INSIDE THE ULTIMA ONLINE GOLD DEMO
- Environment Variables

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

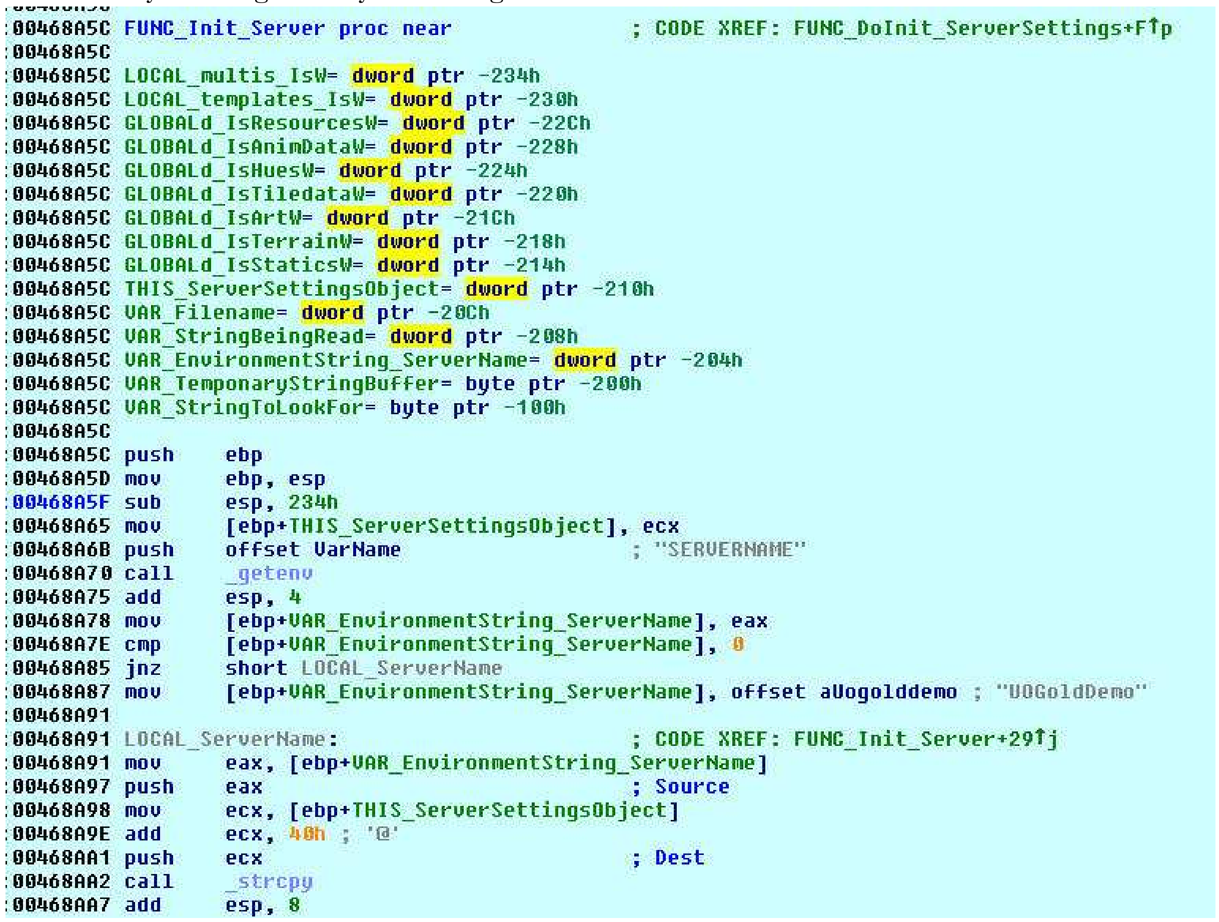
ABOUT ME

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 opening screen of this game is still grafted in my visual memory and I can recall it at any time without any problems.

SERVER.TXT

Inside the uogolddemo subdirectory there is a file “server.txt”, this file contains readable text and I wondered what the values mean, and how they are used inside the demo.

