INSIDE THE ULTIMA ONLINE GOLD DEMO
- ENABLING DEBUG OUTPUT

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 does 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?

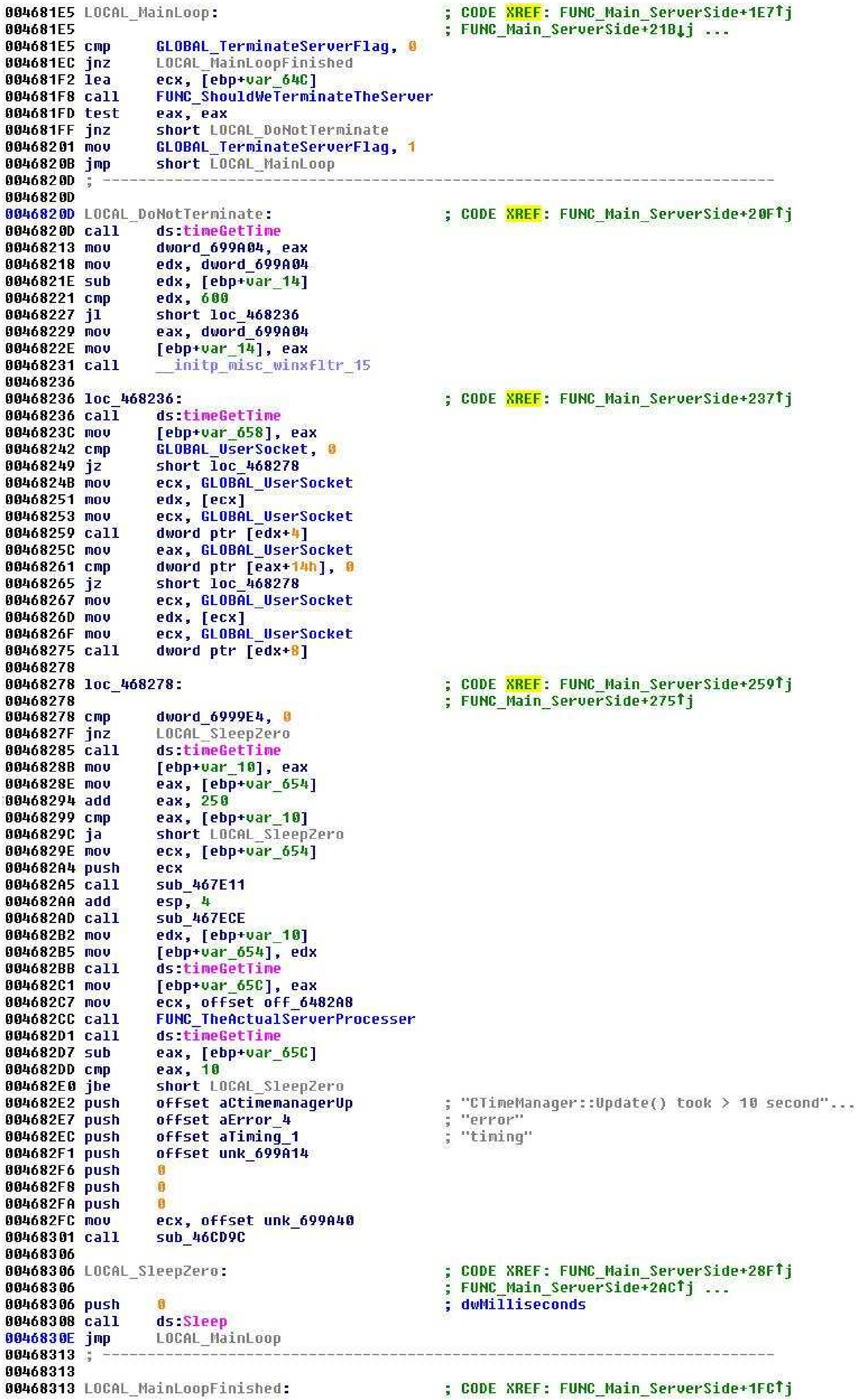
UTILITIES USED

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? ☺

THE SERVER LOOP



```
004681E5 LOCAL_MainLoop:                         ; CODE XREF: FUNC_Main_ServerSide+1E7↑j
004681E5                                         ; FUNC_Main_ServerSide+21B↓j ...
004681E5 cmp     GLOBAL_TerminateServerFlag, 0
004681EC jnz     LOCAL_MainLoopFinished
004681F2 lea     ecx, [ebp+var_64C]
004681F8 call    FUNC_ShouldWeTerminateTheServer
004681FD test    eax, eax
004681FF jnz     short LOCAL_DoNotTerminate
00468201 mov     GLOBAL_TerminateServerFlag, 1
0046820B jmp     short LOCAL_MainLoop
0046820D ; ---------------------------------------------------------------------------
0046820D
0046820D LOCAL_DoNotTerminate:                   ; CODE XREF: FUNC_Main_ServerSide+20F↑j
0046820D call    ds:timeGetTime
00468213 mov     dword_699A04, eax
00468218 mov     edx, dword_699A04
0046821E sub     edx, [ebp+var_14]
00468221 cmp     edx, 600
00468227 jl      short loc_468236
00468229 mov     eax, dword_699A04
0046822E mov     [ebp+var_14], eax
00468231 call    __initp_misc_winxfltr_15
00468236
00468236 loc_468236:                             ; CODE XREF: FUNC_Main_ServerSide+237↑j
00468236 call    ds:timeGetTime
0046823C mov     [ebp+var_658], eax
00468242 cmp     GLOBAL_UserSocket, 0
00468249 jz      short loc_468278
0046824B mov     ecx, GLOBAL_UserSocket
00468251 mov     edx, [ecx]
00468253 mov     ecx, GLOBAL_UserSocket
00468259 call    dword ptr [edx+4]
0046825C mov     eax, GLOBAL_UserSocket
00468261 cmp     dword ptr [eax+14h], 0
00468265 jz      short loc_468278
00468267 mov     ecx, GLOBAL_UserSocket
0046826D mov     edx, [ecx]
0046826F mov     ecx, GLOBAL_UserSocket
00468275 call    dword ptr [edx+8]
00468278
00468278 loc_468278:                             ; CODE XREF: FUNC_Main_ServerSide+259↑j
00468278                                         ; FUNC_Main_ServerSide+275↑j
00468278 cmp     dword_6999E4, 0
0046827F jnz     LOCAL_SleepZero
00468285 call    ds:timeGetTime
0046828B mov     [ebp+var_10], eax
0046828E mov     eax, [ebp+var_654]
00468294 add     eax, 250
00468299 cmp     eax, [ebp+var_10]
0046829C ja      short LOCAL_SleepZero
0046829E mov     ecx, [ebp+var_654]
004682A4 push    ecx
004682A5 call    sub_467E11
004682AA add     esp, 4
004682AD call    sub_467ECE
004682B2 mov     edx, [ebp+var_10]
004682B5 mov     [ebp+var_654], edx
004682BB call    ds:timeGetTime
004682C1 mov     [ebp+var_65C], eax
004682C7 mov     ecx, offset off_6482A8
004682CC call    FUNC_TheActualServerProcesser
004682D1 call    ds:timeGetTime
004682D7 sub     eax, [ebp+var_65C]
004682DD cmp     eax, 10
004682E0 jbe     short LOCAL_SleepZero
004682E2 push    offset aCtimemanagerUp          ; "CTimeManager::Update() took > 10 second"...
004682E7 push    offset aError_4                 ; "error"
004682EC push    offset aTiming_1                ; "timing"
004682F1 push    offset unk_699A14
004682F6 push    0
004682F8 push    0
004682FA push    0
004682FC mov     ecx, offset unk_699A40
00468301 call    sub_46CD9C
00468306
00468306 LOCAL_SleepZero:                        ; CODE XREF: FUNC_Main_ServerSide+28F↑j
00468306                                         ; FUNC_Main_ServerSide+2AC↑j ...
00468306 push    0                               ; dwMilliseconds
00468308 call    ds:Sleep
0046830E jmp     LOCAL_MainLoop
00468313 ; ---------------------------------------------------------------------------
00468313
00468313 LOCAL_MainLoopFinished:                 ; CODE XREF: FUNC_Main_ServerSide+1FC↑j
```

