INSIDE THE ULTIMA ONLINE CLIENT
- INSERTING A SLEEP

GOAL

The Ultima Online client utilizes too much CPU power when it's not doing anything useful. For example, when we are at the logon screen or when we lost connection with the server during game play.

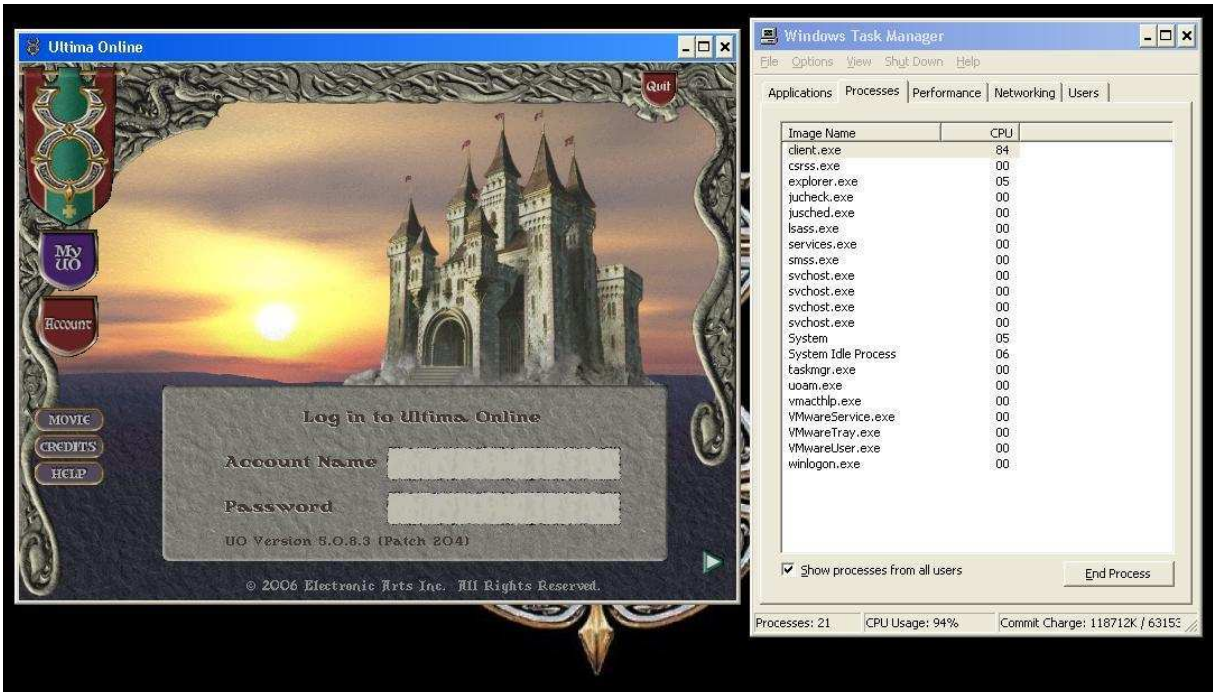

Ultima Online
Quit
My UO
Account
MOVIE
CREDITS
HELP
Log in to Ultima Online
Account Name
Password
UO Version 5.0.8.3 (Patch 204)
© 2006 Electronic Arts Inc. All Rights Reserved.
Windows Task Manager
File Options View Shut Down Help
Applications Processes Performance Networking Users
End Process

Image Name	CPU
client.exe	84
csrss.exe	00
explorer.exe	05
jucheck.exe	00
jusched.exe	00
lsass.exe	00
services.exe	00
smss.exe	00
svchost.exe	00
svchost.exe	00
svchost.exe	00
svchost.exe	00
System	05
System Idle Process	06
taskmgr.exe	00
uoam.exe	00
vmacthlp.exe	00
VMwareService.exe	00
VMwareTray.exe	00
VMwareUser.exe	00
winlogon.exe	00

☑ Show processes from all users

Processes: 21 CPU Usage: 94% Commit Charge: 118712K / 63153

In this document I will describe how I made a patch for the client and hopefully, you learn how to patch your own client when I'm not there to do it for you.

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 computers starting from age 12, and Ultima VII was the first game I bought myself, don't ask how I acquired games before that. Oh yeah, I learned GFA Basic at age 13, switched to Borland C++ 2.0 at age 14, and assembler came to me at age 15, and that's when it all started for real.

INSIDE THE CLIENT

There are many different clients out there, remember, client has a minimum 10 year old history. Each binary is different but in the end will share some code with the original one. Compilers also evolved and newer clients will utilize more and modern optimizations techniques.

I chose to patch client version 5.0.8.3. Load it into IDA and read on.

Locating the message loop shouldn't be too hard. UO Client is written in C++ thus the message loop will look like this:

```
While(GetMessage(…))
{
   TranslateMessage(…);
   DispatchMessage(…);
}
```

or

GetMessage(…) can be replaced by PeekMessage(…) which is more common for games anyways.

