UnrealEngine POLYGON 全逆向笔记

> ⚠ **Warning**
>
> 本文仅用于交流学习请勿用于不法用途，如有侵权联系Euarno@outlook.com或Euarno@qq.com要求删除
>
> 本文由Euarno原创，转载请标明出处

准备阶段

游戏环境

- [x] 在**Steam**下载并安装**POLYGON**
- [x] 找到游戏目录**POLYGON\POLYGON\Binaries\Win64**
- [x] 找到游戏文件**POLYGON-Win64-Shipping.exe**
- [x] 拖进**IDA**等待漫长分析过程

开发环境

- [x] **Visual Studio 2022**
- [x] **C++20**

逆向环境

- [x] **Cheat Engine**
- [x] **IDA Pro**
- [x] **Inject Tool**

> 💡 **Tip**
>
> 游戏有**EasyAntiCheat**保护

开始

Directx Virtual Table

1. 直接打开**CE**添加这几个地址,确认游戏是基于Dx11

ID	Description	ShowAsSigned	VariableType	Address
0	无描述	0	4 Bytes	d3d11.dll
1	无描述	0	4 Bytes	d3d12.dll
2	无描述	0	4 Bytes	d3d9.dll

2. 找到**ImGui**官方源代码,编译一份**example_win32_directx11.exe**,手动增加一下**SwapChain**地址输出.官方库地址:**https://github.com/ocornut/imgui**

3. 运行得到输出:**g_pSwapChain:00007FFA24E17000**

4. 在**CE**中搜索并进行指针扫描

ID	Description	ShowAsSigned	ShowAsHex	VariableType	Address
0	SwapChain		1	8 Bytes	1C8BD143B20

ID	Description	LastState Value	LastState RealAddress	VariableType	Address	Offset(倒序)
0	指针扫描结果	618754048	1C8BD143B20	4 Bytes	"POLYGON-Win64-Shipping.exe"+08157630	0, 70, 90, 68, 10
1	指针扫描结果	618754048	1C8BD143B20	4 Bytes	"POLYGON-Win64-Shipping.exe"+08244B40	0, 70, 80, 528, 10
3	指针扫描结果	618754048	1C8BD143B20	4 Bytes	"POLYGON-Win64-Shipping.exe"+0827D5B8	0, 70, 20, 778, 110
4	指针扫描结果	618754048	1C8BD143B20	4 Bytes	"POLYGON-Win64-Shipping.exe"+0827D5C0	0, 70, 20, 778, 110
5	指针扫描结果	618754048	1C8BD143B20	4 Bytes	"POLYGON-Win64-Shipping.exe"+08296C30	0, 70, 20, 748, 110
6	指针扫描结果	618754048	1C8BD143B20	4 Bytes	"POLYGON-Win64-Shipping.exe"+08296C38	0, 70, 20, 748, 110
7	指针扫描结果	618754048	1C8BD143B20	4 Bytes	"POLYGON-Win64-Shipping.exe"+082C7F60	0, 130, 10, 390, 110
8	指针扫描结果	618754048	1C8BD143B20	4 Bytes	"POLYGON-Win64-Shipping.exe"+082C7F60	0, 130, 10, 398, 110
9	指针扫描结果	618754048	1C8BD143B20	4 Bytes	"POLYGON-Win64-Shipping.exe"+082C7F68	0, 130, 10, 390, 110
10	指针扫描结果	618754048	1C8BD143B20	4 Bytes	"POLYGON-Win64-Shipping.exe"+082C7F68	0, 130, 10, 398, 110
11	指针扫描结果	618754048	1C8BD143B20	4 Bytes	"POLYGON-Win64-Shipping.exe"+0817B270	0, 10, 0, 110, 160
12	指针扫描结果	618754048	1C8BD143B20	4 Bytes	"POLYGON-Win64-Shipping.exe"+0817B270	0, 10, 8, 110, 160
13	指针扫描结果	618754048	1C8BD143B20	4 Bytes	"POLYGON-Win64-Shipping.exe"+082312C8	0, 10, 0, 110, 160
14	指针扫描结果	618754048	1C8BD143B20	4 Bytes	"POLYGON-Win64-Shipping.exe"+082312C8	0, 10, 8, 110, 160
15	指针扫描结果	618754048	1C8BD143B20	4 Bytes	"opencv_world455.dll"+02723310	0, 10, 0, 110, 3E8
16	指针扫描结果	618754048	1C8BD143B20	4 Bytes	"opencv_world455.dll"+02723310	0, 10, 8, 110, 3E8

