

Brute Forcing Internal Unreal Engine Structures

Proof of Concept Documentation

31/03/2008

The following document will show examples that require some expected knowledge on the internals of the Unreal Engine (Core module). This method can be applied to any Unreal Engine 3 implementation, however Unreal Tournament 3 was used in this document as it is one of the newer builds.

Programs used:

OllyDbg 1.10

The Interactive Disassembler 5.2 (Make sure you have IDA cache Unicode strings)

Unreal Tournament 3 Demo

Unreal Tournament 2003

Introduction:

The general idea of this method is to use the global object and name tables to find instances of class properties, then use the property offset variable to get the relative positions of the variables in the structure.

Some pre-requisites are required to do this:

Structures:

TArray (It still follows the same format as the UE1 Core)

FNameEntry (Slightly modified, I will explain how to update the structure later)

Instances:

GObjObjects TArray

Names TArray

Offsets:

The static offset to the Name variable from UObject

Getting the Prerequisites:

To find the prerequisites we're going to use cross-referencing from the Core.dll in UT2003 and try to find similar instances in the disassembled code of the UE3 application.

First we have to find all the references of GObjObjects in Core.dll then follow each reference and look for a point of reference that you will be able to compare with. Try to make use of character arrays that look semi-important.

Please take note that very little modification to the Core has been made throughout the years. Generally most reference points you find will be the same in all the different builds of the core, so once you have a point of reference, try and find similar instances in the UE3 disassembled code.

In the following examples it shows a string being used as the reference point and the assumption that the address found is indeed GObjObjects.

NOTE: It may be hard to follow the images so I will just include the disassembled code after them.

The screenshot shows a debugger window titled "MyDog - Core.dll [CPU - main thread, module Core.dll]". The main pane displays disassembled assembly code. Key annotations include:

- Core,10191940 UNICODE "Other object in slot" <- Point of reference
- <- GObjObjects instance moved to ecx
- <- ecx is the array and eax is the index
- UNICODE "This is 3a"
- UNICODE "This is 3a"

The bottom pane shows a memory dump with addresses and hex values. The Command window at the bottom contains the text "Entry point of debugged DLL".

Proof of Concept Documentation by Miles Goodings (Tamimego)

IDA - C:\Program Files\Unreal Tournament 3 Demo\Binaries\UT3Demo.exe - [IDA View-A]

File Edit Jump Search View Debugger Options Windows Help

Hex View-A Exports Imports Names Functions Structures Comments Strings

```

.text:00AF587E
.text:00AF587E loc_AF587E: ; CODE XREF: sub_AF5740+EAtj
.text:00AF587E cmp [ecx+eax*4], esi
.text:00AF5881 jz loc_AF589E
.text:00AF5887 mov ecx, off_1C8CEC0
.text:00AF588D push offset aOtherObjectInS ; "Other object in slot"
.text:00AF5892 push 2Fh
.text:00AF5897 push ecx
.text:00AF5898 call sub_AC3CE0
.text:00AF589D add esp, 0Ch
.text:00AF58A0 push 0
.text:00AF58A2 lea edx, [esp+30h+var_24]
.text:00AF58A6 push edx
.text:00AF58A7 mov ecx, esi
.text:00AF58AB call sub_AF16F0
.text:00AF58AE mov ecx, [eax*4]
.text:00AF58B1 test ecx, ecx
.text:00AF58B3 mov [esp+2Ch+var_4], 2
.text:00AF58B8 jz short loc_AF58C1
.text:00AF58BD mov eax, [eax]
.text:00AF58BF jmp short loc_AF58C6
.text:00AF58C1 ;
.text:00AF58C1 loc_AF58C1: ; CODE XREF: sub_AF5740+170tj
.text:00AF58C1 mov eax, offset word_16CECC
.text:00AF58C6 ; CODE XREF: sub_AF5740+17Ftj
.text:00AF58C6 loc_AF58C6:
.text:00AF58C6 push eax
.text:00AF58C7 mov eax, off_1C8CEC0
.text:00AF58CC push offset aThisIsS_1 ; "This is: %s"
.text:00AF58D1 push 2Fh
.text:00AF58D6 push eax
.text:00AF58DB call sub_AC3CE0