TEACH YOURSELF BY READING MORE ABOUT MESSAGE LOOPS

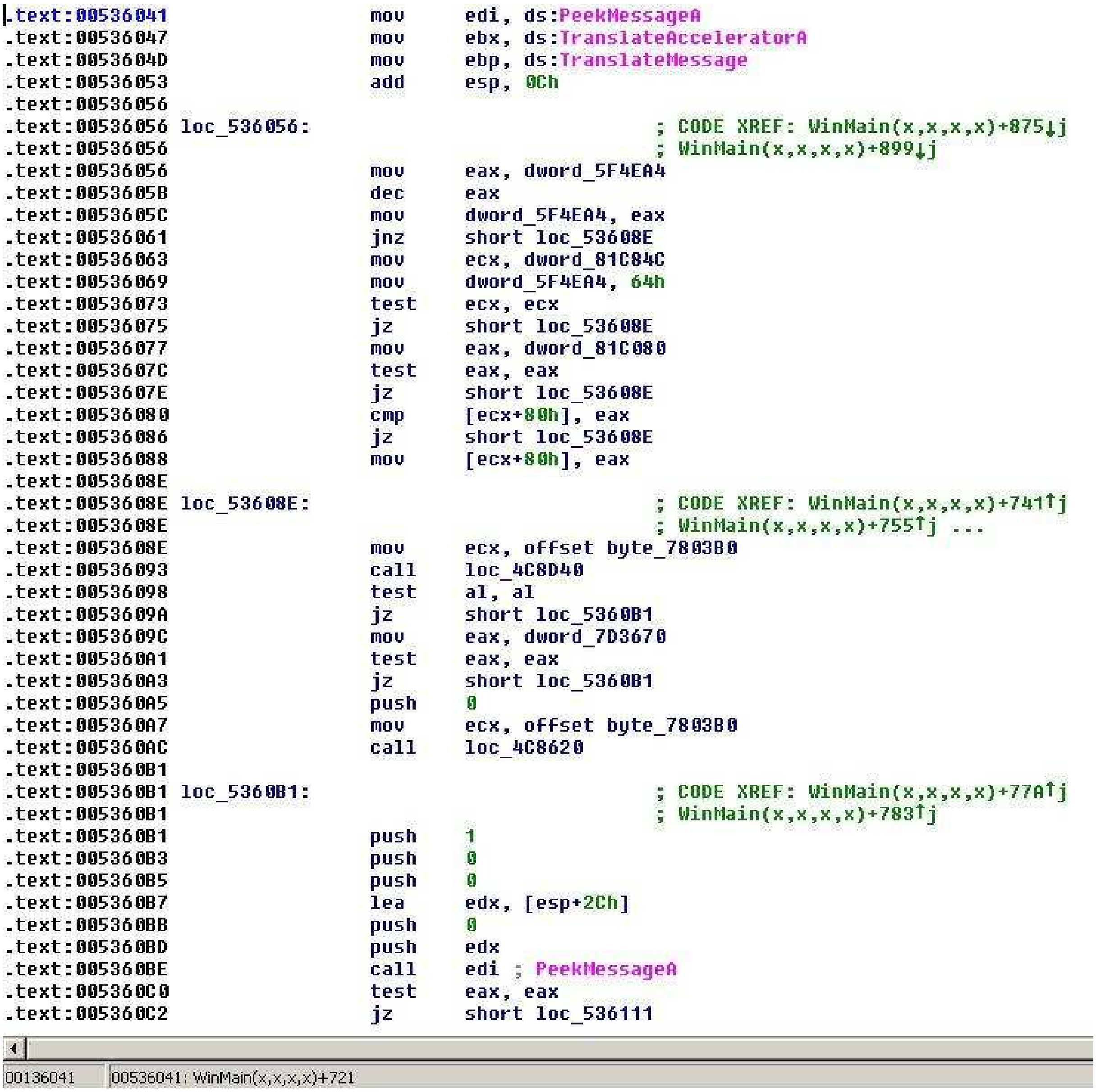
http://winprog.org/tutorial/message_loop.html
http://blogs.msdn.com/oldnewthing/archive/2005/02/09/369804.aspx
http://software.intel.com/en-us/articles/peekmessage-optimizing-applications-for-extended-battery-life/

MAKING SENSE OF THE MESSAGE LOOP

Now that we have located the message loop we have to analyze it, we do this by adding comments and/or renaming the labels IDA created for us. Feel free to use the built-in debugger of IDA to help with understanding the structures and variables involved.

This is not really a tutorial about IDA Pro or about reversing in general; hence I'm not going to provide you with too much details on how to do things.