5. 重新开游戏简单筛选一下,两个应该是都可以用的

ID	Description	LastState Value	LastState RealAddress	VariableType	Address	Offset(倒序)
0	指针扫描结果	618754048	1ED4E835FB0	4 Bytes	"POLYGON-Win64-Shipping.exe"+0817B270	0, 10, 0, 110, 160
1	指针扫描结果	618754048	1ED4E835FB0	4 Bytes	"POLYGON-Win64-Shipping.exe"+082312C8	0, 10, 0, 110, 160

6. 最终选用

ID	Description	VariableType	Address	Offset(倒序)
0	指针扫描结果	4 Bytes	"POLYGON-Win64-Shipping.exe"+0817B270	0, 10, 0, 110, 160

> **Note**
>
> 在同一台设备上,**Dx虚表位置"固定"**

GName

> **Important**
>
> 有关**GName**的寻找原理请参见前文,本文不做赘述

1. 先通过字符串熟练地找到`void __fastcall FNamePool_FNamePool(__int64 a1)`
2. 分析交叉引用,如下表:

Direction	Type	Address	Text
Down	p	sub_142B09630+47	call FNamePool_FNamePool
Down	p	sub_142B09740+48	call FNamePool_FNamePool
Down	p	sub_142B0A890+30	call FNamePool_FNamePool
Down	p	sub_142B0A950+30	call FNamePool_FNamePool
Down	p	sub_142B0AA10+30	call FNamePool_FNamePool
Down	p	sub_142B0B100+4E	call FNamePool_FNamePool
Down	p	sub_142B0BCF0+4E	call FNamePool_FNamePool
Down	p	sub_142B0C260+30	call FNamePool_FNamePool
Down	p	sub_142B0CCB0+95	call FNamePool_FNamePool
Down	p	sub_142B0CE90+1F	call FNamePool_FNamePool
Down	p	sub_142B0D0B0+21	call FNamePool_FNamePool
Down	p	sub_142B0D110+21	call FNamePool_FNamePool
Down	p	sub_142B0D540+28	call FNamePool_FNamePool
Down	p	sub_142B0D5D0+28	call FNamePool_FNamePool
Down	p	sub_142B0D670+28	call FNamePool_FNamePool
Down	p	sub_142B0D700+28	call FNamePool_FNamePool
Down	p	sub_142B0DA90+28	call FNamePool_FNamePool
Down	p	sub_142B10080+72	call FNamePool_FNamePool

Direction	Type	Address	Text
Down	p	sub_142B159A0+25B	call FNamePool_FNamePool
Down	p	sub_142B16D10+3A	call FNamePool_FNamePool
Down	p	sub_142B16D10+D1	call FNamePool_FNamePool
Down	p	sub_142B16EA0+3F	call FNamePool_FNamePool
Down	p	sub_142B16F90+30	call FNamePool_FNamePool
Down	o	.pdata:0000000148513348	RUNTIME_FUNCTION <rva FNamePool_FNamePool, rva algn_142B092C5,
Up	p	sub_1407D8E60+154	call FNamePool_FNamePool
Up	p	sub_1407D90C0+154	call FNamePool_FNamePool
Up	p	sub_142B04A20+A3	call FNamePool_FNamePool
Up	p	sub_142B04B20+C3	call FNamePool_FNamePool
Up	p	sub_142B04D00+FE	call FNamePool_FNamePool
Up	p	sub_142B04D00+25B	call FNamePool_FNamePool
Up	p	sub_142B05050+A5	call FNamePool_FNamePool