```

00AF5880 00AF5880: sub_AF5740+140

Flushing buffers, please wait...
 File 'C:\Program Files\Unreal Tournament 3 Demo\Binaries\UT3Demo.exe' is successfully loaded into the database.
 Compiling file 'C:\Program Files\IDA\idc\idc.idc'...
 Executing function 'main'...
 Compiling file 'C:\Program Files\IDA\idc\onload.idc'...
 Executing function 'Onload'...
 IDA is analysing the input file...
 You may start to explore the input file right now.
 Union 5: 787 3100000 - Microsoft Visual C++ 7.0 first runtime
 ACQUAITE: Down: Disk: 7GB. Current location:

IDA - C:\Program Files\Unreal Tournament 3 Demo\binaries\UT3demo.exe - [IDA View A]

File Edit Jump Search View Debugger Options Windows Help

Text dword_102C43C

IDA View A Hex View A Exports Imports Names Functions Structures Run Strings

```

.text:00AF58B8      jz     short loc_AF58C1
.text:00AF58BD      mov    eax, [eax]
.text:00AF58BF      jmp    short loc_AF58C6
;-----
.text:00AF58C1      loc_AF58C1:                                     ; CODE XREF: sub_AF5740+17B1j
.text:00AF58C1      mov    eax, offset word_16CECC
;-----
.text:00AF58C6      loc_AF58C6:                                     ; CODE XREF: sub_AF5740+17F1j
.text:00AF58C6      push   eax
.text:00AF58C7      mov    eax, off_1CDBCE0
.text:00AF58C8      push   offset aThisIs1_1 ; "This is: %s"
.text:00AF58C9      push   2Fh
.text:00AF58CA      push   eax
.text:00AF58CB      call   sub_AC3CE0
.text:00AF58CC      add    esp, 10h
.text:00AF58CD      lea    ecx, [esp+2Ch+var_24]
.text:00AF58CE      mov    [esp+2Ch+var_4], 0FFFFFFFh
.text:00AF58CF      call   sub_A0A860
.text:00AF58D0      mov    esi, [esi+4]
.text:00AF58D1      mov    edx, dword_102C43C <- This is GobJObjects
.text:00AF58D2      push   0
.text:00AF58D3      lea    ecx, [esp+30h+var_18]
.text:00AF58D4      push   ecx
.text:00AF58D5      mov    ecx, [edx+esi+4] <- Getting the element from the array, edx is the array and esi is the index
.text:00AF58D6      call   sub_AF4CF0
.text:00AF58D7      cmp    dword ptr [eax+h], 0
.text:00AF58D8      mov    [esp+2Ch+var_4], 3
.text:00AF58D9      jz     short loc_AF591A
.text:00AF58DA      mov    eax, [eax]
.text:00AF58DB      jmp    short loc_AF591F
;-----
.text:00AF591A      .text:00AF591A

```

< 006F58F3 00AF58F3: sub_AF5740+1B3

Flushing buffers, please wait...
 File 'C:\Program Files\Unreal Tournament 3 Demo\binaries\UT3demo.exe' is successfully loaded into the database.
 Compiling file 'C:\Program Files\IDA\idc\idc1.idc'...
 Executing function 'main'...
 Compiling file 'C:\Program Files\IDA\idc\onload.idc'...
 Executing function 'onload'...
 IDA is analyzing the input file...
 You may start to explore the input file right now.
 Using 512K stringtable. Microsoft Visual C++ 6.0 not running.