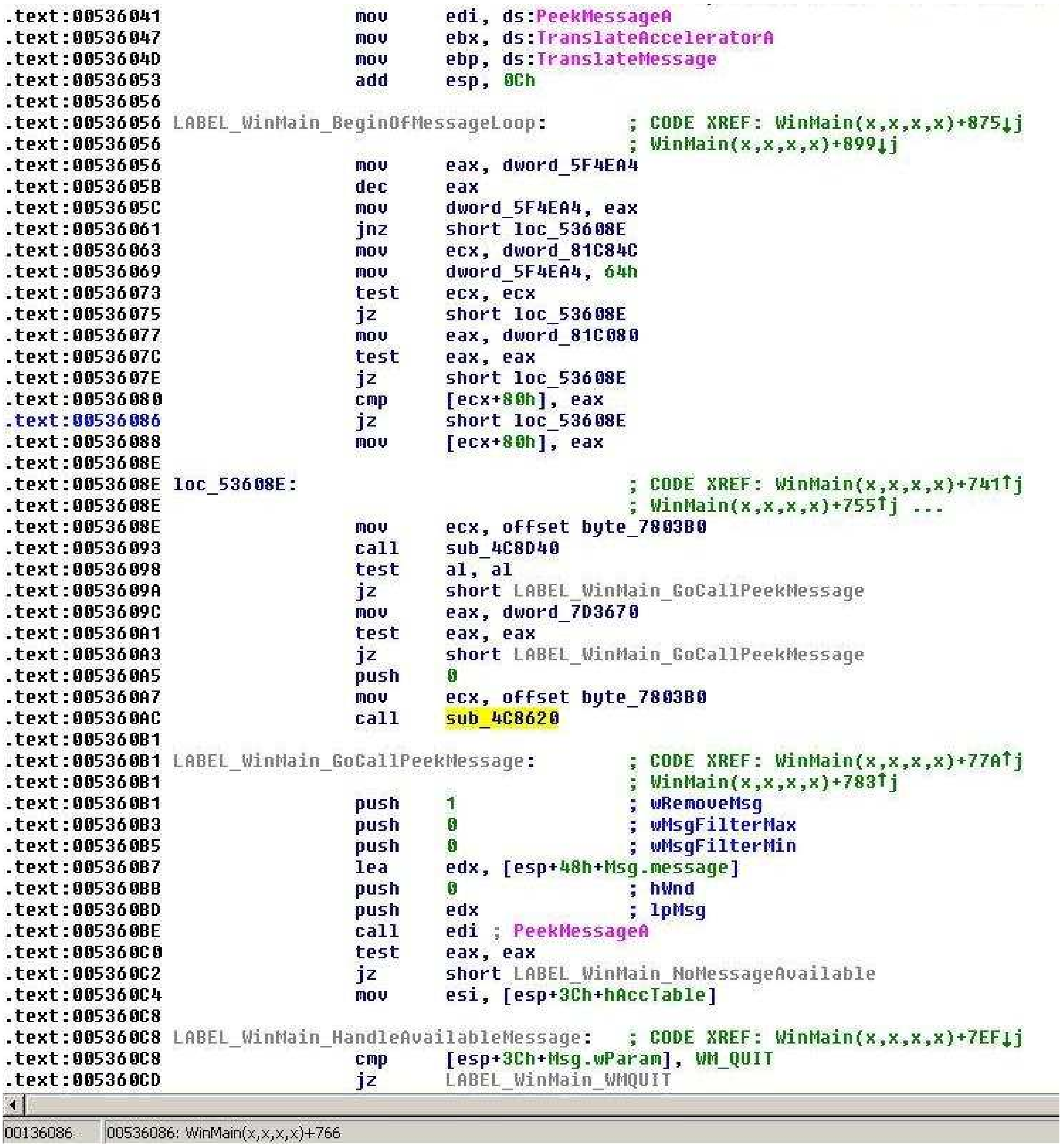
Before analysis, a screenshot of what is the beginning of the message loop:



```
.text:00536041                 mov     edi, ds:PeekMessageA
.text:00536047                 mov     ebx, ds:TranslateAcceleratorA
.text:0053604D                 mov     ebp, ds:TranslateMessage
.text:00536053                 add     esp, 0Ch
.text:00536056
.text:00536056 loc_536056:                              ; CODE XREF: WinMain(x,x,x,x)+875↓j
.text:00536056                                          ; WinMain(x,x,x,x)+899↓j
.text:00536056                 mov     eax, dword_5F4EA4
.text:0053605B                 dec     eax
.text:0053605C                 mov     dword_5F4EA4, eax
.text:00536061                 jnz     short loc_53608E
.text:00536063                 mov     ecx, dword_81C84C
.text:00536069                 mov     dword_5F4EA4, 64h
.text:00536073                 test    ecx, ecx
.text:00536075                 jz      short loc_53608E
.text:00536077                 mov     eax, dword_81C080
.text:0053607C                 test    eax, eax
.text:0053607E                 jz      short loc_53608E
.text:00536080                 cmp     [ecx+80h], eax
.text:00536086                 jz      short loc_53608E
.text:00536088                 mov     [ecx+80h], eax
.text:0053608E
.text:0053608E loc_53608E:                              ; CODE XREF: WinMain(x,x,x,x)+741↑j
.text:0053608E                                          ; WinMain(x,x,x,x)+755↑j ...
.text:0053608E                 mov     ecx, offset byte_7803B0
.text:00536093                 call    loc_4C8D40
.text:00536098                 test    al, al
.text:0053609A                 jz      short loc_5360B1
.text:0053609C                 mov     eax, dword_7D3670
.text:005360A1                 test    eax, eax
.text:005360A3                 jz      short loc_5360B1
.text:005360A5                 push    0
.text:005360A7                 mov     ecx, offset byte_7803B0
.text:005360AC                 call    loc_4C8620
.text:005360B1
.text:005360B1 loc_5360B1:                              ; CODE XREF: WinMain(x,x,x,x)+77A↑j
.text:005360B1                                          ; WinMain(x,x,x,x)+783↑j
.text:005360B1                 push    1
.text:005360B3                 push    0
.text:005360B5                 push    0
.text:005360B7                 lea     edx, [esp+2Ch]
.text:005360BB                 push    0
.text:005360BD                 push    edx
.text:005360BE                 call    edi ; PeekMessageA
.text:005360C0                 test    eax, eax
.text:005360C2                 jz      short loc_536111
```

00136041 | 00536041: WinMain(x,x,x,x)+721

After analysis, we have a screenshot of this same part of the Message Loop:

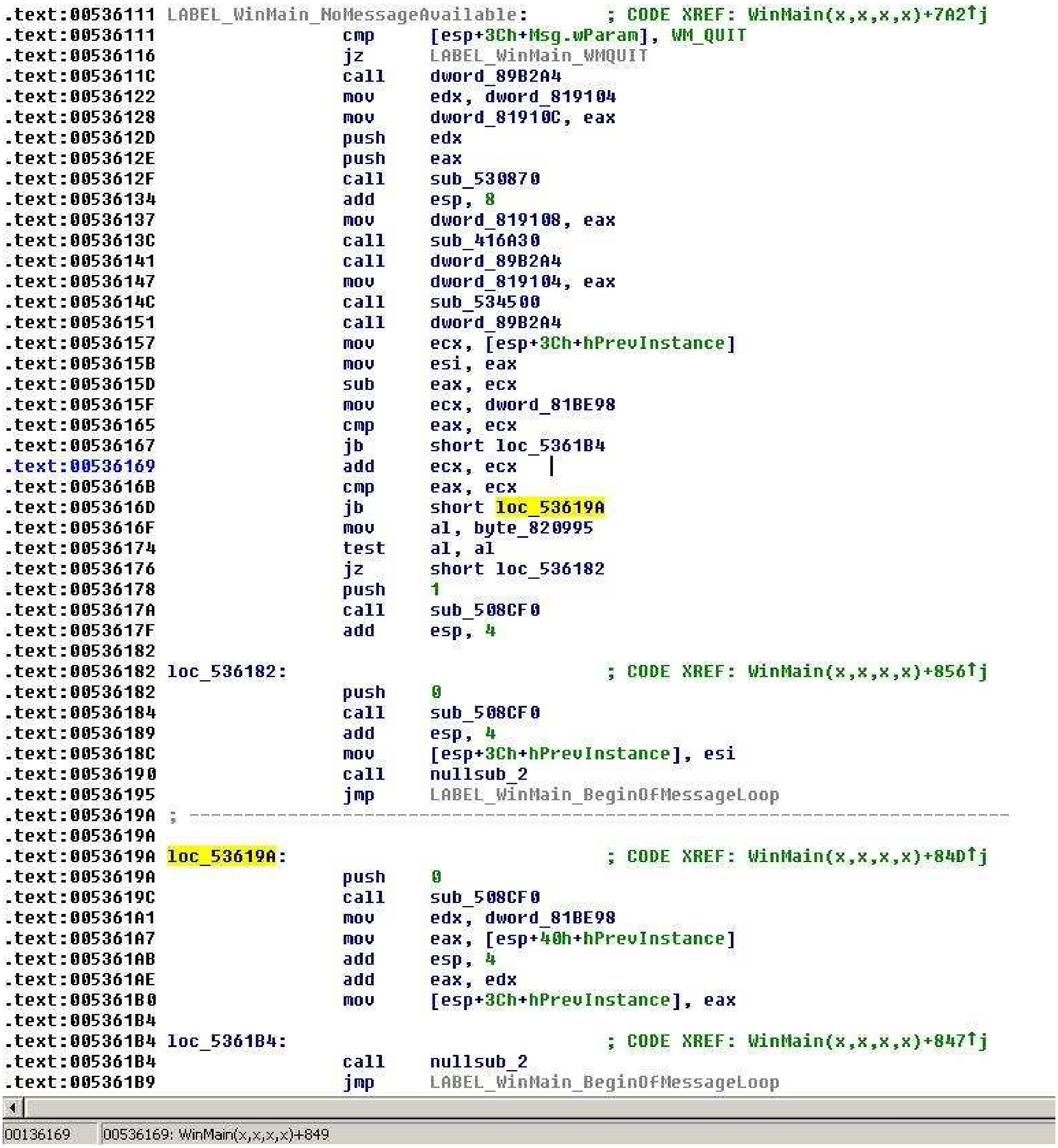


```
.text:00536041                  mov     edi, ds:PeekMessageA
.text:00536047                  mov     ebx, ds:TranslateAcceleratorA
.text:0053604D                  mov     ebp, ds:TranslateMessage
.text:00536053                  add     esp, 0Ch
.text:00536056
.text:00536056 LABEL_WinMain_BeginOfMessageLoop:       ; CODE XREF: WinMain(x,x,x,x)+875↓j
.text:00536056                                         ; WinMain(x,x,x,x)+899↓j
.text:00536056                  mov     eax, dword_5F4EA4
.text:0053605B                  dec     eax
.text:0053605C                  mov     dword_5F4EA4, eax
.text:00536061                  jnz     short loc_53608E
.text:00536063                  mov     ecx, dword_81C84C
.text:00536069                  mov     dword_5F4EA4, 64h
.text:00536073                  test    ecx, ecx
.text:00536075                  jz      short loc_53608E
.text:00536077                  mov     eax, dword_81C080
.text:0053607C                  test    eax, eax
.text:0053607E                  jz      short loc_53608E
.text:00536080                  cmp     [ecx+80h], eax
.text:00536086                  jz      short loc_53608E
.text:00536088                  mov     [ecx+80h], eax
.text:0053608E
.text:0053608E loc_53608E:                             ; CODE XREF: WinMain(x,x,x,x)+741↑j
.text:0053608E                                         ; WinMain(x,x,x,x)+755↑j ...
.text:0053608E                  mov     ecx, offset byte_7803B0
.text:00536093                  call    sub_4C8D40
.text:00536098                  test    al, al
.text:0053609A                  jz      short LABEL_WinMain_GoCallPeekMessage
.text:0053609C                  mov     eax, dword_7D3670
.text:005360A1                  test    eax, eax
.text:005360A3                  jz      short LABEL_WinMain_GoCallPeekMessage
.text:005360A5                  push    0
.text:005360A7                  mov     ecx, offset byte_7803B0
.text:005360AC                  call    sub_4C8620
.text:005360B1
.text:005360B1 LABEL_WinMain_GoCallPeekMessage:        ; CODE XREF: WinMain(x,x,x,x)+77A↑j
.text:005360B1                                         ; WinMain(x,x,x,x)+783↑j
.text:005360B1                  push    1               ; wRemoveMsg
.text:005360B3                  push    0               ; wMsgFilterMax
.text:005360B5                  push    0               ; wMsgFilterMin
.text:005360B7                  lea     edx, [esp+48h+Msg.message]
.text:005360BB                  push    0               ; hWnd
.text:005360BD                  push    edx             ; lpMsg
.text:005360BE                  call    edi ; PeekMessageA
.text:005360C0                  test    eax, eax
.text:005360C2                  jz      short LABEL_WinMain_NoMessageAvailable
.text:005360C4                  mov     esi, [esp+3Ch+hAccTable]
.text:005360C8
.text:005360C8 LABEL_WinMain_HandleAvailableMessage:   ; CODE XREF: WinMain(x,x,x,x)+7EF↓j
.text:005360C8                  cmp     [esp+3Ch+Msg.wParam], WM_QUIT
.text:005360CD                  jz      LABEL_WinMain_WMQUIT
```

00136086 00536086: WinMain(x,x,x,x)+766

Between the beginning of the loop and the actually PeekMessage call at 005360BE, the client is doing stuff. Further analysis will be required to know what exactly is going on.

Next, here is a screenshot of the code part WinMain_NoMessageAvailable which is executed, as the label says, in case there is no message available. ☺