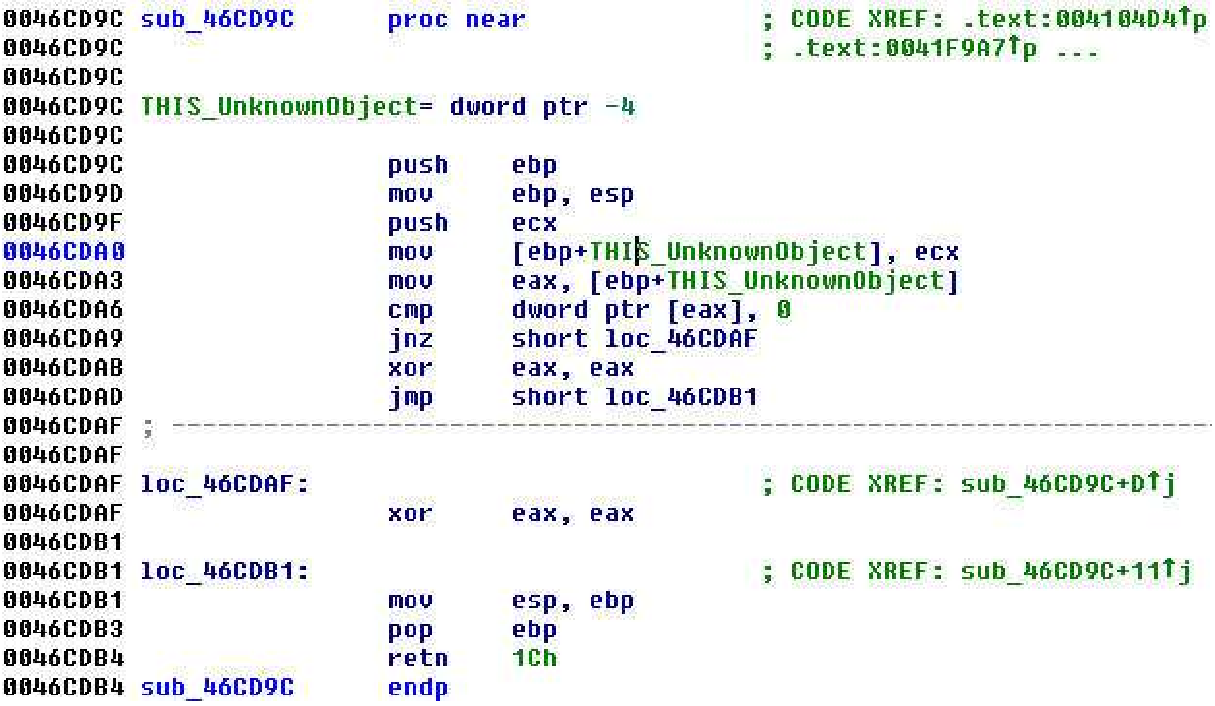
FROM SERVER LOOP TO SUB_46CD9C

The picture on the previous page showed the complete server loop, it's running in the main thread and will only stop when the client thread destroys the game window.

The loop will read packets and will send packets (if any), the actually server processing (skills, NPC's, events, and whatever is hidden deep inside) is done at FUNC_TheActualServerProcessor. This document is not about that function, haha!

Now, if this processing took more than 10 seconds, then a call is made to a function at address 0046CD9C (sub_46CD9C). The related text indicates this function logs an error.

Let's look at this function:



```
0046CD9C sub_46CD9C      proc near               ; CODE XREF: .text:004104D4↑p
0046CD9C                                         ; .text:0041F9A7↑p ...
0046CD9C
0046CD9C THIS_UnknownObject= dword ptr -4
0046CD9C
0046CD9C                 push    ebp
0046CD9D                 mov     ebp, esp
0046CD9F                 push    ecx
0046CDA0                 mov     [ebp+THIS_UnknownObject], ecx
0046CDA3                 mov     eax, [ebp+THIS_UnknownObject]
0046CDA6                 cmp     dword ptr [eax], 0
0046CDA9                 jnz     short loc_46CDAF
0046CDAB                 xor     eax, eax
0046CDAD                 jmp     short loc_46CDB1
0046CDAF ; ---------------------------------------------------------------------------
0046CDAF
0046CDAF loc_46CDAF:                             ; CODE XREF: sub_46CD9C+D↑j
0046CDAF                 xor     eax, eax
0046CDB1
0046CDB1 loc_46CDB1:                             ; CODE XREF: sub_46CD9C+11↑j
0046CDB1                 mov     esp, ebp
0046CDB3                 pop     ebp
0046CDB4                 retn    1Ch
0046CDB4 sub_46CD9C      endp
```

Now, that function is not doing anything useful! A pointer is verified against NULL, but in both cases the function return zero or false. My guess: there was an actual logging function here but it was left out during uodemo compilation.

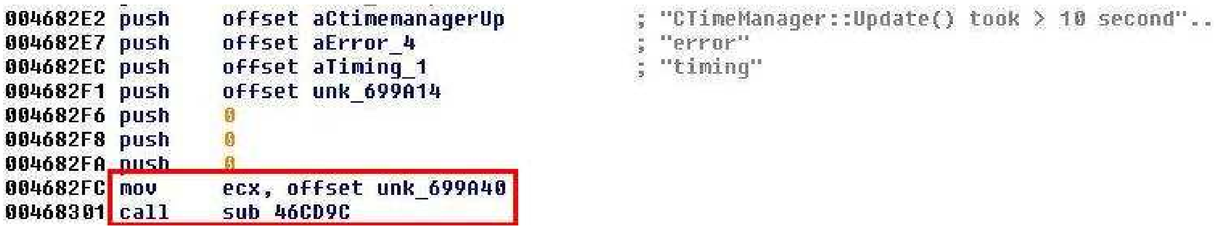
C++ version (it's a class function):

```
bool UnknownObject::sub_46CD9C()
{
  if(this->something == NULL)
    return false;
  return false;
}
```

And some guys out there say decompilation is impossible? Ah!

MORE ANALYSIS

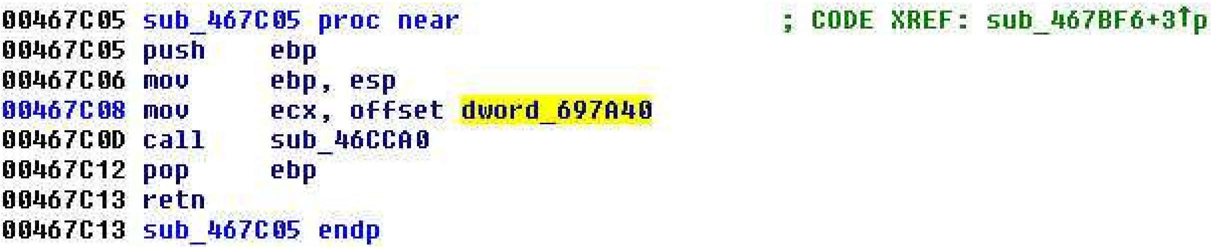
Before doing something with that function, let's look somewhat deeper.



```
004682E2 push    offset aCtimemanagerUp          ; "CTimeManager::Update() took > 10 second"..
004682E7 push    offset aError_4                 ; "error"
004682EC push    offset aTiming_1                ; "timing"
004682F1 push    offset unk_699A14
004682F6 push    0
004682F8 push    0
004682FA push    0
004682FC mov     ecx, offset unk_699A40
00468301 call    sub_46CD9C
```

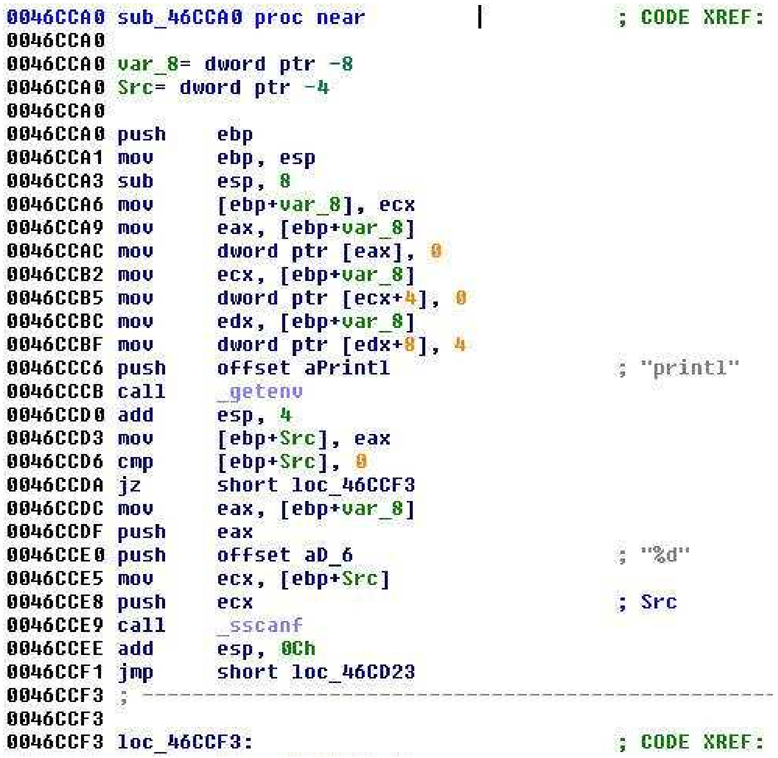
The possible logger function is operating on an object instance stored at unk_699A40.

