the refresh probability in the game appears to be controlled locally. However, since the game has already been shut down, we are unable to verify this. If this issue does indeed exist, it would result in significant losses for the game, players, and investors

The property "RealtimeProbabilityFromGameProgress" is locally enforced to be set as 1.0.

Repair Suggestions:

1. Risk assessment: Analyze the specific risks that may arise from leaked information, including reverse engineering, vulnerability exploitation, cheat tool development, etc.
2. Security reinforcement:
   a) Fix known vulnerabilities: If the leaked PDB files expose vulnerabilities in the code, promptly patch these vulnerabilities and release security updates.
   b) Code auditing: Conduct a comprehensive code audit to identify other potential security risks that may have been leaked.
3. Anomaly behavior detection

a) Player behavior monitoring: Monitor abnormal player behavior in the game, especially behavior that may involve cheating or vulnerability exploitation.

b) Automated tools: Use automated tools to detect and prevent potential vulnerability exploitation and cheating behavior.

4. Long-term improvement measures:

Security training

a) Employee training: Conduct regular security training to enhance employees' security awareness and skills, preventing similar incidents from happening again.

b) Simulation exercises: Perform simulation exercises to ensure that employees are familiar with emergency response procedures.

Security audits

c) Regular audits: Conduct regular internal and external security audits to identify and fix security vulnerabilities.

d) Third-party assessments: Invite third-party security experts to conduct independent evaluations and provide objective security recommendations.

Security Development Lifecycle (SDL)

e) Secure coding practices: Implement secure coding practices during the development process to minimize security vulnerabilities in the code.

f) Continuous integration and deployment: Integrate security testing into the continuous integration and deployment (CI/CD) pipeline to ensure that every code update undergoes security review.

Damocles

WEB3 Security Analysis:

Currently, the asset contracts in Last Remains consist of two NFT contracts, namely Last Remains Equipment and Last Remains Characters. Both contracts are deployed on the Polygon network.

Based on the current information, the Equipment contract is utilizing the ERC721 protocol without setting a maximum supply limit. Additionally, the minting permissions for the Equipment contract are held by the official address of the project team..

```
    }

    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);
```

The Character contract in Last Remains is utilizing the ERC1155 protocol and has a maximum supply limit set. The minting permissions have been transferred to a black

hole address, ensuring that there are no issues with additional supply.

```
    /// @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;
    }
```

Due to the continuous generation of Equipment NFTs in the game, the mechanism leads to a constant increase in the supply of equipment.

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