AC-016CB1A6 Done. Disk: 35B

Core.dll Disassembled Text (OllyDbg):

```

10139CEE . 8B15 50F61810 MOV EDX,DWORD PTR DS:[?GLog@@3PAVFOutputDevice@@A] ; Core.10191940
10139CF4 . 68 C8201610 PUSH Core.101620C8 ; UNICODE
"Other object in slot"
10139CF9 . 68 FF020000 PUSH 2FF
10139CFE . 52 PUSH EDX
10139CFF . E8 FC47FFFF CALL Core.?Logf@FOutputDevice@@QAAXW4EName@@PBGZZ
10139D04 . 8B46 04 MOV EAX,DWORD PTR DS:[ESI+4]
10139D07 . 8B0D 88A02510 MOV ECX,DWORD PTR DS:[?GObjObjects@UObject@@0V?$TArr>
10139D0D . 8B0C81 MOV ECX,DWORD PTR DS:[ECX+EAX*4]
10139D10 . 83C4 0C ADD ESP,0C
10139D13 . 6A 00 PUSH 0 ; /Arg1 =
00000000
10139D15 . E8 56CAFFFF CALL Core.?GetFullName@UObject@@QBEPBGPAG@Z ;
\?GetFullName@UObject@@QBEPBGPAG@Z
10139D1A . 8B15 50F61810 MOV EDX,DWORD PTR DS:[?GLog@@3PAVFOutputDevice@@A] ; Core.10191940
10139D20 . 50 PUSH EAX
10139D21 . 68 AC201610 PUSH Core.101620AC ; UNICODE
"Other is: %s"
10139D26 . 68 FF020000 PUSH 2FF
10139D2B . 52 PUSH EDX
10139D2C . E8 CF47FFFF CALL Core.?Logf@FOutputDevice@@QAAXW4EName@@PBGZZ
10139D31 . 83C4 10 ADD ESP,10
10139D34 . 6A 00 PUSH 0 ; /Arg1 =
00000000
10139D36 . 8BCE MOV ECX,ESI ; |
10139D38 . E8 33CAFFFF CALL Core.?GetFullName@UObject@@QBEPBGPAG@Z ;
\?GetFullName@UObject@@QBEPBGPAG@Z
10139D3D . 50 PUSH EAX
10139D3E . A1 50F61810 MOV EAX,DWORD PTR DS:[?GLog@@3PAVFOutputDevice@@A]
10139D43 . 68 0C211610 PUSH Core.1016210C ; UNICODE "This
is: %s"
10139D48 . 68 FF020000 PUSH 2FF
10139D4D . 50 PUSH EAX
10139D4E . E8 AD47FFFF CALL Core.?Logf@FOutputDevice@@QAAXW4EName@@PBGZZ
10139D53 . 83C4 10 ADD ESP,10
10139D56 . 33C0 XOR EAX,EAX
10139D58 . 8B4D F4 MOV ECX,DWORD PTR SS:[EBP-C]
10139D5B . 64:890D 000000>MOV DWORD PTR FS:[0],ECX