I remember seeing that address before and I went looking for it again. I found it. Did you read my document about Environment Variables? Hint: "printl".



```
00467C05 sub_467C05 proc near                     ; CODE XREF: sub_467BF6+3↑p
00467C05 push    ebp
00467C06 mov     ebp, esp
00467C08 mov     ecx, offset dword_697A40
00467C0D call    sub_46CCA0
00467C12 pop     ebp
00467C13 retn
00467C13 sub_467C05 endp
```

Let's look at sub_46CCA0:

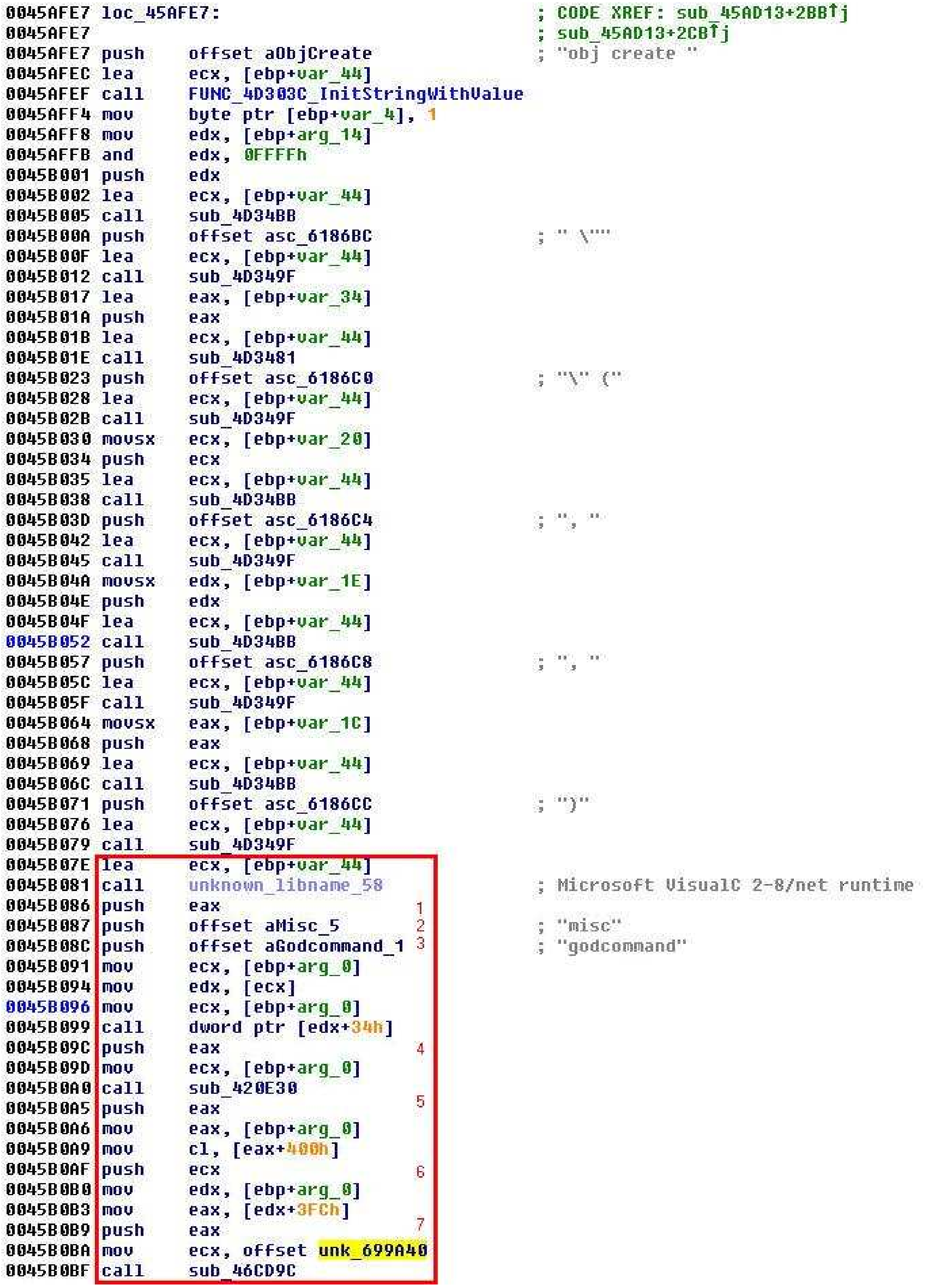


```
0046CCA0 sub_46CCA0 proc near                     ; CODE XREF:
0046CCA0
0046CCA0 var_8= dword ptr -8
0046CCA0 Src= dword ptr -4
0046CCA0
0046CCA0 push    ebp
0046CCA1 mov     ebp, esp
0046CCA3 sub     esp, 8
0046CCA6 mov     [ebp+var_8], ecx
0046CCA9 mov     eax, [ebp+var_8]
0046CCAC mov     dword ptr [eax], 0
0046CCB2 mov     ecx, [ebp+var_8]
0046CCB5 mov     dword ptr [ecx+4], 0
0046CCBC mov     edx, [ebp+var_8]
0046CCBF mov     dword ptr [edx+8], 4
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:
```

If you read that document, then you know I told you "printl" is read but the read value is never used. My guess: it's some sort value that indicates after how many lines the log file must be flushed. But since the actual log output code is not in there, this is pure speculation!

THE GOD COMMAND

By following cross references to sub_46CD9C I found out that GOD client commands are also logged on OSI. Seems logical, you want to know what your GM's are doing right?



```
0045AFE7 loc_45AFE7:                                        ; CODE XREF: sub_45AD13+2BB↑j
0045AFE7                                                    ; sub_45AD13+2CB↑j
0045AFE7 push    offset aObjCreate                          ; "obj create "
0045AFEC lea     ecx, [ebp+var_44]
0045AFEF call    FUNC_4D303C_InitStringWithValue
0045AFF4 mov     byte ptr [ebp+var_4], 1
0045AFF8 mov     edx, [ebp+arg_14]
0045AFFB and     edx, 0FFFFh
0045B001 push    edx
0045B002 lea     ecx, [ebp+var_44]
0045B005 call    sub_4D34BB
0045B00A push    offset asc_6186BC                          ; " \""
0045B00F lea     ecx, [ebp+var_44]
0045B012 call    sub_4D349F
0045B017 lea     eax, [ebp+var_34]
0045B01A push    eax
0045B01B lea     ecx, [ebp+var_44]
0045B01E call    sub_4D3481
0045B023 push    offset asc_6186C0                          ; "\" ("
0045B028 lea     ecx, [ebp+var_44]
0045B02B call    sub_4D349F
0045B030 movsx   ecx, [ebp+var_20]
0045B034 push    ecx
0045B035 lea     ecx, [ebp+var_44]
0045B038 call    sub_4D34BB
0045B03D push    offset asc_6186C4                          ; ", "
0045B042 lea     ecx, [ebp+var_44]
0045B045 call    sub_4D349F
0045B04A movsx   edx, [ebp+var_1E]
0045B04E push    edx
0045B04F lea     ecx, [ebp+var_44]
0045B052 call    sub_4D34BB
0045B057 push    offset asc_6186C8                          ; ", "
0045B05C lea     ecx, [ebp+var_44]
0045B05F call    sub_4D349F
0045B064 movsx   eax, [ebp+var_1C]
0045B068 push    eax
0045B069 lea     ecx, [ebp+var_44]
0045B06C call    sub_4D34BB
0045B071 push    offset asc_6186CC                          ; ")"
0045B076 lea     ecx, [ebp+var_44]
0045B079 call    sub_4D349F
0045B07E lea     ecx, [ebp+var_44]
0045B081 call    unknown_libname_58                         ; Microsoft VisualC 2-8/net runtime
0045B086 push    eax                        1
0045B087 push    offset aMisc_5             2               ; "misc"
0045B08C push    offset aGodcommand_1       3               ; "godcommand"
0045B091 mov     ecx, [ebp+arg_0]
0045B094 mov     edx, [ecx]
0045B096 mov     ecx, [ebp+arg_0]
0045B099 call    dword ptr [edx+34h]
0045B09C push    eax                        4
0045B09D mov     ecx, [ebp+arg_0]
0045B0A0 call    sub_420E30
0045B0A5 push    eax                        5
0045B0A6 mov     eax, [ebp+arg_0]
0045B0A9 mov     cl, [eax+400h]
0045B0AF push    ecx                        6
0045B0B0 mov     edx, [ebp+arg_0]
0045B0B3 mov     eax, [edx+3FCh]
0045B0B9 push    eax                        7
0045B0BA mov     ecx, offset unk_699A40
0045B0BF call    sub_46CD9C
```

NOTE: the log function is expecting 7 parameters (see RET 001Ch) (0x1C / 4 = 7)

OUTPUT DEBUG STRING

This log function is more than interesting and it seems that it is not only used to log errors or warnings, it also logs informational messages from the server like godcommands.

