INSIDE THE ULTIMA ONLINE GOLD DEMO - UODEMO.DAT versus DIRECT FILE ACCESS

GOAL

It's our goal to get a deep understanding of how the Ultima Online Gold Demo works. This demo is a representation of the rule set from the Ultima Online Second Age Era.

There is proof that some people have already reversed this demo partially or as a whole, however so far no tools or knowledge has been published. This project is to overcome those shortcomings.

URL's with some proof for this:
http://www.runuo.com/forums/general-discussion/94767-help-m-files.html
http://azaroth.org/2008/12/31/your-topic/ (posting by Faust)

If we understand the demo there is a big chance we can alter the demo and even create our own demo. By default mounting horses is not possible in the demo, but what if we can alter the demo and unlock horses; can we then see how horses behaved during T2A?

This demo is 10 years old and I do not understand no one published his/her work. Maybe that DMCA thing is in the way?

UTILITIES USED

IDA Pro, a very professional utility, definitely worth buying, Standard version is affordable.
HxD, a very neat hex editor and above all, it's free

ABOUT ME

I'm just a guy who loves the Ultima universe and knows a bit assembler. Why not combine the two? ☺ I've been into computer starting from age twelve, and Ultima VII was the first game I bought myself. The "coolest" thing I ever did may well be the patch described in this document, but the "coolest" thing I did 15 years ago happened in school. I was reading about viruses because I wondered how they worked. Back then I discovered how to write programs with DEBUG.COM (MSDOS application). So instead of attending the lessons and doing boring exercises I was fooling around in assembler. I once patched COMMAND.COM so it would display a welcome message every time the computer started, much like a virus would patch COMMAND.COM :=). The patch didn't last long, I was very afraid to get caught so I removed it at the end of the 2 hour lesson. But it worked my dear reader ☺.

STRUCTURES

I have already uncovered how UODEMO.DAT is encrypted and in the mean-time I delved deeper in the code and documented more functions and structures involved. I'm going to share you my work.

I start by showing you the (internal) structures I partially analyzed:

```
00000000 struct_DAT_HeaderEntry struc ; (sizeof=0x118)
00000000 FileName        db 260 dup(?)           ; string(C)
00000104 PointerToFileData dd ?
00000108 ReservedSize?   dd ?
0000010C IsReadOnly      dd ?
00000110 StoredSize      dd ?
00000114 CurrentPosition dd ?
00000118 struct_DAT_HeaderEntry ends
00000118
00000000 ; ---------------------------------------------------------------
00000000
00000000 struct_ContainerHandle struc ; (sizeof=0x2C)
00000000 ContainerID     dd ?
00000004 MemoryBlock     dd ?
00000008 field_8         dd ?
0000000C field_C         dd ?
00000010 CurrentPosition dd ?
00000014 field_14        dd ?
00000018 OriginalLength  dd ?
0000001C ActualLength    dd ?
00000020 field_20        dd ?
00000024 field_24        dd ?
00000028 AccessThroughContainerCode dd ?
0000002C struct_ContainerHandle ends
0000002C
00000000 ; ---------------------------------------------------------------
00000000
00000000 struct_UODEMODAT struc ; (sizeof=0xCD274)
00000000 field_0         dd 70 dup(?)
00000118 FileHandleToUODEMODAT dd ?
0000011C embedded        struct_UODEMODATembedded ?
0000012C HeaderList      struct_DAT_HeaderEntry 3000 dup(?)
000CD26C HeaderCount     dd ?
000CD270 PointerToBeginOfDataArea dd ?
000CD274 struct_UODEMODAT ends
000CD274
00000000 ; ---------------------------------------------------------------
00000000
00000000 struct_UODEMODATembedded struc ; (sizeof=0x10)
00000000 field_0         dd ?
00000004 field_4         dd ?
00000008 field_8         dd ?
0000000C field_C         dd ?
00000010 struct_UODEMODATembedded ends
```

Even though I called them "struct", they are more likely to be classes but know that a C++ class is nothing more than an advanced structure under the hood.