```

UT3.exe Disassembled Text (IDA):

```

.text:00AF5887      mov     ecx, off_1CBCEC0
.text:00AF588D      push    offset aOtherObjectIns ; "Other object in slot"
.text:00AF5892      push    2FFh
.text:00AF5897      push    ecx
.text:00AF5898      call   sub_AC3CE0
.text:00AF589D      add     esp, 0Ch
.text:00AF58A0      push    0
.text:00AF58A2      lea     edx, [esp+30h+var_24]
.text:00AF58A6      push    edx
.text:00AF58A7      mov     ecx, esi
.text:00AF58A9      call   sub_AF1CF0
.text:00AF58AE      mov     ecx, [eax+4]
.text:00AF58B1      test    ecx, ecx
.text:00AF58B3      mov     [esp+2Ch+var_4], 2
.text:00AF58BB      jz      short loc_AF58C1
.text:00AF58BD      mov     eax, [eax]
.text:00AF58BF      jmp     short loc_AF58C6
.text:00AF58C1      ; -----
.text:00AF58C1      loc_AF58C1:      ; CODE XREF: sub_AF5740+17Bj
.text:00AF58C1      mov     eax, offset word_16ECECC
.text:00AF58C6      loc_AF58C6:      ; CODE XREF: sub_AF5740+17Fj
.text:00AF58C6      push    eax
.text:00AF58C7      mov     eax, off_1CBCEC0
.text:00AF58CC      push    offset aThisIsS_1 ; "This is: %s"
.text:00AF58D1      push    2FFh
.text:00AF58D6      push    eax
.text:00AF58D7      call   sub_AC3CE0
.text:00AF58DC      add     esp, 10h
.text:00AF58DF      lea     ecx, [esp+2Ch+var_24]
.text:00AF58E3      mov     [esp+2Ch+var_4], 0FFFFFFFFh
.text:00AF58EB      call   sub_404860
.text:00AF58F0      mov     esi, [esi+4]
.text:00AF58F3      mov     edx, dword_1D2C43C
.text:00AF58F9      push    0
.text:00AF58FB      lea     ecx, [esp+30h+var_18]
.text:00AF58FF      push    ecx
.text:00AF5900      mov     ecx, [edx+esi*4]
.text:00AF5903      call   sub_AF1CF0
.text:00AF5908      cmp     dword ptr [eax+4], 0
.text:00AF590C      mov     [esp+2Ch+var_4], 3
.text:00AF5914      jz      short loc_AF591A
.text:00AF5916      mov     eax, [eax]
.text:00AF5918      jmp     short loc_AF591F
.text:00AF591A      ; -----
.text:00AF591A      loc_AF591A:      ; CODE XREF: sub_AF5740+1D4j
.text:00AF591A      mov     eax, offset word_16ECECC
.text:00AF591F      loc_AF591F:      ; CODE XREF: sub_AF5740+1D8j
.text:00AF591F      push    eax
.text:00AF5920      mov     eax, off_1CBCEC0
.text:00AF5925      push    offset aOtherIsS ; "Other is: %s"
.text:00AF592A      push    2FFh
.text:00AF592F      push    eax
.text:00AF5930      call   sub_AC3CE0
.text:00AF5935      lea     ecx, [esp+3Ch+var_18]
.text:00AF5939      jmp     loc_AF57FD
.text:00AF593E      ; -----
.text:00AF593E      loc_AF593E:      ; CODE XREF: sub_AF5740+141j
.text:00AF593E      mov     eax, 1
.text:00AF5943      mov     ecx, [esp+2Ch+var_C]

```

<- This is Gobjobjects

Getting the Prerequisites (Continued):

The name table requires a bit more effort to find. From the variants of the UE3 I've seen so far, all of them use some new format for retrieving the names. I haven't had the need to reverse the new method since directly reading the names from the name table works just as well.

In the following disassembly I find a reference to the name table in Core.dll and look for a similar instance in the UT3.exe, just like we did with object table. After that, I trace back the first string pushed to appSprintf. It leads to a function that takes the object as ecx and then later adds the static position of the Name variable to the register. Then there is a sub-function called, which has a reference to the name table.

NOTE: The reference string I used was "%s[%i]"

Core.dll Disassembled Text (OllyDbg):

```

1013B654 . 8B4B 20      MOV ECX,DWORD PTR DS:[EBX+20]
1013B657 . 8B15 F45F2510 MOV EDX,DWORD PTR DS:[?Names@FName@@0V?$TArray@PAUFN>
1013B65D . 8B048A      MOV EAX,DWORD PTR DS:[EDX+ECX*4]
1013B660 . 57          PUSH EDI
1013B661 . 83C0 0C     ADD EAX,0C
1013B664 . 50          PUSH EAX
1013B665 . 8D8D 9CFDFFF LEA ECX,DWORD PTR SS:[EBP-264]
1013B66B . 68 18261610 PUSH Core.10162618 ; UNICODE
"%s[%i]"
1013B670 . 51          PUSH ECX
1013B671 . E8 4A88FDFF CALL Core.?appSprintf@@YAHPAGPBGGZ
1013B676 . 83C4 10     ADD ESP,10