I started my investigation by searching for server.txt and I found this function:



```
:00468A5C FUNC_Init_Server proc near            ; CODE XREF: FUNC_DoInit_ServerSettings+F↑p
:00468A5C
:00468A5C LOCAL_multis_IsW= dword ptr -234h
:00468A5C LOCAL_templates_IsW= dword ptr -230h
:00468A5C GLOBALd_IsResourcesW= dword ptr -22Ch
:00468A5C GLOBALd_IsAnimDataW= dword ptr -228h
:00468A5C GLOBALd_IsHuesW= dword ptr -224h
:00468A5C GLOBALd_IsTiledataW= dword ptr -220h
:00468A5C GLOBALd_IsArtW= dword ptr -21Ch
:00468A5C GLOBALd_IsTerrainW= dword ptr -218h
:00468A5C GLOBALd_IsStaticsW= dword ptr -214h
:00468A5C THIS_ServerSettingsObject= dword ptr -210h
:00468A5C VAR_Filename= dword ptr -20Ch
:00468A5C VAR_StringBeingRead= dword ptr -208h
:00468A5C VAR_EnvironmentString_ServerName= dword ptr -204h
:00468A5C VAR_TemporaryStringBuffer= byte ptr -200h
:00468A5C VAR_StringToLookFor= byte ptr -100h
:00468A5C
:00468A5C push    ebp
:00468A5D mov     ebp, esp
:00468A5F sub     esp, 234h
:00468A65 mov     [ebp+THIS_ServerSettingsObject], ecx
:00468A6B push    offset VarName                 ; "SERVERNAME"
:00468A70 call    _getenv
:00468A75 add     esp, 4
:00468A78 mov     [ebp+VAR_EnvironmentString_ServerName], eax
:00468A7E cmp     [ebp+VAR_EnvironmentString_ServerName], 0
:00468A85 jnz     short LOCAL_ServerName
:00468A87 mov     [ebp+VAR_EnvironmentString_ServerName], offset aUogolddemo ; "UOGoldDemo"
:00468A91
:00468A91 LOCAL_ServerName:                      ; CODE XREF: FUNC_Init_Server+29↑j
:00468A91 mov     eax, [ebp+VAR_EnvironmentString_ServerName]
:00468A97 push    eax                            ; Source
:00468A98 mov     ecx, [ebp+THIS_ServerSettingsObject]
:00468A9E add     ecx, 40h ; '@'
:00468AA1 push    ecx                            ; Dest
:00468AA2 call    _strcpy
:00468AA7 add     esp, 8
```

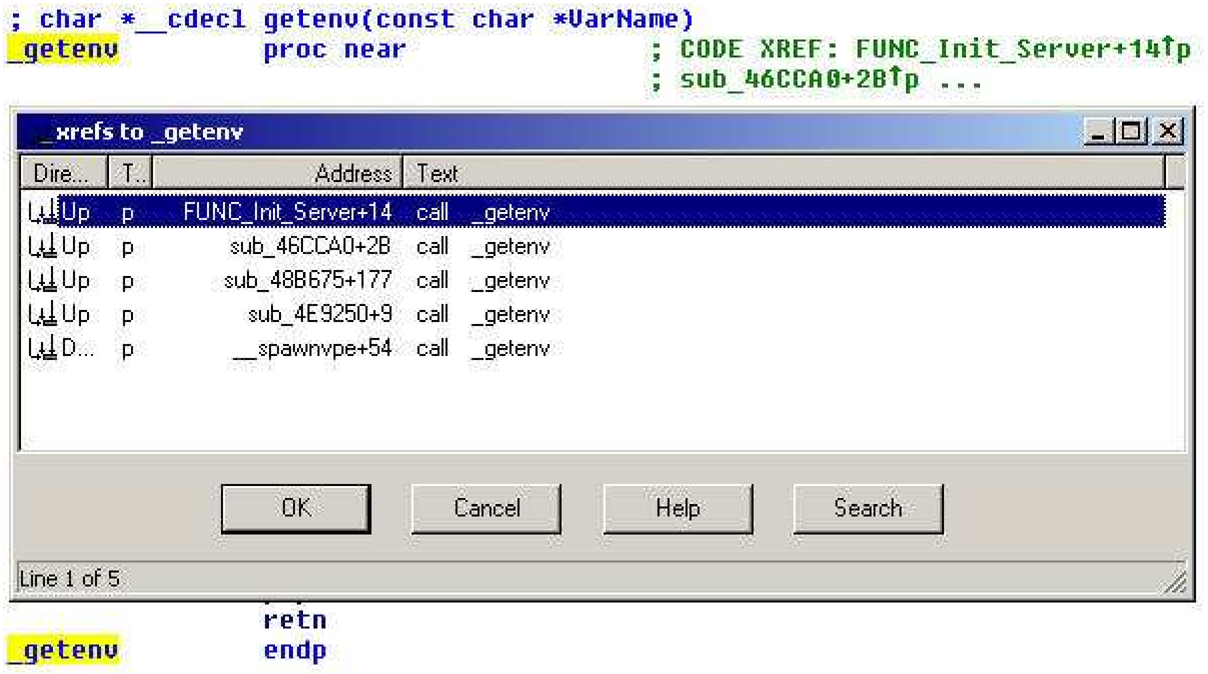
It’s the first call in this function that struck my attention. A call to getenv; why would the demo do that? It is only a demo, is that environment variable used for making the demo do other things? Is it a remainder from the server code? Each OSI server shares the same code base but the environment variable defines which server controls what part of Britannia?