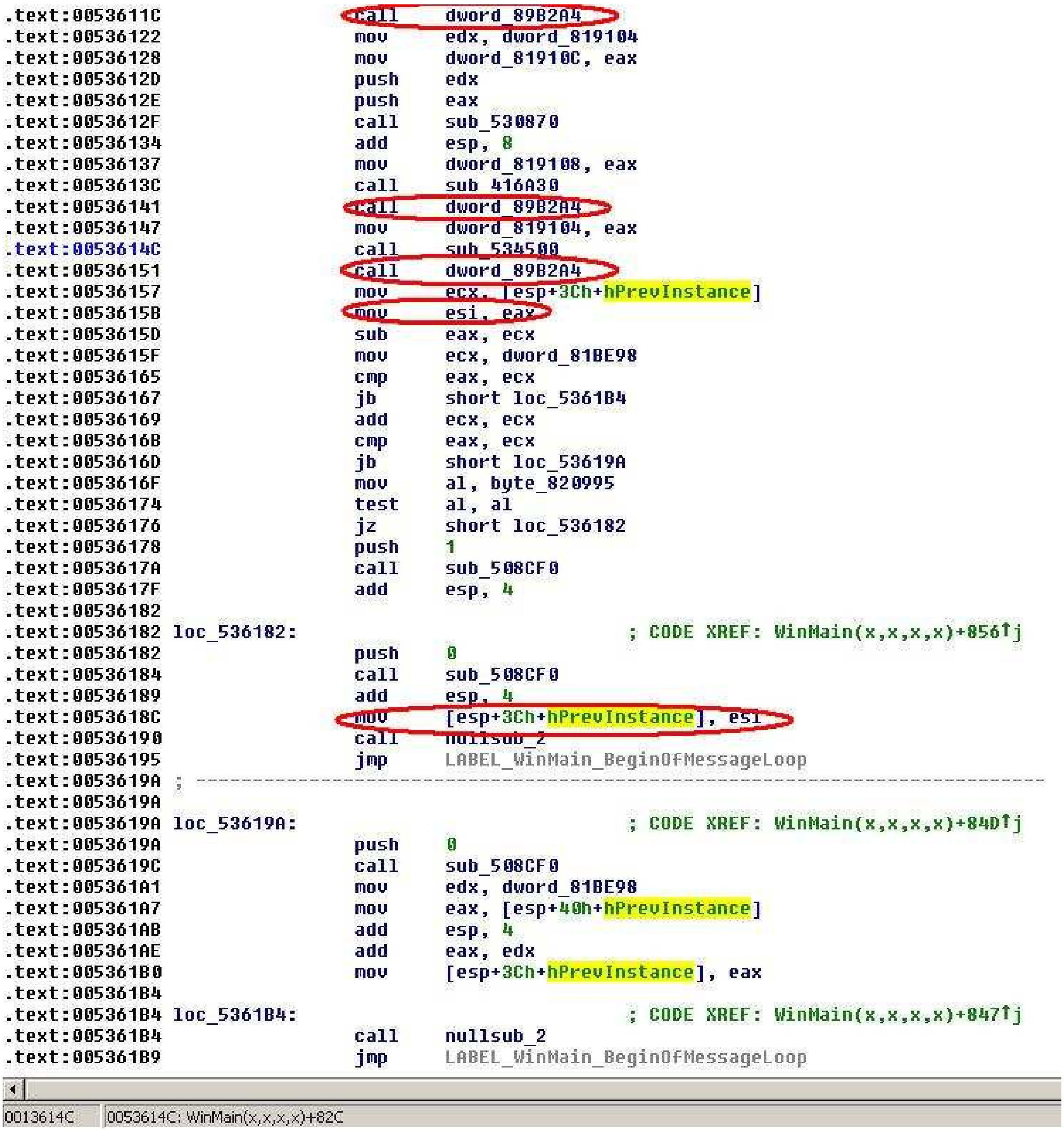


```
.text:00536111 LABEL_WinMain_NoMessageAvailable:       ; CODE XREF: WinMain(x,x,x,x)+7A2↑j
.text:00536111                 cmp     [esp+3Ch+Msg.wParam], WM_QUIT
.text:00536116                 jz      LABEL_WinMain_WMQUIT
.text:0053611C                 call    dword_89B2A4
.text:00536122                 mov     edx, dword_819104
.text:00536128                 mov     dword_81910C, eax
.text:0053612D                 push    edx
.text:0053612E                 push    eax
.text:0053612F                 call    sub_530870
.text:00536134                 add     esp, 8
.text:00536137                 mov     dword_819108, eax
.text:0053613C                 call    sub_416A30
.text:00536141                 call    dword_89B2A4
.text:00536147                 mov     dword_819104, eax
.text:0053614C                 call    sub_534500
.text:00536151                 call    dword_89B2A4
.text:00536157                 mov     ecx, [esp+3Ch+hPrevInstance]
.text:0053615B                 mov     esi, eax
.text:0053615D                 sub     eax, ecx
.text:0053615F                 mov     ecx, dword_81BE98
.text:00536165                 cmp     eax, ecx
.text:00536167                 jb      short loc_5361B4
.text:00536169                 add     ecx, ecx
.text:0053616B                 cmp     eax, ecx
.text:0053616D                 jb      short loc_53619A
.text:0053616F                 mov     al, byte_820995
.text:00536174                 test    al, al
.text:00536176                 jz      short loc_536182
.text:00536178                 push    1
.text:0053617A                 call    sub_508CF0
.text:0053617F                 add     esp, 4
.text:00536182
.text:00536182 loc_536182:                             ; CODE XREF: WinMain(x,x,x,x)+856↑j
.text:00536182                 push    0
.text:00536184                 call    sub_508CF0
.text:00536189                 add     esp, 4
.text:0053618C                 mov     [esp+3Ch+hPrevInstance], esi
.text:00536190                 call    nullsub_2
.text:00536195                 jmp     LABEL_WinMain_BeginOfMessageLoop
.text:0053619A ; ---------------------------------------------------------------------------
.text:0053619A
.text:0053619A loc_53619A:                             ; CODE XREF: WinMain(x,x,x,x)+84D↑j
.text:0053619A                 push    0
.text:0053619C                 call    sub_508CF0
.text:005361A1                 mov     edx, dword_81BE98
.text:005361A7                 mov     eax, [esp+40h+hPrevInstance]
.text:005361AB                 add     esp, 4
.text:005361AE                 add     eax, edx
.text:005361B0                 mov     [esp+3Ch+hPrevInstance], eax
.text:005361B4
.text:005361B4 loc_5361B4:                             ; CODE XREF: WinMain(x,x,x,x)+847↑j
.text:005361B4                 call    nullsub_2
.text:005361B9                 jmp     LABEL_WinMain_BeginOfMessageLoop
```

Notice that before execution is resumed at LABEL_WinMain_BeginOfMessageLoop 3 different code paths can be executed. Also, notice the usage of hPrevInstance! That parameter of WinMain has not been in use since the release of Windows 95. It's Windows 3.1 and earlier stuff! This means that the UO Client is using the parameter for something else. At startup (under 95 and up) it is guaranteed to be 0.