```

UT3.exe Disassembled Text (IDA):

```

.text:004F7C3C      lea      eax, [esp+0A4h+var_30]
.text:004F7C40      push    eax
.text:004F7C41      call    sub_40D430      <- eax is set from here
.text:004F7C46      mov     [esp+0A4h+var_4], 8
.text:004F7C51      mov     ebx, 20h
.text:004F7C56      loc_4F7C56: ; CODE XREF: sub_4F78C0+373j
.text:004F7C56      cmp     [eax+4], esi
.text:004F7C59      mov     [esp+0A4h+var_90], ebx
.text:004F7C5D      jz      short loc_4F7C63
.text:004F7C5F      mov     eax, [eax]
.text:004F7C61      jmp     short loc_4F7C68
.text:004F7C63 ; -----
.text:004F7C63      loc_4F7C63: ; CODE XREF: sub_4F78C0+39Dj
.text:004F7C63      mov     eax, offset word_16ECECC
.text:004F7C68      loc_4F7C68: ; CODE XREF: sub_4F78C0+3A1j
.text:004F7C68      mov     ecx, [edi+10h]
.text:004F7C6B      push    ecx
.text:004F7C6C      push    eax      <- This is the name in eax
.text:004F7C6D      lea     edx, [esp+0ACh+var_3C]
.text:004F7C71      push    offset aSI_7 ; "%s[%i]"
.text:004F7C76      push    edx
.text:004F7C77      call    sub_426E90
.text:004F7C7C      add     esp, 10h

```

Function sub_40D430:

```

.text:0040D430 sub_40D430      proc near                                ; CODE XREF: sub_40E8D0+DEp
.text:0040D430                                                         ; sub_422930+8Ap ...
.text:0040D430
.text:0040D430 var_4          = dword ptr -4
.text:0040D430 arg_0          = dword ptr 8
.text:0040D430
.text:0040D430      push     ebp
.text:0040D431      mov      ebp, esp
.text:0040D433      push     ecx
.text:0040D434      test     ecx, ecx
.text:0040D436      push     esi
.text:0040D437      mov      esi, [ebp+arg_0]
.text:0040D43A      mov      [ebp+var_4], 0
.text:0040D441      jnz      short loc_40D458
.text:0040D443      push     offset aNone_0 ; "None"
.text:0040D448      mov      ecx, esi
.text:0040D44A      call     sub_40D380
.text:0040D44F      mov      eax, esi
.text:0040D451      pop      esi
.text:0040D452      mov      esp, ebp
.text:0040D454      pop      ebp
.text:0040D455      retn     4
.text:0040D458 ; -----
.text:0040D458 loc_40D458:                                         ; CODE XREF: sub_40D430+11j
.text:0040D458      cmp      dword ptr [ecx+4], 0FFFFFFFFh
.text:0040D45C      jnz      short loc_40D473
.text:0040D45E      push     offset aUninitialized ; "<uninitialized>"
.text:0040D463      mov      ecx, esi
.text:0040D465      call     sub_40D380
.text:0040D46A      mov      eax, esi
.text:0040D46C      pop      esi
.text:0040D46D      mov      esp, ebp
.text:0040D46F      pop      ebp
.text:0040D470      retn     4
.text:0040D473 ; -----
.text:0040D473 loc_40D473:                                         ; CODE XREF: sub_40D430+2Cj
.text:0040D473      push     esi
.text:0040D474      add      ecx, 2Ch                <- 2Ch UObject->Name
.text:0040D477      call     sub_AEA910             <- FName to Unknown container
.text:0040D47C      mov      eax, esi
.text:0040D47E      pop      esi
.text:0040D47F      mov      esp, ebp
.text:0040D481      pop      ebp
.text:0040D482      retn     4
.text:0040D482 sub_40D430      endp
.text:0040D482 ; -----
.text:0040D485      align 10h