If you look at the assembler code, if the environment variable SERVERNAME does not exist then it will default to UOGoldDemo. That’s the directory where server.txt and some other files are stored in. Thus by setting SERVERNAME you can have those files placed somewhere else. With the default uodemo.exe this will fail because there is no other directory in uodemo.dat. But with the UODEMO.DAT removal patch we can now unlock this technology.

OTHER ENVIRONMENT VARIABLES

Are there other environment variables we can set in the demo? To check this, you have to go to the getenv function and check for cross-references.

Screenshot:



```
; char *__cdecl getenv(const char *VarName)
_getenv         proc near               ; CODE XREF: FUNC_Init_Server+14↑p
                                        ; sub_46CCA0+2B↑p ...
```

xrefs to _getenv

```
Dire...  T..            Address  Text
Up       p    FUNC_Init_Server+14  call  _getenv
Up       p         sub_46CCA0+2B  call  _getenv
Up       p        sub_48B675+177  call  _getenv
Up       p          sub_4E9250+9  call  _getenv
D...     p         __spawnvpe+54  call  _getenv
```

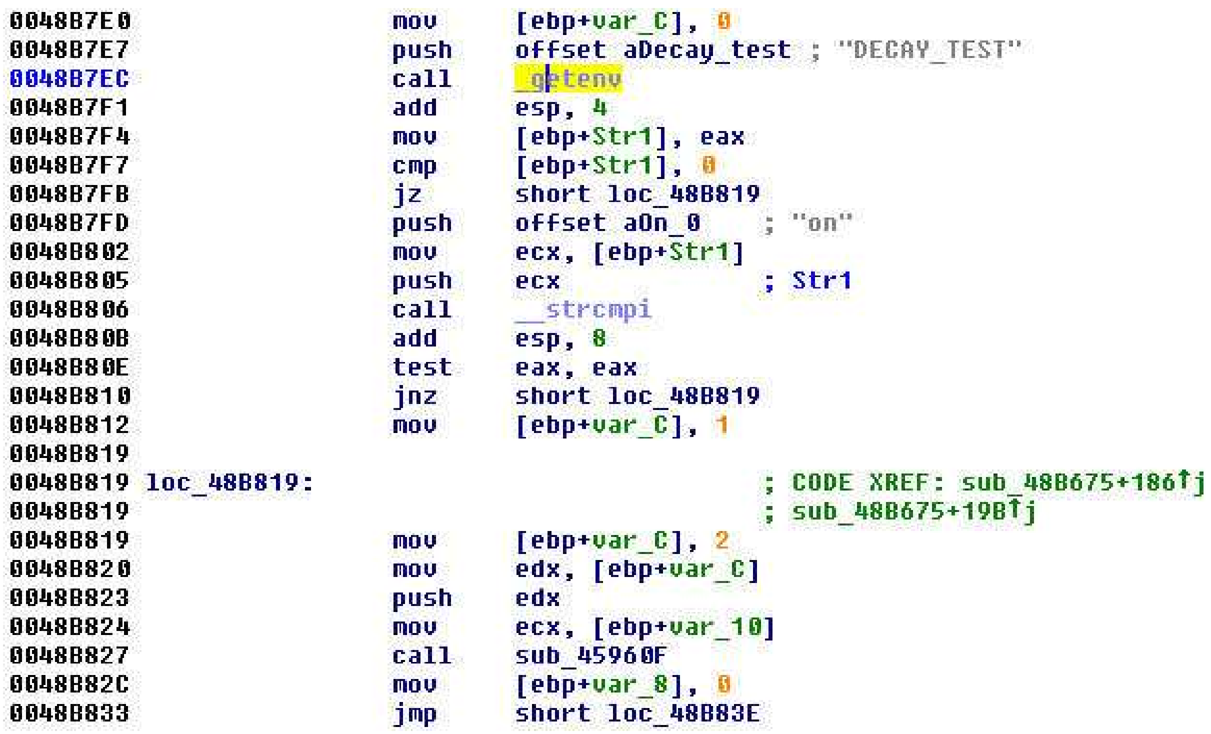
Line 1 of 5

```
                retn
_getenv         endp
```

There are 5 calls to getenv, the first one is executed when reading server.txt, and the 2 last ones are calls made by the C API and are not relevant.

We must take a look at sub_46CCA0 and sub_48B675.