Further reading: http://blogs.msdn.com/oldnewthing/archive/2004/06/15/156022.aspx

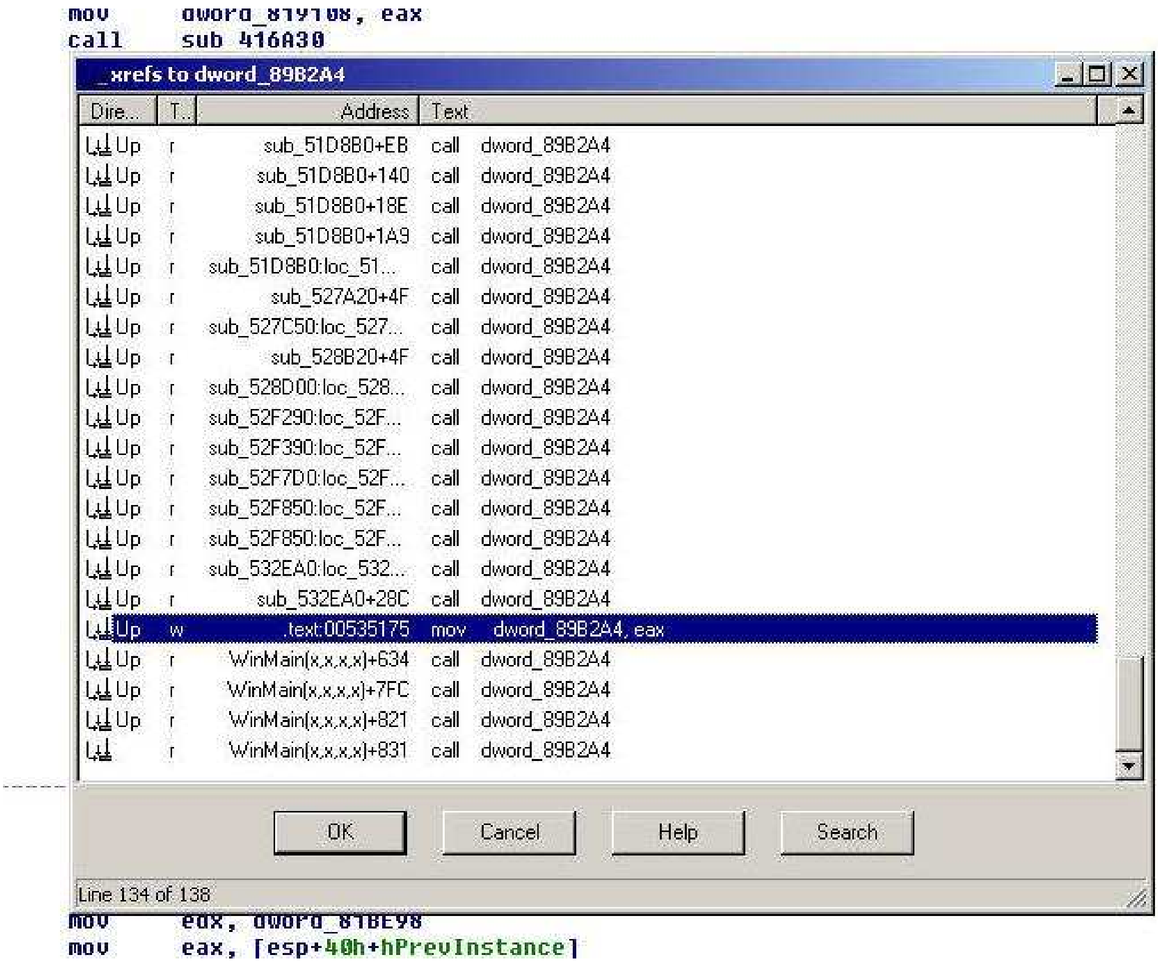
We will further analyze this hPrevInstance thing, look at 0053618C, ESI is stored in hPrevInstance, but where is ESI coming from? Look at 0053615B, the value of EAX is put in ESI right after a function call stored in dword_89B2A4. Also, the function stored in dword_89B2A4 is called more than once! Its meaning must be significant.



```
.text:0053611C                 call    dword_89B2A4
.text:00536122                 mov     edx, dword_819104
.text:00536128                 mov     dword_81910C, eax
.text:0053612D                 push    edx
.text:0053612E                 push    eax
.text:0053612F                 call    sub_530870
.text:00536134                 add     esp, 8
.text:00536137                 mov     dword_819108, eax
.text:0053613C                 call    sub_416A30
.text:00536141                 call    dword_89B2A4
.text:00536147                 mov     dword_819104, eax
.text:0053614C                 call    sub_534500
.text:00536151                 call    dword_89B2A4
.text:00536157                 mov     ecx, [esp+3Ch+hPrevInstance]
.text:0053615B                 mov     esi, eax
.text:0053615D                 sub     eax, ecx
.text:0053615F                 mov     ecx, dword_81BE98
.text:00536165                 cmp     eax, ecx
.text:00536167                 jb      short loc_5361B4
.text:00536169                 add     ecx, ecx
.text:0053616B                 cmp     eax, ecx
.text:0053616D                 jb      short loc_53619A
.text:0053616F                 mov     al, byte_820995
.text:00536174                 test    al, al
.text:00536176                 jz      short loc_536182
.text:00536178                 push    1
.text:0053617A                 call    sub_508CF0
.text:0053617F                 add     esp, 4
.text:00536182
.text:00536182 loc_536182:                             ; CODE XREF: WinMain(x,x,x,x)+856↑j
.text:00536182                 push    0
.text:00536184                 call    sub_508CF0
.text:00536189                 add     esp, 4
.text:0053618C                 mov     [esp+3Ch+hPrevInstance], esi
.text:00536190                 call    nullsub_2
.text:00536195                 jmp     LABEL_WinMain_BeginOfMessageLoop
.text:0053619A ; ---------------------------------------------------------------------------
.text:0053619A
.text:0053619A loc_53619A:                             ; CODE XREF: WinMain(x,x,x,x)+84D↑j
.text:0053619A                 push    0
.text:0053619C                 call    sub_508CF0
.text:005361A1                 mov     edx, dword_81BE98
.text:005361A7                 mov     eax, [esp+40h+hPrevInstance]
.text:005361AB                 add     esp, 4
.text:005361AE                 add     eax, edx
.text:005361B0                 mov     [esp+3Ch+hPrevInstance], eax
.text:005361B4
.text:005361B4 loc_5361B4:                             ; CODE XREF: WinMain(x,x,x,x)+847↑j
.text:005361B4                 call    nullsub_2
.text:005361B9                 jmp     LABEL_WinMain_BeginOfMessageLoop
```