“struct_UODEMODAT” is an internal structure and most importantly it contains an array of 3000 header structures. This means that the UO Demo supports only up to 3000 files stored in UODEMO.DAT.

“struct_DAT_HeaderEntry” is exposed to the real world in its encrypted form. To learn how to decrypt the structures you will have to read my first document about UODEMO.DAT. Note that the filename is an array of 260 characters (bytes). 260 is a magic value in the Windows world as it’s defined as MAXPATH. Very bad btw because NTFS supports longer filenames but due to this limitation very few software can work with longer filenames, even the crappy Windows Explorer can’t handle those!

“struct_ContainerHandle” is another internal structure; it is used when reading data from UODEMO.DAT. As you can see, I didn’t document all fields yet. The files are handled in blocks of 64KB and some of those fields seem to indicate which block is currently in memory. Please note that UODEMO.EXE also supports writing to UODEMO.DAT.

“struct_UODEMODATembedded” is still a complete mystery. It looks like an embedded C++ object but I couldn’t derive its function yet.

FOPEN

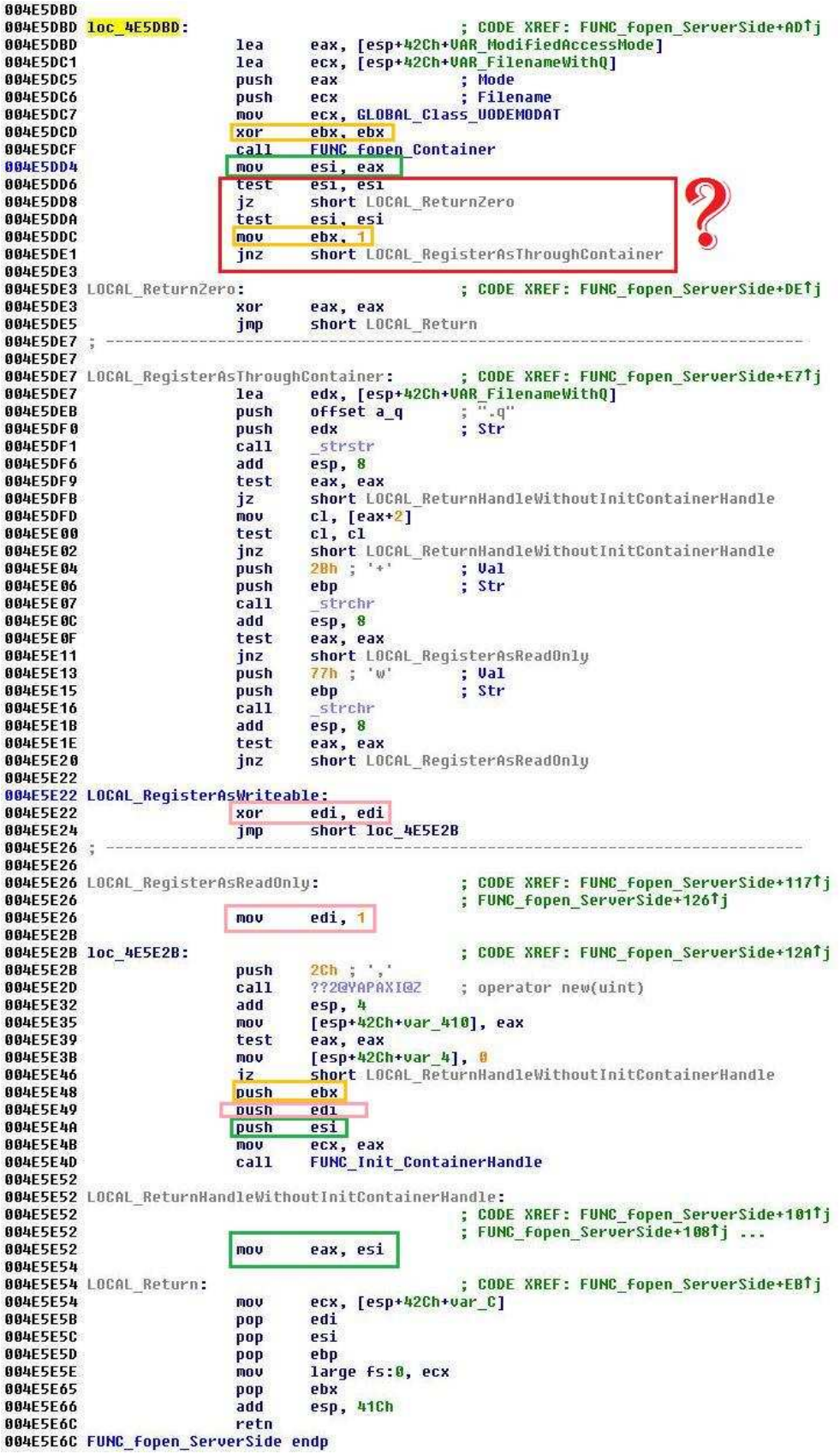
The UODEMO.DAT contains a function at address 0x004E5CFA that is called every time the demo wants to open a file. The function will return a pointer to a struct_DAT_HeaderEntry variable.