DECAY_TEST



```
0048B7E0                 mov     [ebp+var_C], 0
0048B7E7                 push    offset aDecay_test ; "DECAY_TEST"
0048B7EC                 call    _getenv
0048B7F1                 add     esp, 4
0048B7F4                 mov     [ebp+Str1], eax
0048B7F7                 cmp     [ebp+Str1], 0
0048B7FB                 jz      short loc_48B819
0048B7FD                 push    offset aOn_0    ; "on"
0048B802                 mov     ecx, [ebp+Str1]
0048B805                 push    ecx             ; Str1
0048B806                 call    __strcmpi
0048B80B                 add     esp, 8
0048B80E                 test    eax, eax
0048B810                 jnz     short loc_48B819
0048B812                 mov     [ebp+var_C], 1
0048B819
0048B819 loc_48B819:                             ; CODE XREF: sub_48B675+186↑j
0048B819                                         ; sub_48B675+19B↑j
0048B819                 mov     [ebp+var_C], 2
0048B820                 mov     edx, [ebp+var_C]
0048B823                 push    edx
0048B824                 mov     ecx, [ebp+var_10]
0048B827                 call    sub_45960F
0048B82C                 mov     [ebp+var_8], 0
0048B833                 jmp     short loc_48B83E
```

By looking at the code you see that an environment variable DECAY_TEST is read, if its value is not set, then var_C will be 0. If its value is "on" (case-insensitive comparison) then var_C will be 1.

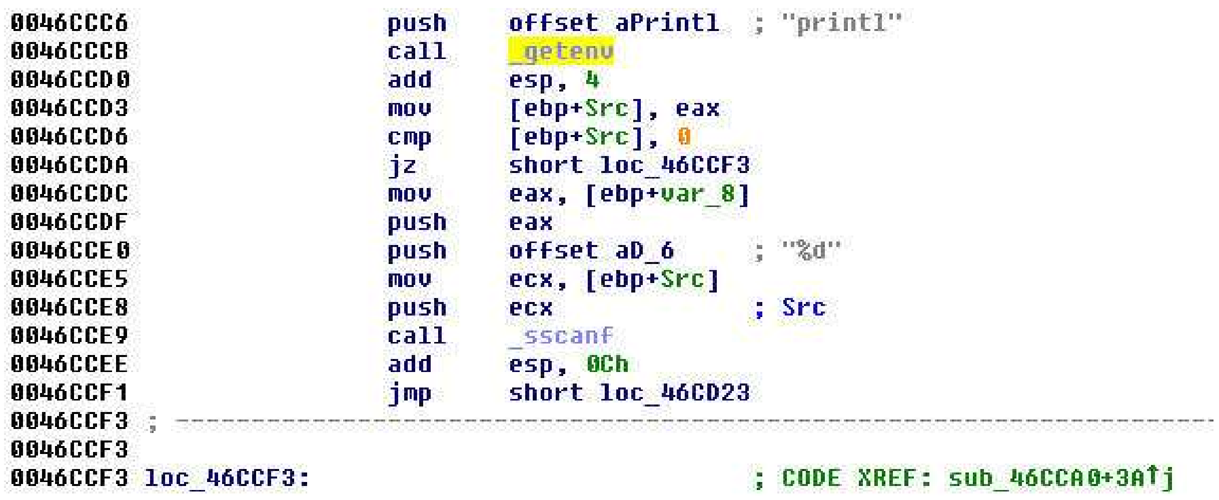
But either case, at 0x0048B819 var_C will always be set to 2!

This means that the value of DECAY_TEST is unused! Now, can we place another value in var_C and will the demo behave different? That's for you to test. I don't care at the moment.

This is C representation of the code above so you can understand the "unlogicness" better:

```
…
{
   …
  int var_C = 0;
  char *Str1 = getenv("DECAY_TEST");
  if(str1 != NULL && _strcmpi(Str1, "on") == 0)
  {
    var_C = 1;
  }
  var_C = 2;
  var_10->sub_45960F(var_c);
   …
}
```

printl



```
0046CCC6                push    offset aPrintl  ; "printl"
0046CCCB                call    _getenv
0046CCD0                add     esp, 4
0046CCD3                mov     [ebp+Src], eax
0046CCD6                cmp     [ebp+Src], 0
0046CCDA                jz      short loc_46CCF3
0046CCDC                mov     eax, [ebp+var_8]
0046CCDF                push    eax
0046CCE0                push    offset aD_6     ; "%d"
0046CCE5                mov     ecx, [ebp+Src]
0046CCE8                push    ecx             ; Src
0046CCE9                call    _sscanf
0046CCEE                add     esp, 0Ch
0046CCF1                jmp     short loc_46CD23
0046CCF3 ; ---------------------------------------------------------------------------
0046CCF3
0046CCF3 loc_46CCF3:                            ; CODE XREF: sub_46CCA0+3A↑j
```

This is another code snippet from the demo where an environment variable named printl is accessed. But I have not discovered what this is used for. From the scanf we can learn that the value must be an integer and not a string.

Mysteries

Some of the mysteries remain, especially DECAY_TEST and printl. DECAY_TEST is not really used and printl, well, analyzing its meaning will be a postponed task.