```

Function sub_AEA910:

```
.text:00AEA910 sub_AEA910      proc near                ; CODE XREF: sub_40D430+47p
.text:00AEA910                                     ; sub_40FAE0+1C5p ...
.text:00AEA910
.text:00AEA910 var_10          = dword ptr -10h
.text:00AEA910 var_C           = dword ptr -0Ch
.text:00AEA910 var_4          = dword ptr -4
.text:00AEA910 arg_0          = dword ptr  4
.text:00AEA910
.text:00AEA910                push    0FFFFFFFh
.text:00AEA912                push    offset loc_15CE699
.text:00AEA917                mov     eax, large fs:0
.text:00AEA91D                push    eax
.text:00AEA91E                push    ecx
.text:00AEA91F                push    ebx
.text:00AEA920                push    ebp
.text:00AEA921                push    esi
.text:00AEA922                push    edi
.text:00AEA923                mov     eax, dword_1CC1BB0
.text:00AEA928                xor     eax, esp
.text:00AEA92A                push    eax
.text:00AEA92B                lea     eax, [esp+24h+var_C]
.text:00AEA92F                mov     large fs:0, eax
.text:00AEA935                mov     ebx, ecx
.text:00AEA937                xor     ebp, ebp
.text:00AEA939                mov     [esp+24h+var_10], ebp
.text:00AEA93D                mov     eax, [ebx]
.text:00AEA93F                mov     ecx, dword_1D1AB80    <- This is the Name table
.text:00AEA945                mov     edi, [ecx+eax*4]
.text:00AEA948                mov     esi, [esp+24h+arg_0]
.text:00AEA94C                add     edi, 10h           <- 10h FNameEntry->Name
.text:00AEA94F                mov     [esi], ebp
.text:00AEA951                mov     [esi+4], ebp
.text:00AEA954                mov     [esi+8], ebp
.text:00AEA957                mov     eax, edi
.text:00AEA959                mov     [esp+24h+var_4], ebp
.text:00AEA95D                mov     [esp+24h+var_10], 1
.text:00AEA965                lea     edx, [eax+2]
.text:00AEA968                jmp     short loc_AEA970
```


Code Examples:

Now that we have working instances of the name and object tables, you have access to thousands of object instances. Here's an example showing a dump of the object table:

```
// UnrealArray based off the Core432 SDK definition of TArray
template<class T> struct UnrealArray
{
    T*          Data;
    unsigned long Length;
    unsigned long Max;
};

// UnrealObject based off our reversal
struct UnrealObject
{
    unsigned char      Unknown      [0x2C];
    unsigned long      NameIndex;
};

// UnrealName based off our reversal
struct UnrealName
{
    unsigned char      Unknown      [0x10];
    unsigned short     Name         [1];
};

void Dump ( void )
{
    // Setup a logged output and the instances
    FILE* Log = fopen ( "Dump.log", "w+" );
    UnrealArray<UnrealObject*>* GlobalObjects;
    UnrealArray<UnrealName*>* GlobalNames;

    GlobalObjects = (UnrealArray<UnrealObject*>*) 0x1D2C43C;
    GlobalNames = (UnrealArray<UnrealName*>*) 0x1D1AB80;

    // Loop through the object table
    for ( unsigned long i = 0; i < GlobalObjects->Length; i++ )
    {
        // check if it's a valid object
        if ( !GlobalObjects->Data [i] )
            continue;

        // Check if the name index is valid
        unsigned long NameIndex = GlobalObjects->Data [i]->NameIndex;
        if ( NameIndex < 0 || NameIndex > GlobalNames->Length )
            continue;

        // Check if the name entry is valid
        if ( !GlobalNames->Data [NameIndex] )
            continue;

        // Write the object index and name to the log
        fprintf ( Log, "Object[%04i] %S\n", i, GlobalNames->Data [NameIndex]->Name );
    }

    // Close the log
    fclose ( Log );
}
```

Here are the first ten objects outputted from the dump:

```
Object[0000]   TextBufferFactory
Object[0001]   Factory
Object[0002]   Object
Object[0003]   TextBuffer
Object[0004]   System
Object[0005]   Subsystem
Object[0006]   StructProperty
Object[0007]   Property
Object[0008]   Field
Object[0009]   StrProperty
```

Brute Force Overview:

Now that we have all the required things to start brute forcing the structures, I will explain how it's going to work. In UProperty there is a variable whose value represents the current properties offset in the class.