Screenshot of the start of this function which I named FUNC_fopen_ServerSide:

```
004E5CFA ; =============== S U B R O U T I N E =======================================
004E5CFA
004E5CFA
004E5CFA FUNC_fopen_ServerSide proc near         ; CODE XREF: sub_40107D+2D↑p
004E5CFA                                         ; sub_43361F+145↑p ...
004E5CFA
004E5CFA VAR_ModifiedAccessMode= byte ptr -41Ch
004E5CFA VAR_AllocatedMemoryForContainerHandle= dword ptr -418h
004E5CFA VAR_FilenameWithQ= byte ptr -40Ch
004E5CFA var_C           = dword ptr -0Ch
004E5CFA var_4           = dword ptr -4
004E5CFA ARG_Filename    = dword ptr  4
004E5CFA ARG_OpenMode    = dword ptr  8
004E5CFA
004E5CFA                 mov     eax, large fs:0
004E5D00                 push    0FFFFFFFFh
004E5D02                 push    offset unknown_libname_691 ; Microsoft VisualC 2-8/net runtime
004E5D07                 push    eax
004E5D08                 mov     large fs:0, esp
004E5D0F                 sub     esp, 418h
004E5D15                 or      ecx, 0FFFFFFFFh
004E5D18                 xor     eax, eax
004E5D1A                 push    ebx
004E5D1B                 push    ebp
004E5D1C                 push    esi
004E5D1D                 push    edi
004E5D1E                 mov     edi, [esp+42Ch+ARG_Filename]
004E5D25                 lea     edx, [esp+42Ch+VAR_FilenameWithQ]
004E5D29                 repne scasb
004E5D2B                 not     ecx
004E5D2D                 sub     edi, ecx
004E5D2F                 mov     eax, ecx
004E5D31                 mov     esi, edi
004E5D33                 mov     edi, edx
004E5D35                 shr     ecx, 2
004E5D38                 rep movsd
004E5D3A                 mov     ecx, eax
004E5D3C                 and     ecx, 3
004E5D3F                 rep movsb
004E5D41                 lea     ecx, [esp+42Ch+VAR_FilenameWithQ]
004E5D45                 push    ecx              ; String
004E5D46                 call    __strlwr
```

The function also does something interesting, the ARG_Filename is a function argument and is converted to a lower-case string and ".q" is added, this result is stored in VAR_FilenameWithQ. After that the ARG_OpenMode is also modified a bit, but nothing important.

Screenshot of the ending of FUNC_fopen_ServerSide:



The image on the previous page required a devoted page because it's a very important screenshot.

Do you know assembler? Then look at the red square and you will completely understand why I put a question mark there. You don't know assembler? Then read on.

Now, what's going on is: FUNC_fopen_Container is called, this name is a bit misleading because UODEMO.DAT is already open but that function will return the address of a struct_DAT_HeaderEntry if the file is found in UODEMO.DAT, otherwise NULL is returned.