> **Note**
>
> `Down` 表示当前函数调用了 `FNamePool_FNamePool` 函数。
>
> `Up` 表示 `FNamePool_FNamePool` 函数被当前函数调用。

3. 依次看看几个Up调用内容,不要找太长的函数,也尽量避开明显提到其他组件的函数,要时时刻刻切记我们寻找的只是简单的通过

```
static bool bNamePoolInitialized;
alignas(FNamePool) static uint8 NamePoolData[sizeof(FNamePool)];
```

这种方式进行的构造函数调用,在为数不多的Up调用中寻找到以下符合要求的伪函数

```
_DWORD *__fastcall sub_142B04A20(_DWORD *a1, _BYTE *a2)
{
  bool v2; // zf
  _BYTE *v3; // r8
  __int64 v5; // rax
  int v6; // eax
  RTL_SRWLOCK *v7; // rax
  _DWORD *result; // rax
  const char *v9; // [rsp+20h] [rbp-18h] BYREF
  int v10; // [rsp+28h] [rbp-10h]
  char v11; // [rsp+2Ch] [rbp-Ch]
  int v12; // [rsp+40h] [rbp+8h] BYREF
  int v13; // [rsp+44h] [rbp+Ch]
```

```
  v2 = *a2 == 0;
  v3 = a2 + 2;
  v9 = a2 + 2;
  v5 = -1i64;
  if ( v2 )
  {
    do
      ++v5;
    while ( v3[v5] );
    v11 = 0;
  }
  else
  {
    do
      ++v5;
    while ( *(_WORD *)&v3[2 * v5] );
    v11 = 1;
  }
  v10 = v5;
  if ( (unsigned int)v5 < 0x400 )
  {
    if ( byte_148089CF9 )
    {
      v7 = &stru_1480AD880;
    }
    else
    {
      FNamePool_FNamePool((__int64)&stru_1480AD880);
      byte_148089CF9 = 1;
    }
    sub_142B16A60(v7, &v12, &v9);
    v13 = v12;
    v6 = v12;
  }
  else
  {
    v10 = 24;
    v9 = "ERROR_NAME_SIZE_EXCEEDED";
    v11 = 0;
    v6 = sub_142B0CCB0(&v9, 1i64);
  }
  *a1 = v6;
  result = a1;
  a1[1] = 0;
  return result;
}
```

4. **GName**偏移通过计算 **0x1480AD880-0x140000000=0x80AD880** 得到,偏移为 **0x80AD880**
5. 我们开**CheatEngine**简单检验一下,也确实是这个结果,我们有充分的理由认为,**GName**地址是 **“POLYGON-Win64-Shipping.exe”+0x80AD880**
6. 通过以下代码验证**GName**正确性:

```
std::string GetName(uint32_t Id)
{
```

```
    uint32_t Block = Id >> 16;
    uint32_t Offset = Id & 65535;
    uint8_t* GameBase = (uint8_t*)GetModuleHandleA("POLYGON-Win64-
Shipping.exe");

    uint8_t** GName = (uint8_t**)(GameBase + 0x80AD880);

    FNameEntry* Info = (FNameEntry*)((GName)[2 + Block] + 2 * Offset);

    return std::string(Info->AnsiName, Info->Len);
}

 printf("Name:%s\n", GetName(0).c_str());
```