0013614C | 0053614C: WinMain(x,x,x,x)+82C

Picture of actual references to dword_89B2A4:



```
mov     dword_819108, eax
call    sub 416A30
```

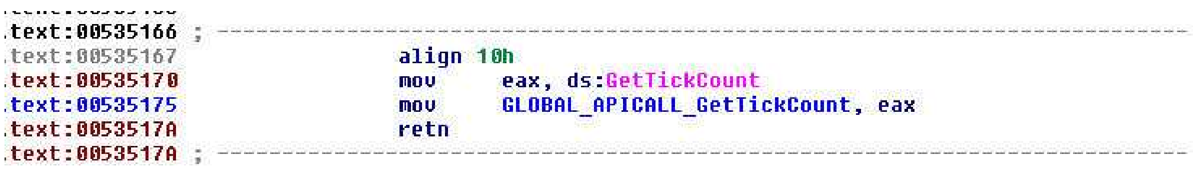
_xrefs to dword_89B2A4

Dire...	T...	Address	Text
Up	r	sub_51D8B0+EB	call dword_89B2A4
Up	r	sub_51D8B0+140	call dword_89B2A4
Up	r	sub_51D8B0+18E	call dword_89B2A4
Up	r	sub_51D8B0+1A9	call dword_89B2A4
Up	r	sub_51D8B0:loc_51...	call dword_89B2A4
Up	r	sub_527A20+4F	call dword_89B2A4
Up	r	sub_527C50:loc_527...	call dword_89B2A4
Up	r	sub_528B20+4F	call dword_89B2A4
Up	r	sub_528D00:loc_528...	call dword_89B2A4
Up	r	sub_52F290:loc_52F...	call dword_89B2A4
Up	r	sub_52F390:loc_52F...	call dword_89B2A4
Up	r	sub_52F7D0:loc_52F...	call dword_89B2A4
Up	r	sub_52F850:loc_52F...	call dword_89B2A4
Up	r	sub_52F850:loc_52F...	call dword_89B2A4
Up	r	sub_532EA0:loc_532...	call dword_89B2A4
Up	r	sub_532EA0+28C	call dword_89B2A4
Up	w	.text:00535175	mov dword_89B2A4, eax
Up	r	WinMain(x,x,x,x)+634	call dword_89B2A4
Up	r	WinMain(x,x,x,x)+7FC	call dword_89B2A4
Up	r	WinMain(x,x,x,x)+821	call dword_89B2A4
	r	WinMain(x,x,x,x)+831	call dword_89B2A4

Line 134 of 138

```
mov     edx, dword_81BE98
mov     eax, [esp+40h+hPrevInstance]
```

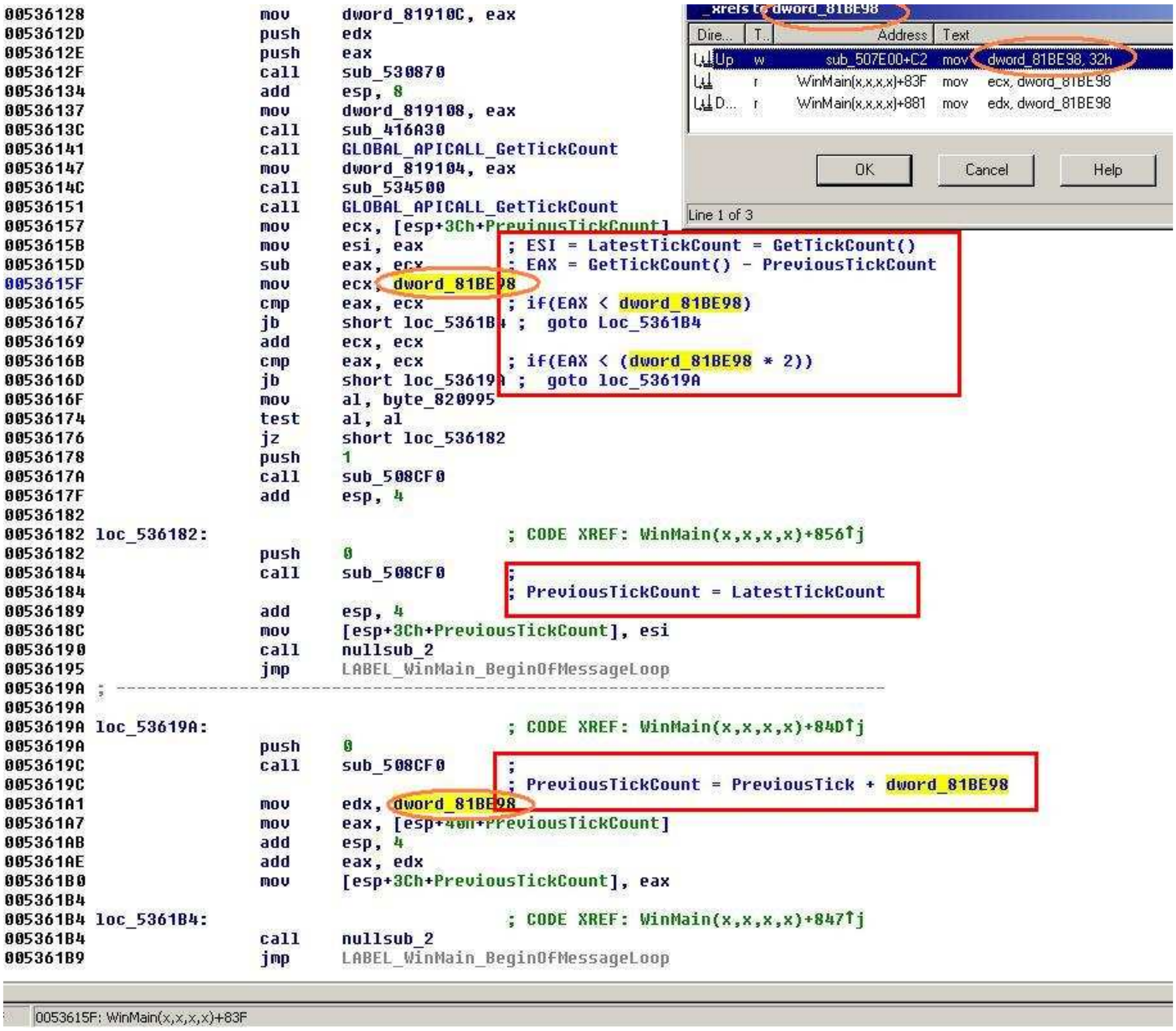
The variable is modified only once at 00535175. Let's take a look there:



```
.text:00535166 ; ---------------------------------------------------------------------------
.text:00535167                 align 10h
.text:00535170                 mov     eax, ds:GetTickCount
.text:00535175                 mov     GLOBAL_APICALL_GetTickCount, eax
.text:0053517A                 retn
.text:0053517A ; ---------------------------------------------------------------------------
```