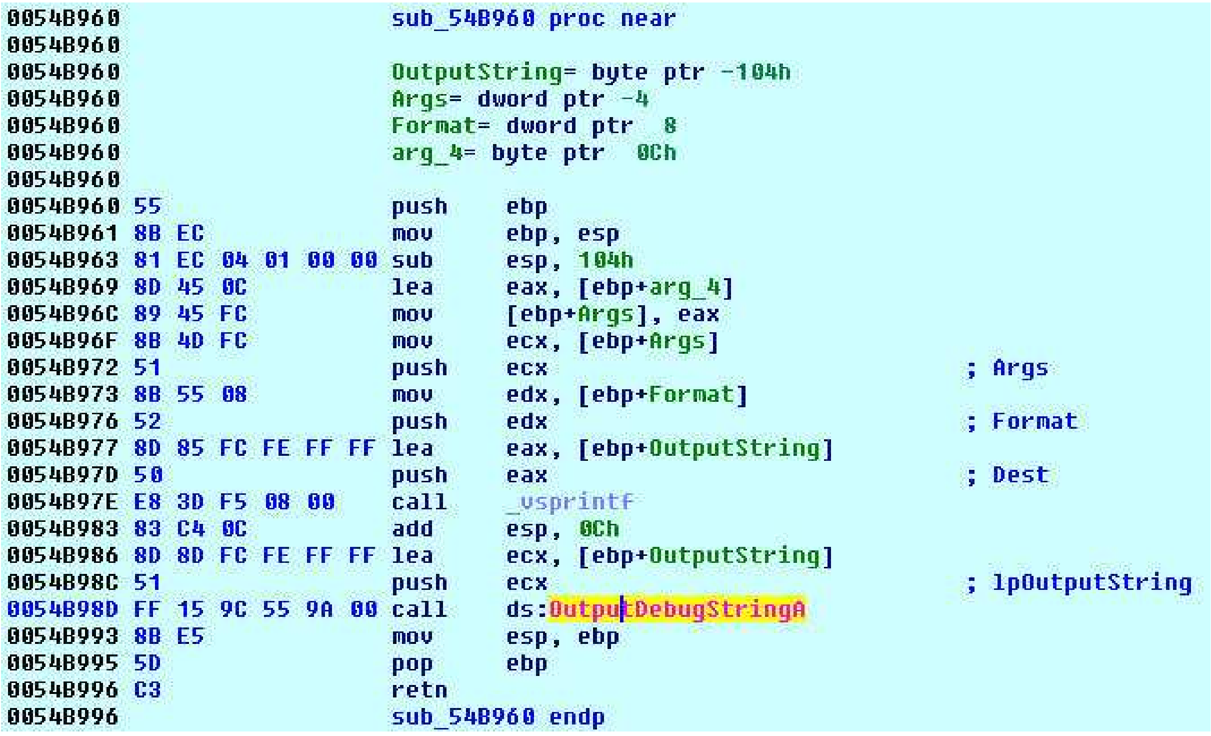
So, three options:

1) ignore this log function and ignore any log attempts
2) write the logs to file
3) write the logs to the Windows Debug Environment (or whatever is called)

I chose option 3 because it's easy to implement.

To do this, look at the import section for the OutputDebugString function. Search google if you don't know what that is. Your life depends on it!

UoDemo imports it, nice, so we don't need to add it. I looked for cross references to this Windows API function and I stumbled upon this code:



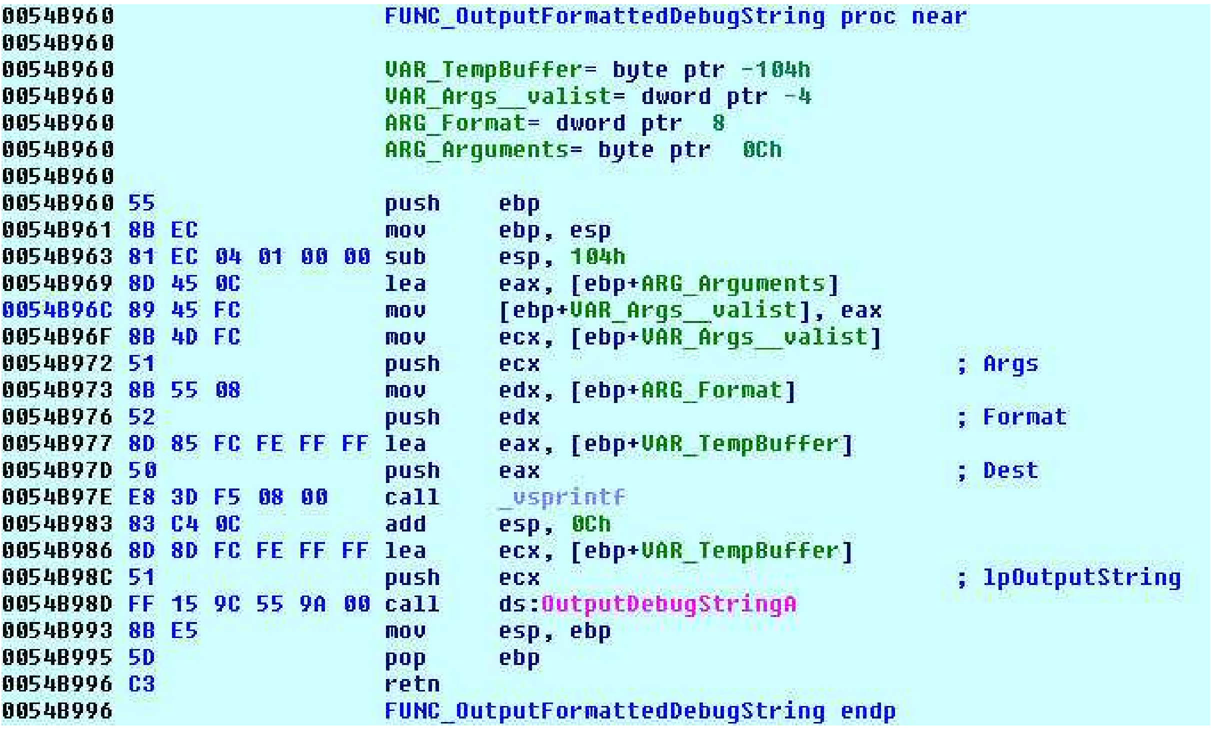
```
0054B960                      sub_54B960 proc near
0054B960
0054B960                      OutputString= byte ptr -104h
0054B960                      Args= dword ptr -4
0054B960                      Format= dword ptr  8
0054B960                      arg_4= byte ptr  0Ch
0054B960
0054B960 55                   push    ebp
0054B961 8B EC                mov     ebp, esp
0054B963 81 EC 04 01 00 00 sub     esp, 104h
0054B969 8D 45 0C             lea     eax, [ebp+arg_4]
0054B96C 89 45 FC             mov     [ebp+Args], eax
0054B96F 8B 4D FC             mov     ecx, [ebp+Args]
0054B972 51                   push    ecx                          ; Args
0054B973 8B 55 08             mov     edx, [ebp+Format]
0054B976 52                   push    edx                          ; Format
0054B977 8D 85 FC FE FF FF lea     eax, [ebp+OutputString]
0054B97D 50                   push    eax                          ; Dest
0054B97E E8 3D F5 08 00       call    _vsprintf
0054B983 83 C4 0C             add     esp, 0Ch
0054B986 8D 8D FC FE FF FF lea     ecx, [ebp+OutputString]
0054B98C 51                   push    ecx                          ; lpOutputString
0054B98D FF 15 9C 55 9A 00 call    ds:OutputDebugStringA
0054B993 8B E5                mov     esp, ebp
0054B995 5D                   pop     ebp
0054B996 C3                   retn
0054B996                      sub_54B960 endp
```

Wow! That code made my heart stop beating! I couldn't believe it would be that easy ☺. Really, "vsprintf" is in there, I hope you know your C API?

This is a basic implementation of an OutputFormattedDebugString. Note, that's how I name that function, you will probably give it a different name.

OUTPUT FORMATTED DEBUG STRING

I took some time and documented this function (which, again, is inside the uodemo):



```
0054B960                   FUNC_OutputFormattedDebugString proc near
0054B960
0054B960                   VAR_TempBuffer= byte ptr -104h
0054B960                   VAR_Args__valist= dword ptr -4
0054B960                   ARG_Format= dword ptr  8
0054B960                   ARG_Arguments= byte ptr  0Ch
0054B960
0054B960 55                push    ebp
0054B961 8B EC             mov     ebp, esp
0054B963 81 EC 04 01 00 00 sub     esp, 104h
0054B969 8D 45 0C          lea     eax, [ebp+ARG_Arguments]
0054B96C 89 45 FC          mov     [ebp+VAR_Args__valist], eax
0054B96F 8B 4D FC          mov     ecx, [ebp+VAR_Args__valist]
0054B972 51                push    ecx                             ; Args
0054B973 8B 55 08          mov     edx, [ebp+ARG_Format]
0054B976 52                push    edx                             ; Format
0054B977 8D 85 FC FE FF FF lea     eax, [ebp+VAR_TempBuffer]
0054B97D 50                push    eax                             ; Dest
0054B97E E8 3D F5 08 00    call    _vsprintf
0054B983 83 C4 0C          add     esp, 0Ch
0054B986 8D 8D FC FE FF FF lea     ecx, [ebp+VAR_TempBuffer]
0054B98C 51                push    ecx                             ; lpOutputString
0054B98D FF 15 9C 55 9A 00 call    ds:OutputDebugStringA
0054B993 8B E5             mov     esp, ebp
0054B995 5D                pop     ebp
0054B996 C3                retn
0054B996                   FUNC_OutputFormattedDebugString endp
```

This is the C version for the curious ones:

```
void FUNC_OutputFormattedDebugString(char *Format, …)
{
  char TempBuffer[260];
  va_list list;

  va_start(list, Format);
  vsprintf(buffer, Format, list);
  va_end(list);

  OutputDebugString(TempBuffer);
}
```

Basic stuff, I'm even sure you can find this function inside the MSDN examples.