得到输出:

```
Name:None
```

GWorld

1. 在**UnrealEngine.cpp**源代码中寻找如下函数

```
UWorld* UEngine::GetWorldFromContextObject(const UObject* Object,
EGetWorldErrorMode ErrorMode) const
{
    if (Object == nullptr)
    {
        switch (ErrorMode)
        {
        case EGetWorldErrorMode::Assert:
            check(Object);
            break;
        case EGetWorldErrorMode::LogAndReturnNull:
            FFrame::KismetExecutionMessage(TEXT("A null object was passed as
a world context object to UEngine::GetWorldFromContextObject()."),
ELogVerbosity::Warning);
            //UE_LOG(LogEngine, Warning,
TEXT("UEngine::GetWorldFromContextObject() passed a nullptr"));
            break;
        case EGetWorldErrorMode::ReturnNull:
            break;
        }
        return nullptr;
    }

    bool bSupported = true;
    UWorld* World = (ErrorMode == EGetWorldErrorMode::Assert) ? Object-
>GetWorldChecked(/*out*/ bSupported) : Object->GetWorld();
    if (bSupported && (World == nullptr) && (ErrorMode ==
EGetWorldErrorMode::LogAndReturnNull))
    {
        FFrame::KismetExecutionMessage(*FString::Printf(TEXT("No world was
found for object (%s) passed in to UEngine::GetWorldFromContextObject()."),
*GetPathNameSafe(Object)), ELogVerbosity::Warning);
```

```
    }
    return (bSupported ? World : GWorld);
}
```

它以**UWorld***作为返回值,在函数中有大量明文字符串可用来作为特征寻找该函数

> **Note**
>
> 如果你发现你在**IDA**中无法搜索到这些字符串,请设置一下识别的字符串风格,把**unicode**加进去

2.
```
.rdata:00000001470E6BE0 aANullObjectWas:                        ; DATA XREF:
sub_144FEE480+1F↑o
.rdata:00000001470E6BE0                 text "UTF-16LE", 'A null object was
passed as a world context object '
.rdata:00000001470E6C46                 text "UTF-16LE", 'to
UEngine::GetWorldFromContextObject().',0
```

```
__int64 __fastcall sub_144FEE480(__int64 a1, __int64 a2, int a3)
{
  __int64 v4; // rsi
  __int64 v6; // rax
  __int64 v7; // rdi
  const wchar_t *v8; // rbx
  const wchar_t *v9; // r8
  __int64 v10; // rdx
  const wchar_t *v11; // [rsp+20h] [rbp-28h] BYREF
  int v12; // [rsp+28h] [rbp-20h]
  const wchar_t *v13; // [rsp+30h] [rbp-18h] BYREF
  int v14; // [rsp+38h] [rbp-10h]
  __int64 v15; // [rsp+58h] [rbp+10h] BYREF

  v4 = a2;
  if ( a2 )
  {
    LOBYTE(v15) = 1;
    if ( a3 == 2 )
      v6 = sub_142C63C80(a2, &v15);
    else
      v6 = (*(__int64 (__fastcall **)(__int64))(*(_QWORD *)a2 + 392i64))(a2);
    v7 = v6;
    if ( !(_BYTE)v15 )
      return qword_1482ACFD0;
    if ( !v6 && a3 == 1 )
    {
      sub_142CD1520(v4, &v13, 0i64);
      v8 = &chText;
      v9 = &chText;
      if ( v14 != (_DWORD)v7 )
        v9 = v13;
      sub_1429C9990(&v11, L"No world was found for object (%s) passed in to
UEngine::GetWorldFromContextObject().", v9);
      LOBYTE(v10) = 3;
      if ( v12 != (_DWORD)v7 )
        v8 = v11;
      sub_142CAF290(v8, v10, 0i64);
      if ( v11 )
```

```
        sub_142A062C0();
      if ( v13 )
        sub_142A062C0();
    }
    if ( !(_BYTE)v15 )
      return qword_1482ACFD0;
    return v7;
  }
  else
  {
    if ( a3 == 1 )
    {
      v15 = 0i64;
      LOBYTE(a2) = 3;
      sub_142CAF290(
        L"A null object was passed as a world context object to
UEngine::GetWorldFromContextObject().",
        a2,
        0i64);
    }
    return 0i64;
  }
}
```

3. 锁定`return qword_1482ACFD0`,直接计算**0x1482ACFD0-0x0x140000000=0x82ACFD0**,偏移为**0x82ACFD0**

GObject

1. 还是先看引擎源码,在**UObjectHash.cpp**,有**GObject**的定义

```
// Global UObject array instance
FUObjectArray GUObjectArray;
```