The return value, the address, is stored in the EAX register which is then copied to the ESI register. Now if ESI is 0 then a jump is made to LOCAL_ReturnZero which will return to the caller and the demo will fail because the file couldn't be opened. Then 1 is put into EBX, notice that EBX was zeroed before the call (orange squares). Then a zero test is made again on the ESI register but this time a jump is made to LOCAL_RegisterAsThroughContainer if ESI isn't zero.

But we already know that ESI isn't zero because of the first test!

That made me think there was code sitting around that has been removed from the final build, this idea has been in the back of my head since the first time I started working on the UODEMO and noticed that ESI test.

Now, we also know that EBX will always be 1! All files that are needed by UODEMO are stored in UODEMO.DAT so why is EBX zeroed out first? That's really an indicator that there was extra code that we don't see anymore.

Another interesting fact, look at the _strstr call at address 0x004E5DF1. So if the VAR_FilenameWithQ does not contain ".q" then the ESI parameter is returned without executing the rest of the code. But we know that ".q" is added always! So why is there an extra check at the end? What in God's name or Allah's name, is that code doing there. It's really not necessary. But it is necessary if code exists that would accesses files without UODEMO.DAT!

Also, let's take a look at the EDI register, it will be 1 if the modified access mode does not contain "w" or a "+". This is C stuff and basically if EDI is zero then the file is writeable; if EDI is 1 then the file is read-only.

After that, memory for a struct_ContainerHandle is allocated and this structure/object is initialized. 3 important parameters, EBX, EDI and ESI! EBX = 1 (because we read from UODEMO.DAT), EDI is a read-only/writeable indicator and ESI is the address of the header entry.

FCLOSE

This is the function that will close files opened by FUNC_fopen_ServerSide:

```
004E5F5A ; =============== S U B R O U T I N E =======================================
004E5F5A
004E5F5A
004E5F5A FUNC_fclose_ServerSide proc near        ; CODE XREF: sub_40107D+139↑p
004E5F5A                                         ; sub_4034E0+17↑p ...
004E5F5A
004E5F5A FileHandleOrFilename= dword ptr  4
004E5F5A
004E5F5A                push    esi
004E5F5B                push    edi
004E5F5C                mov     edi, [esp+8+FileHandleOrFilename]
004E5F60                push    edi
004E5F61                call    sub_4E557A
004E5F66                mov     esi, eax
004E5F68                add     esp, 4
004E5F6B                test    esi, esi
004E5F6D                jnz     short loc_4E5F7B
004E5F6F                push    edi             ; File
004E5F70                call    _fclose
004E5F75                add     esp, 4
004E5F78                pop     edi
004E5F79                pop     esi
004E5F7A                retn
004E5F7B ; ---------------------------------------------------------------------------
004E5F7B
004E5F7B loc_4E5F7B:                             ; CODE XREF: FUNC_fclose_ServerSide+13↑j
004E5F7B                mov     ecx, esi
004E5F7D                call    FUNC_fclose_Container__WithFlush
004E5F82                mov     ecx, esi
004E5F84                mov     edi, eax
004E5F86                call    sub_4E50FA
004E5F8B                push    esi             ; Memory
004E5F8C                call    ??3@YAXPAX@Z    ; operator delete(void *)
004E5F91                add     esp, 4
004E5F94                mov     eax, edi
004E5F96                pop     edi
004E5F97                pop     esi
004E5F98                retn
004E5F98 FUNC_fclose_ServerSide endp
```

This is very interesting and again, did no one notice this before? What is _fclose doing there? Files are never opened directly from disk but always from UODEMO.DAT, so why is a _fclose needed?

Know that FUNC_fclose_ServerSide takes a struct_DAT_HeaderEntry as parameter.
Know that _fclose takes a FILE as parameter (C documentation).
Know that FUNC_fclose_Container__WithFlush takes a struct_ContainerHandle as parameter.