The idea is to make the unused logger function call this function and have it format the 7 parameters into readable text.

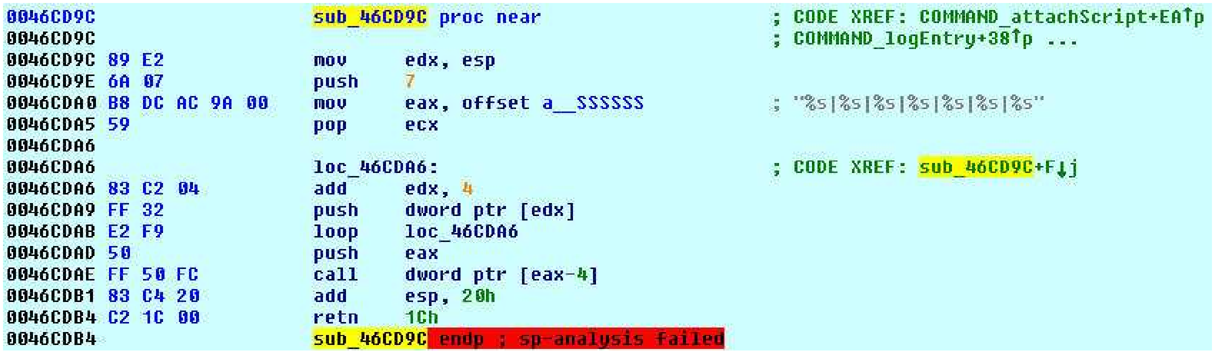
NOTE: this function can also be used to attach to the script engine and have it log during script creation or after script creation, suddenly many options are open!

ADDITIONAL NOTE: this function is probably in there (yet unreferenced) because OSI used/uses it during development

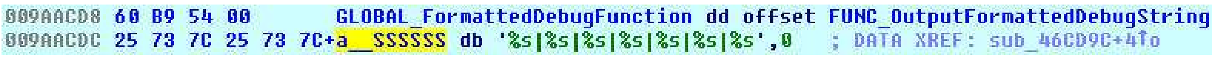
LOGFUNCTION PATCH – VERSION 1

I'm going to show you 3 different versions of my patch. Why 3? Because when you are doing something you suddenly discover a better way to attack the problem.

This was my first version:



```
0046CD9C                     sub_46CD9C proc near                   ; CODE XREF: COMMAND_attachScript+EA↑p
0046CD9C                                                            ; COMMAND_logEntry+38↑p ...
0046CD9C 89 E2               mov     edx, esp
0046CD9E 6A 07               push    7
0046CDA0 B8 DC AC 9A 00      mov     eax, offset a__SSSSSS          ; "%s|%s|%s|%s|%s|%s|%s"
0046CDA5 59                  pop     ecx
0046CDA6
0046CDA6                     loc_46CDA6:                            ; CODE XREF: sub_46CD9C+F↓j
0046CDA6 83 C2 04            add     edx, 4
0046CDA9 FF 32               push    dword ptr [edx]
0046CDAB E2 F9               loop    loc_46CDA6
0046CDAD 50                  push    eax
0046CDAE FF 50 FC            call    dword ptr [eax-4]
0046CDB1 83 C4 20            add     esp, 20h
0046CDB4 C2 1C 00            retn    1Ch
0046CDB4                     sub_46CD9C endp ; sp-analysis failed
```



```
009AACD8 60 B9 54 00       GLOBAL_FormattedDebugFunction dd offset FUNC_OutputFormattedDebugString
009AACDC 25 73 7C 25 73 7C+a__SSSSSS db '%s|%s|%s|%s|%s|%s|%s',0    ; DATA XREF: sub_46CD9C+4↑o
```

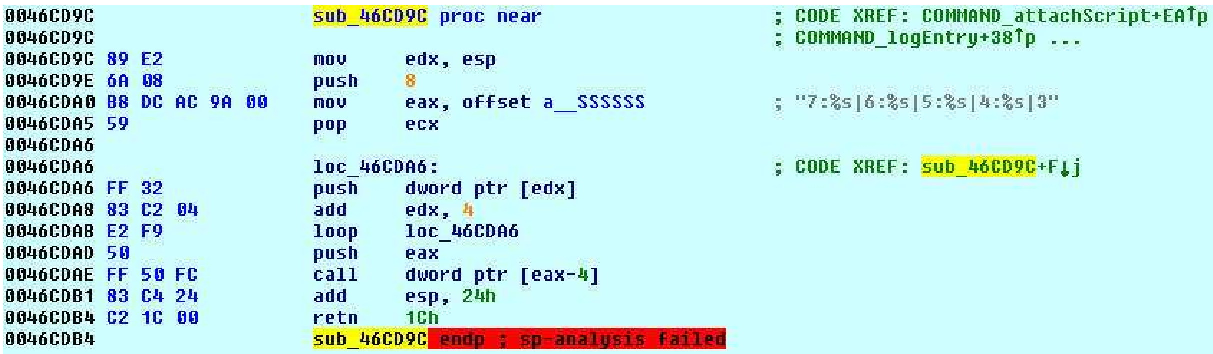
The function will push 7 values and then call the FUNC_OutputFormattedDebugString function. The function pointer is stored in memory. This is because of optimizing for size. I applied some tricks here but I wasn't very happy.

Why not? Because I realized I also wanted to log the caller EIP, so when viewing the log output you can read the IP address that the function will return to. That way you start analyzing inside the code after you noticed something interesting in the debug output.

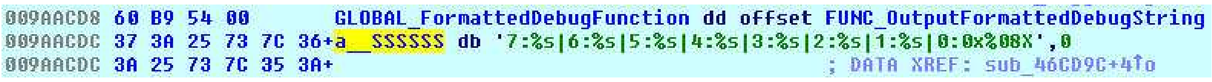
NOTE: this function pushes the parameters backwards; the first pushed value is actually the last value pushed by the caller. So when interpreting the debug output, you must do this backwards.

LOGFUNCTION PATCH – VERSION 2

This is the second version, which will log the EIP address of the caller. I also added numbers to indicate that the parameters are pushed backwards.



```
0046CD9C                      sub_46CD9C proc near                 ; CODE XREF: COMMAND_attachScript+EA↑p
0046CD9C                                                           ; COMMAND_logEntry+38↑p ...
0046CD9C 89 E2                mov     edx, esp
0046CD9E 6A 08                push    8
0046CDA0 B8 DC AC 9A 00       mov     eax, offset a__SSSSSS        ; "7:%s|6:%s|5:%s|4:%s|3"
0046CDA5 59                   pop     ecx
0046CDA6
0046CDA6                      loc_46CDA6:                          ; CODE XREF: sub_46CD9C+F↓j
0046CDA6 FF 32                push    dword ptr [edx]
0046CDA8 83 C2 04             add     edx, 4
0046CDAB E2 F9                loop    loc_46CDA6
0046CDAD 50                   push    eax
0046CDAE FF 50 FC             call    dword ptr [eax-4]
0046CDB1 83 C4 24             add     esp, 24h
0046CDB4 C2 1C 00             retn    1Ch
0046CDB4                      sub_46CD9C endp ; sp-analysis failed
```