So the general idea is to get an instance of the Name UProperty from UObject and scan the bytes until we find 2Ch which is the position of UObject->Name. From there we can get any UProperty instance and get the relative offset in the class it belongs to.

The three important variables for UObject we need are Outer, Name and Class. Since we have Name, all we need is Class and Outer.

So what you have to do is:

1. Scan the object table until you find the UProperty of Name (The first instance of a object with the name of "Name")
2. Use this UProperty to get the offset to UProperty->Offset using the brute force trick
3. Find the UProperty of Outer and get its relative offset
4. Find the UProperty of Class and get its relative offset

Now using these three variables you can create full object names in the style of the UE2 object names (Class Package.ClassName.ObjectName). With this you can find any variable's UProperty of your choice in the object table and get the relative offset.

Also you can brute force other such variables as PropertySize from UStruct. The idea behind this is demonstrated in the proof of concept code below.

Once you have this you need to set up two loops. One that adds 4 bytes to the class offset per loop and another loop inside that scans for so many bytes, until it finds the temporary value made from the class offset.

This method may seem a bit trivial, but I've used it many times and it's never failed. The class offset should be one of the last properties in the UObject structure.

PropertySize contains the size of class, so if we increment the class offset a few times, each going up by four (Since the class size will always be aligned to 4 bytes).

Then just scan for that value for a range of bytes and it will eventually hit the PropertySize. If it finds more than one, that then narrows it down a lot as compared to manually finding it.

After you have these variables you can use them as reference points to find other offsets. For example UStruct->Children is always 4 bytes after PropertySize.

You can then use the PropertySize in another way such as, UObject->PropertySize would be the offset of UField->Super, UField->PropertySize would be the offset of UProperty->ArrayDim, etc.

You can find many values just by using PropertySize, PropertyOffset and some guessing.

Original Proof of Concept Code with Comments (Not for c&ping):

```
// Static character arrays for the string compares
WCHAR* Object_Name      = L"Name";
WCHAR* Object_Outer     = L"Outer";
WCHAR* Object_Class     = L"Class";
WCHAR* Object_Object    = L"Object";

// Some storage pointers for saved values
int      Object_Start    = 0;
DWORD    Object_ClassPtr = 0x0;
DWORD    Offset_Max      = 0x150;
DWORD    Offset_MaxObjects = 0x4;
DWORD    Offset_Name     = 0x2C;
DWORD    Offset_Outer    = 0x0;
DWORD    Offset_Class     = 0x0;
DWORD    Offset_PropertyOffset = 0x0;
DWORD    Offset_PropertySize = 0x0;

// Loop the object table
for ( unsigned long i = 0; i < ObjectManager->Count; i++ )
{
    DWORD Object = (DWORD) ObjectManager->Data[i];
    // Check if the object is valid
    if ( !Object )
        continue;
    // Get the name index
    DWORD Name = *(PDWORD) ( (DWORD) Object + (DWORD) Offset_Name );
    // Find the Name UProperty
    if ( wcscmp ( NameManager->Data[Name]->Name, Object_Name ) == 0 )
    {
        // Scan the range of bytes to the size of Offset_Max
        for ( DWORD j = Offset_Name; j < Offset_Max; j++ )
        {
            DWORD Offset = *(PDWORD) ( (DWORD) Object + (DWORD) j );
            // Check if the offset matches the UObject->Name offset
            if ( Offset == Offset_Name )
            {
                // Store the PropertyOffset offset
                Offset_PropertyOffset = j;
                // Take the current object index and move it back a few entries
                // Outer starts before Name so we have to do this for the next loop
                Object_Start = i - (Offset_Max / 4);
                goto JmpOne;
            }
        }
    }
}

// Check if we found a PropertyOffset offset
JmpOne:
if ( !Offset_PropertyOffset )
    return;

// Loop the object table
for ( unsigned long i = Object_Start; i < ObjectManager->Count; i++ )
{
    DWORD Object = (DWORD) ObjectManager->Data[i];
    // Check if the object is valid
    if ( !Object )
        continue;
    // Get the name index
    DWORD Name = *(PDWORD) ( (DWORD) Object + (DWORD) Offset_Name );
    if ( !Offset_Outer )
    {
        // Check if the object name matches Object_Outer
        if ( wcscmp ( NameManager->Data[Name]->Name, Object_Outer ) == 0 )
        {
            // Get the relative offset for the property
            Offset_Outer = *(PDWORD) ((DWORD) Object + (DWORD) Offset_PropertyOffset);
        }
    }
    if ( !Offset_Class )
    {
        // Check if the object name matches Object_Class
        if ( wcscmp ( NameManager->Data[Name]->Name, Object_Class ) == 0 )
        {
            // Get the relative offset for the property
            Offset_Class = *(PDWORD) ((DWORD) Object + (DWORD) Offset_PropertyOffset);
        }
    }
}
```