FGETS

This is the function that gets called when a text file is being handled, for example server.txt:

```
004E5F9A ; =============== S U B R O U T I N E =======================================
004E5F9A
004E5F9A
004E5F9A FUNC_fgets_ServerSide proc near         ; CODE XREF: sub_40107D+78↑p
004E5F9A                                         ; COMMAND_initArrayFromFile+9A↑p ...
004E5F9A
004E5F9A Buf             = dword ptr  4
004E5F9A MaxCount        = dword ptr  8
004E5F9A FileHandleOrFilename= dword ptr  0Ch
004E5F9A
004E5F9A                 push    esi
004E5F9B                 mov     esi, [esp+4+FileHandleOrFilename]
004E5F9F                 push    esi
004E5FA0                 call    sub_4E557A
004E5FA5                 add     esp, 4
004E5FA8                 test    eax, eax
004E5FAA                 jnz     short loc_4E5FC1
004E5FAC                 mov     eax, [esp+4+MaxCount]
004E5FB0                 mov     ecx, [esp+4+Buf]
004E5FB4                 push    esi              ; File
004E5FB5                 push    eax              ; MaxCount
004E5FB6                 push    ecx              ; Buf
004E5FB7                 call    _fgets
004E5FBC                 add     esp, 0Ch
004E5FBF                 pop     esi
004E5FC0                 retn
004E5FC1 ; ---------------------------------------------------------------------------
004E5FC1
004E5FC1 loc_4E5FC1:                             ; CODE XREF: FUNC_fgets_ServerSide+10↑j
004E5FC1                 mov     edx, [esp+4+MaxCount]
004E5FC5                 mov     ecx, [esp+4+Buf]
004E5FC9                 push    edx
004E5FCA                 push    ecx
004E5FCB                 mov     ecx, eax
004E5FCD                 call    FUNC_fgets_Container
004E5FD2                 pop     esi
004E5FD3                 retn
004E5FD3 FUNC_fgets_ServerSide endp
```

If you don't know what "fgets" does then google or learn the C language.

The same comments of FUNC_fclose_ServerSide apply to this function! What is _fgets doing there!?

Know that FUNC_fgets_ServerSide takes a struct_DAT_HeaderEntry as parameter.
Know that _fgets takes a FILE as parameter (C documentation).
Know that FUNC_fgets_Container takes a struct_ContainerHandle as parameter.

DOING THE MATH

Two plus two is one. Sorry, it's one plus one is two.

Now what's going on is: both FUNC_fclose_ServerSide and FUNC_gets_ServerSide call a function sub_4E557A. If this function returns NULL then the C functions (fgets/fclose) are called directly with the same parameter as sub_4E557A was called with. If the function didn't return NULL than the FUNC_..._Container functions are called. Because the FUNC_..._Container functions take a struct_ContainerHandle as parameter we can derive that sub_4E557A returns a struct_ContainerHandle which will be NULL if it can't find the struct_DAT_HeaderEntry.

Now I also looked at sub_4E557A and there stuff going which is related with calls made by FUNC_Init_ContainerHandle (see FUNC_fopen_ServerSide). So when FUNC_fopen_ServerSide returns a header entry, this header entry will have been linked internally with a struct_ContainerHandle by FUNC_Init_ContainerHandle. Remember that if the file name doesn't contain ".q" no struct_ContainerHandle will be initialized! But we also know that a file without ".q" will never be found in UODEMO.DAT so the demo will always fail.

Remember that we had that red square with weird code? Well, I think there was code that would call _fopen and return a FILE handle instead of a struct_DAT_HeaderEntry. I wanted to test this by adding a call to _fopen and I tested it.

You know what? IT WORKED!

GIVE ME SPACE

There is obviously not enough space to add calls to fopen and so-on. This isn't the same as patching a jump or something; we're going to add real code.