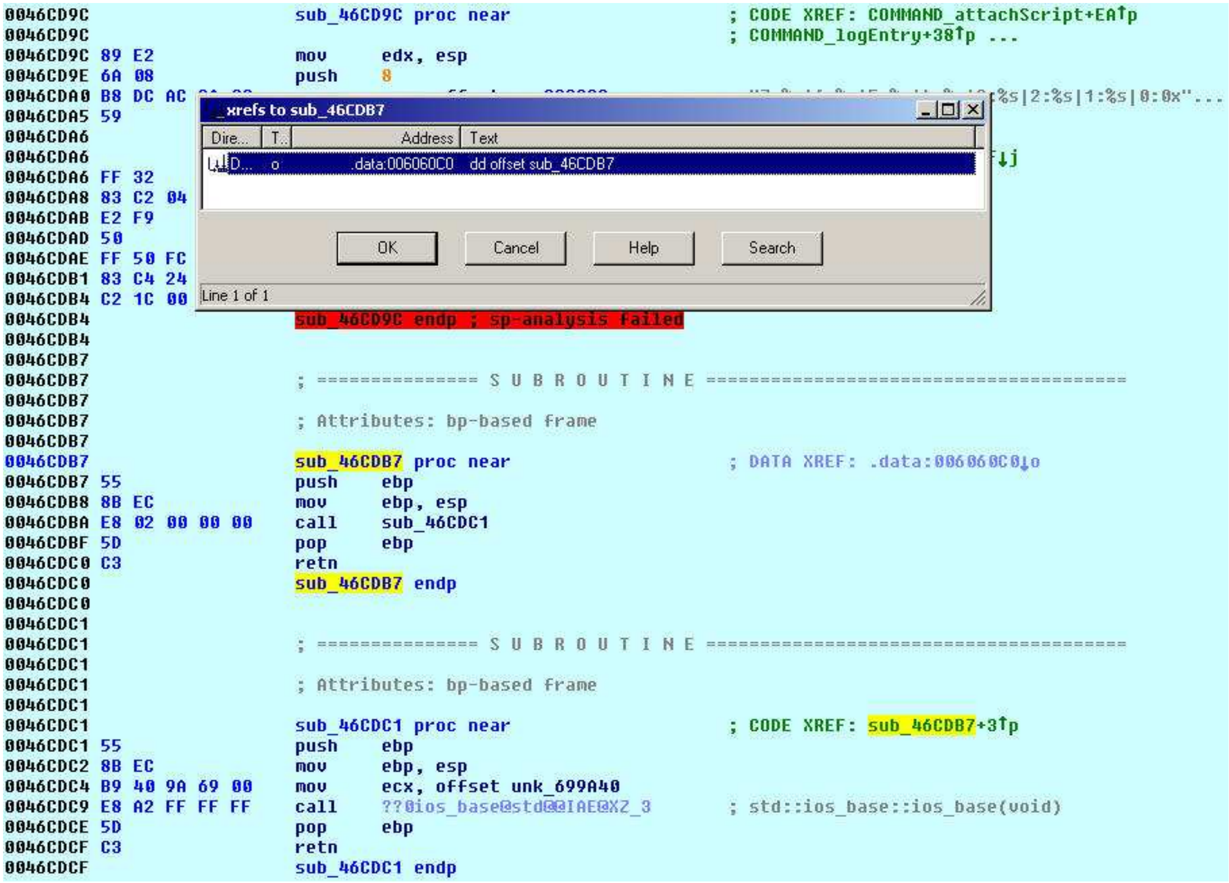


```
009AACD8 60 B9 54 00       GLOBAL_FormattedDebugFunction dd offset FUNC_OutputFormattedDebugString
009AACDC 37 3A 25 73 7C 36+a__SSSSSS db '7:%s|6:%s|5:%s|4:%s|3:%s|2:%s|1:%s|0:0x%08X',0
009AACDC 3A 25 73 7C 35 3A+                                   ; DATA XREF: sub_46CD9C+4↑o
```

But I still wasn't happy 100%. I didn't like the output, I wanted it to be in original order and I wanted the EIP address to be the first value logged.

LOGFUNCTION PATCH – VERSION 3 – WASTE OF TIME

The function following sub_46CD9C is called only once and that function only calls sub_46CDC1.



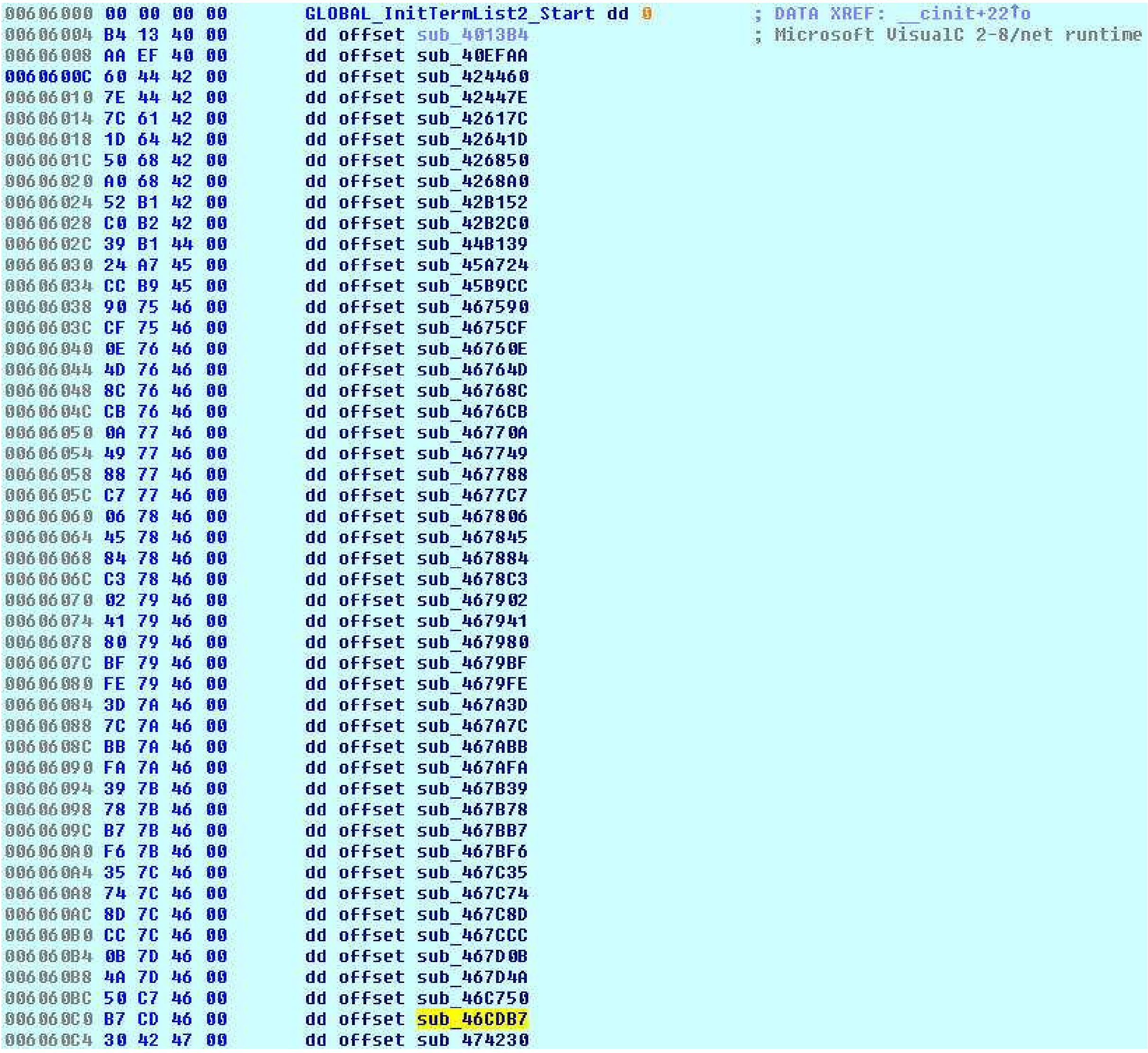
```
0046CD9C                       sub_46CD9C proc near                    ; CODE XREF: COMMAND_attachScript+EA↑p
0046CD9C                                                               ; COMMAND_logEntry+38↑p ...
0046CD9C 89 E2                 mov     edx, esp
0046CD9E 6A 08                 push    8
0046CDA0 B8 DC AC                                                                  %s|2:%s|1:%s|0:0x"...
0046CDA5 59
0046CDA6
0046CDA6                                                                         ↓j
0046CDA6 FF 32
0046CDA8 83 C2 04
0046CDAB E2 F9
0046CDAD 50
0046CDAE FF 50 FC
0046CDB1 83 C4 24
0046CDB4 C2 1C 00
0046CDB4                       sub_46CD9C endp ; sp-analysis failed
0046CDB4
0046CDB7
0046CDB7                       ; =============== S U B R O U T I N E =======================================
0046CDB7
0046CDB7                       ; Attributes: bp-based frame
0046CDB7
0046CDB7                       sub_46CDB7 proc near                    ; DATA XREF: .data:006060C0↓o
0046CDB7 55                    push    ebp
0046CDB8 8B EC                 mov     ebp, esp
0046CDBA E8 02 00 00 00        call    sub_46CDC1
0046CDBF 5D                    pop     ebp
0046CDC0 C3                    retn
0046CDC0                       sub_46CDB7 endp
0046CDC0
0046CDC1
0046CDC1                       ; =============== S U B R O U T I N E =======================================
0046CDC1
0046CDC1                       ; Attributes: bp-based frame
0046CDC1
0046CDC1                       sub_46CDC1 proc near                    ; CODE XREF: sub_46CDB7+3↑p
0046CDC1 55                    push    ebp
0046CDC2 8B EC                 mov     ebp, esp
0046CDC4 B9 40 9A 69 00        mov     ecx, offset unk_699A40
0046CDC9 E8 A2 FF FF FF        call    ??0ios_base@std@@IAE@XZ_3       ; std::ios_base::ios_base(void)
0046CDCE 5D                    pop     ebp
0046CDCF C3                    retn
0046CDCF                       sub_46CDC1 endp
```

xrefs to sub_46CDB7

Dire...	T..	Address	Text
D...	o	.data:006060C0	dd offset sub_46CDB7

Because sub_46CDB7 is called only once it means that it can be completely removed by editing or modifying only one cross reference.