2. 分析对**GUObjectArray**的引用,有很多含有字符串的函数可以作为寻找**UObject**的跳板,我选择了这个函数

```
void UObjectBaseInit()
{
    SCOPED_BOOT_TIMING("UObjectBaseInit");

    // Zero initialize and later on get value from .ini so it is overridable
per game/ platform...
    int32 MaxObjectsNotConsideredByGC = 0;
    int32 SizeOfPermanentObjectPool = 0;
    int32 MaxUObjects = 2 * 1024 * 1024; // Default to ~2M UObjects
    bool bPreAllocateUObjectArray = false;

    // To properly set MaxObjectsNotConsideredByGC look for "Log: XXX objects
as part of root set at end of initial load."
    // in your log file. This is being logged from LaunchEnglineLoop after
objects have been added to the root set.

    // Disregard for GC relies on seekfree loading for interaction with
linkers. We also don't want to use it in the Editor, for which
```

```
    // FPlatformProperties::RequiresCookedData() will be false. Please note
that GIsEditor and FApp::IsGame() are not valid at this point.
    if (FPlatformProperties::RequiresCookedData())
    {
        if (IsRunningCookOnTheFly())
        {
            GCreateGCClusters = false;
        }
        else
        {
            GConfig->GetInt(TEXT("/Script/Engine.GarbageCollectionSettings"),
TEXT("gc.MaxObjectsNotConsideredByGC"), MaxObjectsNotConsideredByGC,
GEngineIni);

            // Not used on PC as in-place creation inside bigger pool
interacts with the exit purge and deleting UObject directly.
            GConfig->GetInt(TEXT("/Script/Engine.GarbageCollectionSettings"),
TEXT("gc.SizeOfPermanentObjectPool"), SizeOfPermanentObjectPool, GEngineIni);
        }

        // Maximum number of UObjects in cooked game
        GConfig->GetInt(TEXT("/Script/Engine.GarbageCollectionSettings"),
TEXT("gc.MaxObjectsInGame"), MaxUObjects, GEngineIni);

        // If true, the UObjectArray will pre-allocate all entries for
UObject pointers
        GConfig->GetBool(TEXT("/Script/Engine.GarbageCollectionSettings"),
TEXT("gc.PreAllocateUObjectArray"), bPreAllocateUObjectArray, GEngineIni);
    }
    else
    {
#if IS_PROGRAM
        // Maximum number of UObjects for programs can be low
        MaxUObjects = 100000; // Default to 100K for programs
        GConfig->GetInt(TEXT("/Script/Engine.GarbageCollectionSettings"),
TEXT("gc.MaxObjectsInProgram"), MaxUObjects, GEngineIni);
#else
        // Maximum number of UObjects in the editor
        GConfig->GetInt(TEXT("/Script/Engine.GarbageCollectionSettings"),
TEXT("gc.MaxObjectsInEditor"), MaxUObjects, GEngineIni);
#endif
    }

    if (MaxObjectsNotConsideredByGC <= 0 && SizeOfPermanentObjectPool > 0)
    {
        // If permanent object pool is enabled but disregard for GC is
disabled, GC will mark permanent object pool objects
        // as unreachable and may destroy them so disable permanent object
pool too.
        // An alternative would be to make GC not mark permanent object pool
objects as unreachable but then they would have to
        // be considered as root set objects because they could be
referencing objects from outside of permanent object pool.
        // This would be inconsistent and confusing and also counter
productive (the more root set objects the more expensive MarkAsUnreachable
phase is).
```

```
        SizeOfPermanentObjectPool = 0;
        UE_LOG(LogInit, Warning, TEXT("Disabling permanent object pool
because disregard for GC is disabled (gc.MaxObjectsNotConsideredByGC=%d)."),
MaxObjectsNotConsideredByGC);
    }

    // Log what we're doing to track down what really happens as log in
LaunchEngineLoop doesn't report those settings in pristine form.
    UE_LOG(LogInit, Log, TEXT("%s for max %d objects, including %i objects
not considered by GC, pre-allocating %i bytes for permanent pool."),
        bPreAllocateUObjectArray ? TEXT("Pre-allocating") :
TEXT("Presizing"),
        MaxUObjects, MaxObjectsNotConsideredByGC, SizeOfPermanentObjectPool);

    GUObjectAllocator.AllocatePermanentObjectPool(SizeOfPermanentObjectPool);
    GUObjectArray.AllocateObjectPool(MaxUObjects,
MaxObjectsNotConsideredByGC, bPreAllocateUObjectArray);
#if UE_WITH_OBJECT_HANDLE_LATE_RESOLVE

UE::CoreUObject::Private::InitObjectHandles(GUObjectArray.GetObjectArrayCapac
ity());
#endif

    void InitGarbageElimination();
    InitGarbageElimination();

    void InitAsyncThread();
    InitAsyncThread();

    // Note initialized.
    Internal::GetUObjectSubsystemInitialised() = true;

    UObjectProcessRegistrants();
}
//上边的函数调用下边的函数
void FUObjectArray::AllocateObjectPool(int32 InMaxUObjects, int32
InMaxObjectsNotConsideredByGC, bool bPreAllocateObjectArray)
{
    check(IsInGameThread());

    MaxObjectsNotConsideredByGC = InMaxObjectsNotConsideredByGC;

    // GObjFirstGCIndex is the index at which the garbage collector will
start for the mark phase.
    // If disregard for GC is enabled this will be set to an invalid value so
that later we
    // know if disregard for GC pool has already been closed (at least once)
    ObjFirstGCIndex = DisregardForGCEnabled() ? -1 : 0;

    // Pre-size array.
    check(ObjObjects.Num() == 0);
    UE_CLOG(InMaxUObjects <= 0, LogUObjectArray, Fatal, TEXT("Max UObject
count is invalid. It must be a number that is greater than 0."));
    ObjObjects.PreAllocate(InMaxUObjects, bPreAllocateObjectArray);

    if (MaxObjectsNotConsideredByGC > 0)
```

```
    {
        ObjObjects.AddRange(MaxObjectsNotConsideredByGC);
    }
}
```