I found this function:

```
004E60DA ; =============== S U B R O U T I N E =======================================
004E60DA
004E60DA
004E60DA FUNC_rename_ServerSide proc near
004E60DA
004E60DA var_208         = byte ptr -208h
004E60DA var_104         = byte ptr -104h
004E60DA OldFilename     = dword ptr  4
004E60DA NewFilename     = dword ptr  8
004E60DA
004E60DA                 sub     esp, 208h
004E60E0                 push    ebx
004E60E1                 mov     ebx, [esp+20Ch+NewFilename]
004E60E8                 push    edi
004E60E9                 mov     edi, [esp+210h+OldFilename]
004E60F0                 push    ebx             ; NewFilename
004E60F1                 push    edi             ; OldFilename
004E60F2                 call    _rename
004E60F7                 add     esp, 8
004E60FA                 test    eax, eax
004E60FC                 jnz     short loc_4E6107
004E60FE
004E60FF
004E6100
004E6106
004E6107 ; ---------------------------------------------------------------------------
004E6107
004E6107 loc_4E610                                       FUNC_rename_ServerSide+22↑j
004E6107
004E610A                 xor     eax, eax
004E610C                 repne scasb
004E610E                 not     ecx
004E6110                 sub     edi, ecx
004E6112                 push    esi
004E6113                 lea     edx, [esp+214h+var_104]
004E611A                 mov     eax, ecx
004E611C                 mov     esi, edi
004E611E                 mov     edi, edx
004E6120                 shr     ecx, 2
004E6123                 rep movsd
004E6125                 mov     ecx, eax
004E6127                 xor     eax, eax
004E6129                 and     ecx, 3
004E612C                 mov     dl, byte ptr a q 1+2
```

Yes, it's a rename function. So somehow the demo has support for renaming files, is this code from the OSI servers? The xrefs tells us that this function is never used inside the demo. That's why I decided to overwrite that function with my own code.

THE PATCH – PART 1

Here's my "modified" rename function:

```
004E60DA                FUNC_rename_ServerSide:                 ; CODE XREF: FUNC_fopen_ServerSide+D5↑p
004E60DA 55             push    ebp
004E60DB 89 E5          mov     ebp, esp
004E60DD FF 75 0C       push    dword ptr [ebp+0Ch]
004E60E0 FF 75 08       push    dword ptr [ebp+8]
004E60E3 E8 15 C7 FF FF call    FUNC_fopen_Container
004E60E8 43             inc     ebx
004E60E9 09 C0          or      eax, eax
004E60EB 75 38          jnz     short LOCAL_Return
004E60ED FF 75 0C       push    dword ptr [ebp+0Ch]
004E60F0 FF 75 08       push    dword ptr [ebp+8]
004E60F3 E8 A8 3A 00 00 call    _fopen
004E60F8 83 C4 08       add     esp, 8
004E60FB 31 DB          xor     ebx, ebx
004E60FD 09 C0          or      eax, eax
004E60FF 74 24          jz      short LOCAL_Return
004E6101 87 45 08       xchg    eax, [ebp+8]
004E6104
004E6104                loc_4E6104:                             ; CODE XREF: .text:004E611D↓j
004E6104 68 48 77 62 00 push    offset a_q                      ; ".q"
004E6109 50             push    eax
004E610A E8 01 36 00 00 call    _strstr
004E610F 83 C4 08       add     esp, 8
004E6112 31 DB          xor     ebx, ebx
004E6114 09 C0          or      eax, eax
004E6116 74 0D          jz      short LOCAL_Return
004E6118 83 C0 02       add     eax, 2
004E611B 38 18          cmp     [eax], bl
004E611D 75 E5          jnz     short loc_4E6104
004E611F 88 58 FE       mov     [eax-2], bl
004E6122 8B 45 08       mov     eax, [ebp+8]
004E6125
004E6125                LOCAL_Return:                           ; CODE XREF: .text:004E60EB↑j
004E6125                                                        ; .text:004E60FF↑j ...
004E6125 5D             pop     ebp
004E6126 C2 08 00       retn    8
```

This code will:

1) call FUNC_fopen_Container
2) on success, return
3) on failure, call fopen
4) on failure, return
5) on success, remove ".q" from the filename (very important to make this work)
6) return

THE PATCH – PART 2

Even though we now have created a cool function that will call _fopen, we still need to make the demo call this new function.

This is done in FUNC_fopen_ServerSide.