That one cross reference can be found inside an initialization list, a list created by the compiler to initialize static objects and static variables and is executed by _cinit.



```
00606000 00 00 00 00     GLOBAL_InitTermList2_Start dd 0         ; DATA XREF: __cinit+22↑o
00606004 B4 13 40 00     dd offset sub_4013B4                   ; Microsoft VisualC 2-8/net runtime
00606008 AA EF 40 00     dd offset sub_40EFAA
0060600C 60 44 42 00     dd offset sub_424460
00606010 7E 44 42 00     dd offset sub_42447E
00606014 7C 61 42 00     dd offset sub_42617C
00606018 1D 64 42 00     dd offset sub_42641D
0060601C 50 68 42 00     dd offset sub_426850
00606020 A0 68 42 00     dd offset sub_4268A0
00606024 52 B1 42 00     dd offset sub_42B152
00606028 C0 B2 42 00     dd offset sub_42B2C0
0060602C 39 B1 44 00     dd offset sub_44B139
00606030 24 A7 45 00     dd offset sub_45A724
00606034 CC B9 45 00     dd offset sub_45B9CC
00606038 90 75 46 00     dd offset sub_467590
0060603C CF 75 46 00     dd offset sub_4675CF
00606040 0E 76 46 00     dd offset sub_46760E
00606044 4D 76 46 00     dd offset sub_46764D
00606048 8C 76 46 00     dd offset sub_46768C
0060604C CB 76 46 00     dd offset sub_4676CB
00606050 0A 77 46 00     dd offset sub_46770A
00606054 49 77 46 00     dd offset sub_467749
00606058 88 77 46 00     dd offset sub_467788
0060605C C7 77 46 00     dd offset sub_4677C7
00606060 06 78 46 00     dd offset sub_467806
00606064 45 78 46 00     dd offset sub_467845
00606068 84 78 46 00     dd offset sub_467884
0060606C C3 78 46 00     dd offset sub_4678C3
00606070 02 79 46 00     dd offset sub_467902
00606074 41 79 46 00     dd offset sub_467941
00606078 80 79 46 00     dd offset sub_467980
0060607C BF 79 46 00     dd offset sub_4679BF
00606080 FE 79 46 00     dd offset sub_4679FE
00606084 3D 7A 46 00     dd offset sub_467A3D
00606088 7C 7A 46 00     dd offset sub_467A7C
0060608C BB 7A 46 00     dd offset sub_467ABB
00606090 FA 7A 46 00     dd offset sub_467AFA
00606094 39 7B 46 00     dd offset sub_467B39
00606098 78 7B 46 00     dd offset sub_467B78
0060609C B7 7B 46 00     dd offset sub_467BB7
006060A0 F6 7B 46 00     dd offset sub_467BF6
006060A4 35 7C 46 00     dd offset sub_467C35
006060A8 74 7C 46 00     dd offset sub_467C74
006060AC 8D 7C 46 00     dd offset sub_467C8D
006060B0 CC 7C 46 00     dd offset sub_467CCC
006060B4 0B 7D 46 00     dd offset sub_467D0B
006060B8 4A 7D 46 00     dd offset sub_467D4A
006060BC 50 C7 46 00     dd offset sub_46C750
006060C0 B7 CD 46 00     dd offset sub_46CDB7
006060C4 30 42 47 00     dd offset sub_474230
```

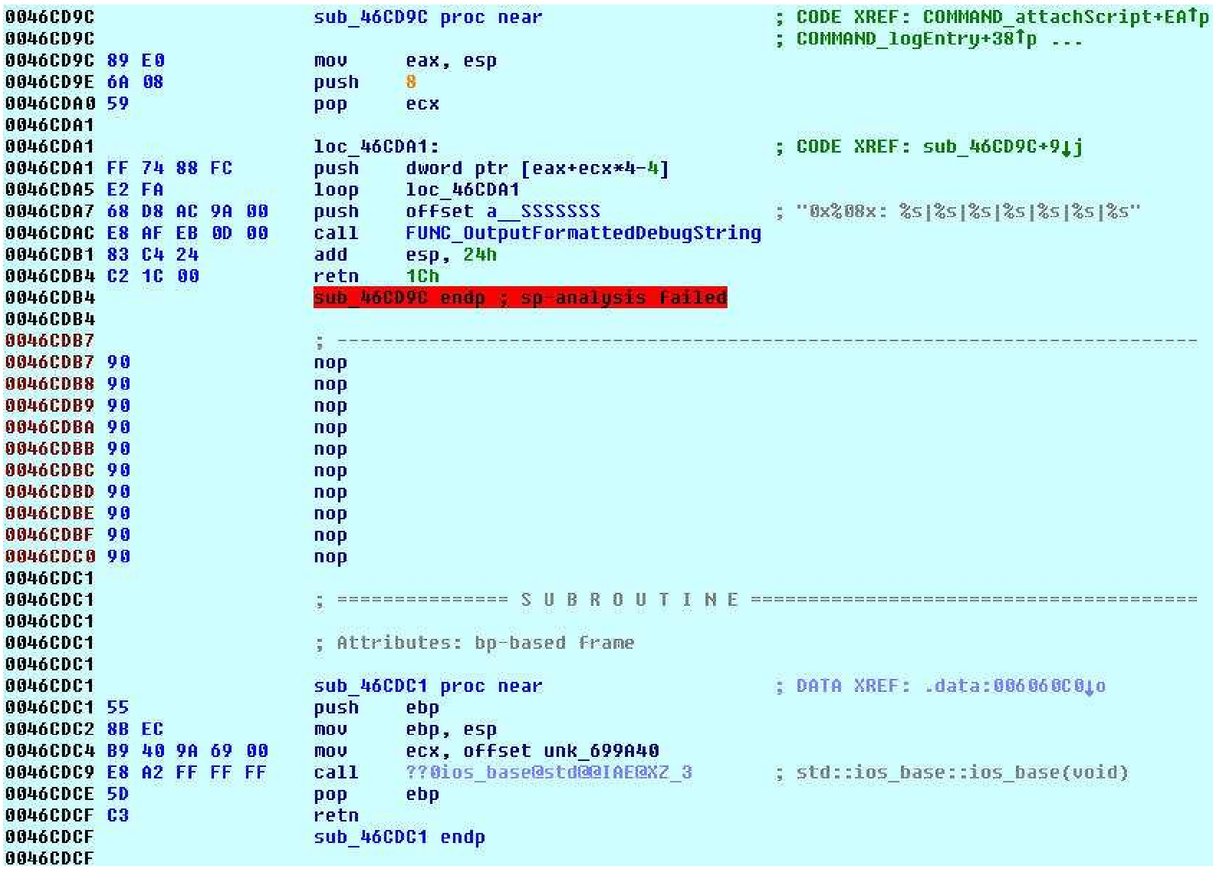
By replacing sub_46CDB7 with sub_46CDC1 you eliminate sub_46CDB7 and that code can now be overwritten with the code for the logger.

However, while coding version 3 of my patch I suddenly realized that it can be done simpler. It was there the whole time but I just didn’t see the possible optimization technique until version 3 was almost finished.

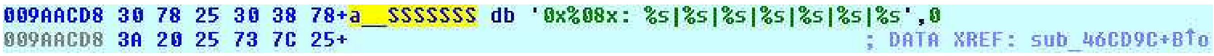
I have no picture of version 3 but instead of “add edx, 4” I used “sub edx, 4” to push in reverse thus maintaining the original push order.

LOGFUNCTION PATCH – VERSION 4

This is the final version which you can find in UoDemo+ Publish 7:



```
0046CD9C                     sub_46CD9C proc near                  ; CODE XREF: COMMAND_attachScript+EA↑p
0046CD9C                                                           ; COMMAND_logEntry+38↑p ...
0046CD9C 89 E0               mov     eax, esp
0046CD9E 6A 08               push    8
0046CDA0 59                  pop     ecx
0046CDA1
0046CDA1                     loc_46CDA1:                           ; CODE XREF: sub_46CD9C+9↓j
0046CDA1 FF 74 88 FC         push    dword ptr [eax+ecx*4-4]
0046CDA5 E2 FA               loop    loc_46CDA1
0046CDA7 68 D8 AC 9A 00      push    offset a__SSSSSSS             ; "0x%08x: %s|%s|%s|%s|%s|%s|%s"
0046CDAC E8 AF EB 0D 00      call    FUNC_OutputFormattedDebugString
0046CDB1 83 C4 24            add     esp, 24h
0046CDB4 C2 1C 00            retn    1Ch
0046CDB4                     sub_46CD9C endp ; sp-analysis failed
0046CDB4
0046CDB7                     ; ---------------------------------------------------------------------------
0046CDB7 90                  nop
0046CDB8 90                  nop
0046CDB9 90                  nop
0046CDBA 90                  nop
0046CDBB 90                  nop
0046CDBC 90                  nop
0046CDBD 90                  nop
0046CDBE 90                  nop
0046CDBF 90                  nop
0046CDC0 90                  nop
0046CDC1
0046CDC1                     ; =============== S U B R O U T I N E =======================================
0046CDC1
0046CDC1                     ; Attributes: bp-based frame
0046CDC1
0046CDC1                     sub_46CDC1 proc near                  ; DATA XREF: .data:006060C0↓o
0046CDC1 55                  push    ebp
0046CDC2 8B EC               mov     ebp, esp
0046CDC4 B9 40 9A 69 00      mov     ecx, offset unk_699A40
0046CDC9 E8 A2 FF FF FF      call    ??0ios_base@std@@IAE@XZ_3     ; std::ios_base::ios_base(void)
0046CDCE 5D                  pop     ebp
0046CDCF C3                  retn
0046CDCF                     sub_46CDC1 endp
0046CDCF
```