Proof of Concept Documentation by Miles Goodings (Tamimego)

```
// Check if we found both the outer and class offsets
if ( !Offset_Outer || !Offset_Class )
    return;

// Loop the object table
for ( unsigned long i = 0; i < ObjectManager->Count; i++ )
{
    DWORD Object = (DWORD) ObjectManager->Data[i];
    // Check if the object is valid
    if ( !Object )
        continue;

    DWORD ObjectName      = *(PDWORD) ( (DWORD) Object + (DWORD) Offset_Name );
    DWORD Class           = *(PDWORD) ( (DWORD) Object + (DWORD) Offset_Class );
    DWORD ClassName       = *(PDWORD) ( (DWORD) Class + (DWORD) Offset_Name );

    // Compare if the class and object name match that of the UObject UClass
    if ( ( wcscmp ( NameManager->Data[ClassName]->Name, Object_Class ) == 0 )
        && ( wcscmp ( NameManager->Data[ObjectName]->Name, Object_Object ) == 0 ) )
    {
        // Found the UClass instance for UObject
        Object_ClassPtr = Object;
        goto JmpTwo;
    }
}

// Check if we got a valid UClass instance
JmpTwo:
if ( !Object_ClassPtr )
    return;

// Loop for the size of MaxObjects
for ( unsigned long i = 0; i < Offset_MaxObjects; i++ )
{
    // Create the predicted class size variable
    DWORD Temp = ( Offset_Class + 0x4 + ( i * 0x4 ) );
    // Scan the range of bytes to the size of Offset_Max
    for ( unsigned long j = Offset_Class; j < Offset_Max; j++ )
    {
        DWORD Offset = *(PDWORD) ( (DWORD) Object_ClassPtr + (DWORD) j );
        // Compare the value to the predicted class size
        if ( Offset == Temp )
        {
            // Found possible PropertySize offset
            Offset_PropertySize = j;
            goto JmpThree;
        }
    }
}

// Check if we found a PropertySize offset
JmpThree:
if ( !Offset_PropertySize )
    return;

// Log the resulting values
fprintf ( Log, "\nObject:\n" );
fprintf ( Log, "\t- Outer\t\t\t0x%X\n",      Offset_Outer );
fprintf ( Log, "\t- Name\t\t\t0x%X\n",      Offset_Name );
fprintf ( Log, "\t- Class\t\t\t0x%X\n",      Offset_Class );
fprintf ( Log, "\nUProperty:\n" );
fprintf ( Log, "\t- PropertyOffset\t0x%X\n", Offset_PropertyOffset );
fprintf ( Log, "\nUStruct:\n" );
fprintf ( Log, "\t- PropertySize\t\t0x%X\n", Offset_PropertySize );
```

Here is the dump:

```

UObject:
    - Outer          0x28
    - Name           0x2C
    - Class          0x34

UProperty:
    - PropertyOffset 0x64

UStruct:
    - PropertySize   0x50

```