Before:

```
004E5DBD loc_4E5DBD:                             ; CODE XREF: FUNC_fopen_ServerSide+AD↑j
004E5DBD                 lea     eax, [esp+42Ch+VAR_ModifiedAccessMode]
004E5DC1                 lea     ecx, [esp+42Ch+VAR_FilenameWithQ]
004E5DC5                 push    eax             ; Mode
004E5DC6                 push    ecx             ; Filename
004E5DC7                 mov     ecx, GLOBAL_Class_UODEMODAT
004E5DCD                 xor     ebx, ebx
004E5DCF                 call    FUNC_fopen_Container
004E5DD4                 mov     esi, eax
004E5DD6                 test    esi, esi
004E5DD8                 jz      short LOCAL_ReturnZero
004E5DDA                 test    esi, esi
004E5DDC                 mov     ebx, 1
004E5DE1                 jnz     short LOCAL_RegisterAsThroughContainer
004E5DE3
004E5DE3 LOCAL_ReturnZero:                       ; CODE XREF: FUNC_fopen_ServerSide+DE↑j
004E5DE3                 xor     eax, eax
004E5DE5                 jmp     short LOCAL_Return
004E5DE7 ; ---------------------------------------------------------------------------
004E5DE7
004E5DE7 LOCAL_RegisterAsThroughContainer:       ; CODE XREF: FUNC_fopen_ServerSide+E7↑j
```

After:

```
:004E5DBD                 loc_4E5DBD:                             ; CODE XREF: FUNC_fopen_ServerSide+AD↑j
:004E5DBD 8D 44 24 10     lea     eax, [esp+42Ch+VAR_ModifiedAccessMode]
:004E5DC1 8D 4C 24 20     lea     ecx, [esp+42Ch+VAR_FilenameWithQ]
:004E5DC5 50              push    eax                             ; Mode
:004E5DC6 51              push    ecx                             ; Filename
:004E5DC7 8B 0D 40 16 70 00 mov     ecx, GLOBAL_Class_UODEMODAT
:004E5DCD 33 DB           xor     ebx, ebx
:004E5DCF E8 06 03 00 00  call    FUNC_rename_ServerSide
:004E5DD4 89 C6           mov     esi, eax
:004E5DD6 85 F6           test    esi, esi
:004E5DD8 74 09           jz      short LOCAL_ReturnZero
:004E5DDA EB 0B           jmp     short LOCAL_RegisterAsThroughContainer
:004E5DDC                 ; ---------------------------------------------------------------------------
:004E5DDC 90              nop
:004E5DDD 90              nop
:004E5DDE 90              nop
:004E5DDF 90              nop
:004E5DE0 90              nop
:004E5DE1 90              nop
:004E5DE2 90              nop
:004E5DE3
:004E5DE3                 LOCAL_ReturnZero:                       ; CODE XREF: FUNC_fopen_ServerSide+DE↑j
:004E5DE3 33 C0           xor     eax, eax
:004E5DE5 EB 6D           jmp     short LOCAL_Return
:004E5DE7                 ; ---------------------------------------------------------------------------
:004E5DE7
:004E5DE7                 LOCAL_RegisterAsThroughContainer:       ; CODE XREF: FUNC_fopen_ServerSide+E0↑j
```

NOTE: the color is different because the second picture was taken while the debugger was active

ADDITIONAL NOTE: don't think I didn't see the demo crash, it took me several tries to make it right, and the first time I didn't remove the ".q" which gave weird results ☺. Also when you do this, make sure you operate on a correct ".rundir", a corrupt ".rundir" will kill the beast most likely.

HOW TO MAKE IT WORK

You can now go ahead and patch your UODEMO.EXE (or UODEMO+.EXE).

But this will never work unless you modify or remove UODEMO.DAT (or UODEMO+.DAT). The fopen function will only be called if the file isn't found inside the DAT archive/container.

You can create an empty DAT file; a 0-bytes long/short DAT file will not crash the demo.

Also, removing UODEMO.DAT will not crash the demo. The program will crash only when it can't find one if the required files it was expecting to find in UODEMO.DAT.