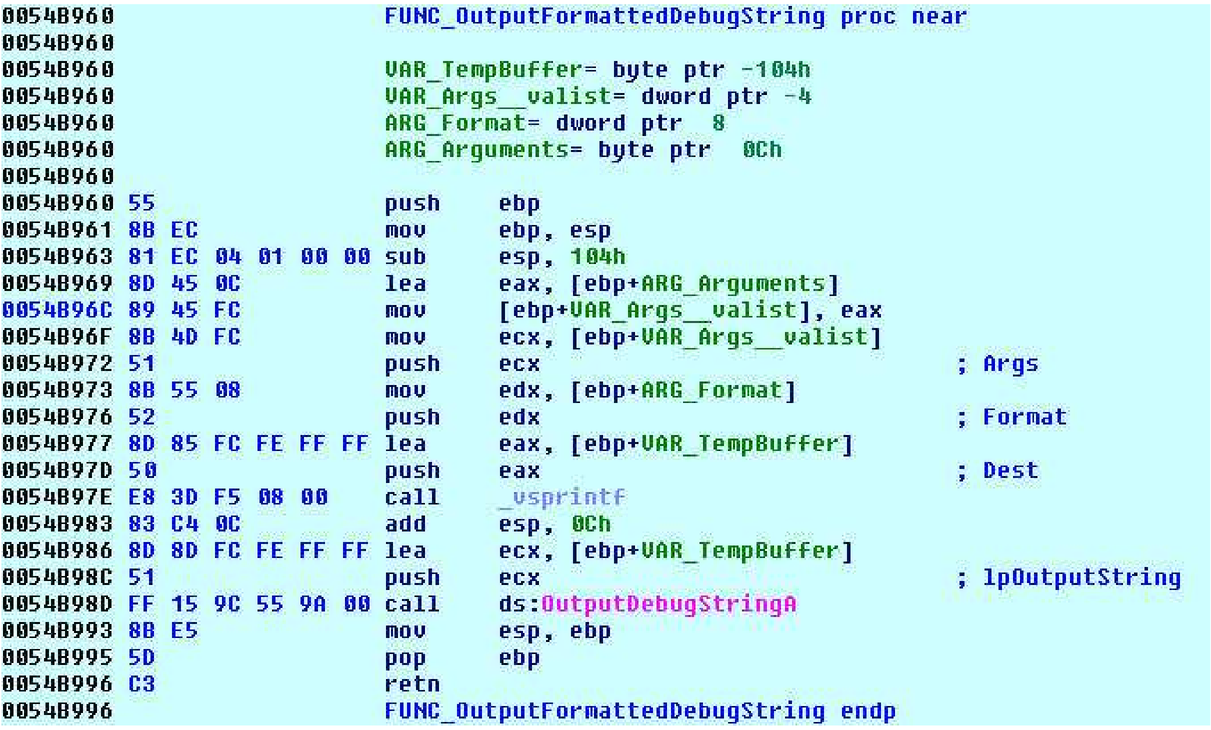
```
009AACD8 30 78 25 30 38 78+a__SSSSSSS db '0x%08x: %s|%s|%s|%s|%s|%s|%s',0
009AACD8 3A 20 25 73 7C 25+                          ; DATA XREF: sub_46CD9C+B↑o
```

Notice that I replaced function sub_46CDB7 during creation of version 3. I didn't want to put it back in (time and mood), so for now it remains patched like that. A weird reminder to the existence of version 3.

The trick is that ECX is counting down while pushing (because of the LOOP instruction), so "SUB EDX, 4" which I originally planned to use is not needed since the down counting ECX can be used for that purpose instead. It's all about optimizing for size to me; a faster version can be created with ease now since there is extra space. Are you up to that task?

INCREASING THE BUFFER FOR THE DEBUG STRING

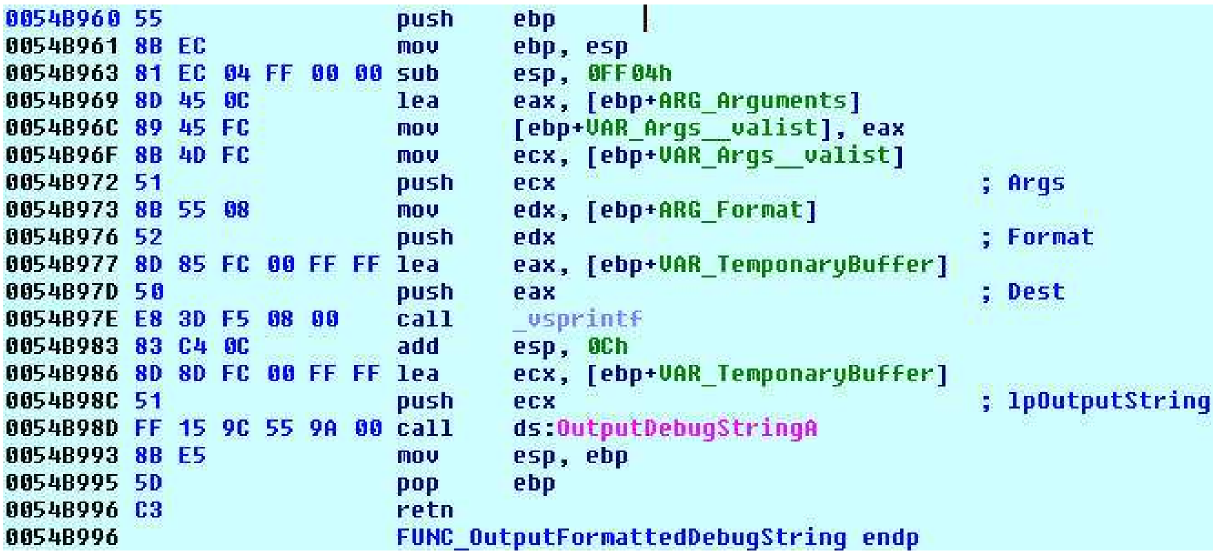
Take a look at the “output formatted debug” function again:



```
0054B960                   FUNC_OutputFormattedDebugString proc near
0054B960
0054B960                   VAR_TempBuffer= byte ptr -104h
0054B960                   VAR_Args__valist= dword ptr -4
0054B960                   ARG_Format= dword ptr  8
0054B960                   ARG_Arguments= byte ptr  0Ch
0054B960
0054B960 55                push    ebp
0054B961 8B EC             mov     ebp, esp
0054B963 81 EC 04 01 00 00 sub     esp, 104h
0054B969 8D 45 0C          lea     eax, [ebp+ARG_Arguments]
0054B96C 89 45 FC          mov     [ebp+VAR_Args__valist], eax
0054B96F 8B 4D FC          mov     ecx, [ebp+VAR_Args__valist]
0054B972 51                push    ecx                             ; Args
0054B973 8B 55 08          mov     edx, [ebp+ARG_Format]
0054B976 52                push    edx                             ; Format
0054B977 8D 85 FC FE FF FF lea     eax, [ebp+VAR_TempBuffer]
0054B97D 50                push    eax                             ; Dest
0054B97E E8 3D F5 08 00    call    _vsprintf
0054B983 83 C4 0C          add     esp, 0Ch
0054B986 8D 8D FC FE FF FF lea     ecx, [ebp+VAR_TempBuffer]
0054B98C 51                push    ecx                             ; lpOutputString
0054B98D FF 15 9C 55 9A 00 call    ds:OutputDebugStringA
0054B993 8B E5             mov     esp, ebp
0054B995 5D                pop     ebp
0054B996 C3                retn
0054B996                   FUNC_OutputFormattedDebugString endp
```

The buffer size is 0x104 bytes or 256+ 4=260 bytes or characters. That’s big, but since we have no clue what kind of logs the uodemo will create it’s better to increase this buffer size. Remember to modify the references to the VAR_TempBuffer also.

The modified function (with a really big buffer, just to be sure):



```
0054B960 55                push    ebp
0054B961 8B EC             mov     ebp, esp
0054B963 81 EC 04 FF 00 00 sub     esp, 0FF04h
0054B969 8D 45 0C          lea     eax, [ebp+ARG_Arguments]
0054B96C 89 45 FC          mov     [ebp+VAR_Args__valist], eax
0054B96F 8B 4D FC          mov     ecx, [ebp+VAR_Args__valist]
0054B972 51                push    ecx                             ; Args
0054B973 8B 55 08          mov     edx, [ebp+ARG_Format]
0054B976 52                push    edx                             ; Format
0054B977 8D 85 FC 00 FF FF lea     eax, [ebp+VAR_TemponaryBuffer]
0054B97D 50                push    eax                             ; Dest
0054B97E E8 3D F5 08 00    call    _vsprintf
0054B983 83 C4 0C          add     esp, 0Ch
0054B986 8D 8D FC 00 FF FF lea     ecx, [ebp+VAR_TemponaryBuffer]
0054B98C 51                push    ecx                             ; lpOutputString
0054B98D FF 15 9C 55 9A 00 call    ds:OutputDebugStringA
0054B993 8B E5             mov     esp, ebp
0054B995 5D                pop     ebp
0054B996 C3                retn
0054B996                   FUNC_OutputFormattedDebugString endp
```

That’s it, use this document to apply a patch yourself and view the results my friend! You’ll be amazed (I think).