3. 在IDA中定位到源码后,分析这部分

```
if (MaxObjectsNotConsideredByGC <= 0 && SizeOfPermanentObjectPool > 0)
{
    // If permanent object pool is enabled but disregard for GC is disabled,
GC will mark permanent object pool objects
    // as unreachable and may destroy them so disable permanent object pool
too.
    // An alternative would be to make GC not mark permanent object pool
objects as unreachable but then they would have to
    // be considered as root set objects because they could be referencing
objects from outside of permanent object pool.
    // This would be inconsistent and confusing and also counter productive
(the more root set objects the more expensive MarkAsUnreachable phase is).
    SizeOfPermanentObjectPool = 0;
    UE_LOG(LogInit, Warning, TEXT("Disabling permanent object pool because
disregard for GC is disabled (gc.MaxObjectsNotConsideredByGC=%d)."),
MaxObjectsNotConsideredByGC);
}
//对应下面的伪代码 ReName了变量
 if ( MaxObjectsNotConsideredByGC <= 0 && SizeOfPermanentObjectPool > 0 )
 {
   v1 = 0;
   SizeOfPermanentObjectPool = 0;
   if ( (unsigned __int8)byte_14807DBE0 >= 3u )
   {
     sub_142A654A0(&byte_14807DBE0, &off_146698318);
     v1 = SizeOfPermanentObjectPool;
   }
}
```

4. 在MaxObjectsNotConsideredByGC = InMaxObjectsNotConsideredByGC时,类成员变量被赋值,对应伪代码

```
dword_148153F28 = MaxObjectsNotConsideredByGC;
//以下是类成员分布

//  /** First index into objects array taken into account for GC.
*/
//  int32 ObjFirstGCIndex;
//  /** Index pointing to last object created in range disregarded for GC.
*/
//  int32 ObjLastNonGCIndex;
//  /** Maximum number of objects in the disregard for GC Pool */
//  int32 MaxObjectsNotConsideredByGC;
```

5. 定位类索引首地址,就是类全局变量的地址,用**0x148153F28-0x8-0x140000000=0x8153F20**