It's actually something basic: GetTickCount. I renamed dword_89B2A4 to GLOBAL_APICALL_GetTickCount, because that's exactly what it is doing. Calling dword_89B2A4 will call GetTickCount. Maybe OSI once thought about implementing different techniques for time keeping but so far only GetTickCount seem to have been used.

This gives us a better picture of what is going inside the client when PeekMessage returns zero:



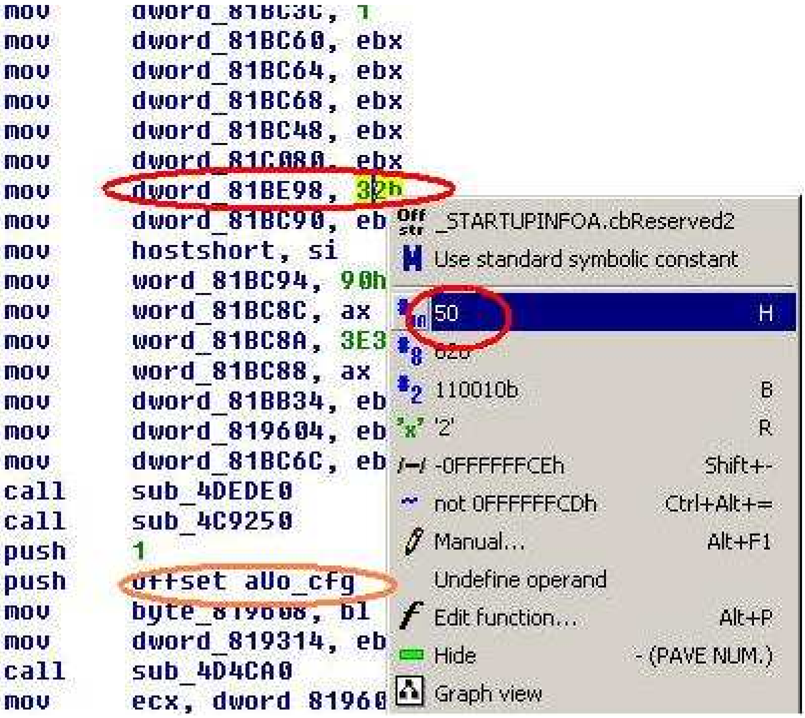
```
_xrefs to dword_81BE98
Dire... T.. Address                Text
Up      w   sub_507E00+C2          mov  dword_81BE98, 32h
        r   WinMain(x,x,x,x)+83F   mov  ecx, dword_81BE98
D...    r   WinMain(x,x,x,x)+881   mov  edx, dword_81BE98
Line 1 of 3
```

```
00536128                 mov     dword_81910C, eax
0053612D                 push    edx
0053612E                 push    eax
0053612F                 call    sub_530870
00536134                 add     esp, 8
00536137                 mov     dword_819108, eax
0053613C                 call    sub_416A30
00536141                 call    GLOBAL_APICALL_GetTickCount
00536147                 mov     dword_819104, eax
0053614C                 call    sub_534500
00536151                 call    GLOBAL_APICALL_GetTickCount
00536157                 mov     ecx, [esp+3Ch+PreviousTickCount]
0053615B                 mov     esi, eax             ; ESI = LatestTickCount = GetTickCount()
0053615D                 sub     eax, ecx             ; EAX = GetTickCount() - PreviousTickCount
0053615F                 mov     ecx, dword_81BE98
00536165                 cmp     eax, ecx             ; if(EAX < dword_81BE98)
00536167                 jb      short loc_5361B4 ;   goto Loc_5361B4
00536169                 add     ecx, ecx
0053616B                 cmp     eax, ecx             ; if(EAX < (dword_81BE98 * 2))
0053616D                 jb      short loc_53619A ;   goto loc_53619A
0053616F                 mov     al, byte_820995
00536174                 test    al, al
00536176                 jz      short loc_536182
00536178                 push    1
0053617A                 call    sub_508CF0
0053617F                 add     esp, 4
00536182
00536182 loc_536182:                                  ; CODE XREF: WinMain(x,x,x,x)+856↑j
00536182                 push    0
00536184                 call    sub_508CF0           ;
00536184                                              ; PreviousTickCount = LatestTickCount
00536189                 add     esp, 4
0053618C                 mov     [esp+3Ch+PreviousTickCount], esi
00536190                 call    nullsub_2
00536195                 jmp     LABEL_WinMain_BeginOfMessageLoop
0053619A ; ---------------------------------------------------------------------------
0053619A
0053619A loc_53619A:                                  ; CODE XREF: WinMain(x,x,x,x)+84D↑j
0053619A                 push    0
0053619C                 call    sub_508CF0           ;
0053619C                                              ; PreviousTickCount = PreviousTick + dword_81BE98
005361A1                 mov     edx, dword_81BE98
005361A7                 mov     eax, [esp+40h+PreviousTickCount]
005361AB                 add     esp, 4
005361AE                 add     eax, edx
005361B0                 mov     [esp+3Ch+PreviousTickCount], eax
005361B4
005361B4 loc_5361B4:                                  ; CODE XREF: WinMain(x,x,x,x)+847↑j
005361B4                 call    nullsub_2
005361B9                 jmp     LABEL_WinMain_BeginOfMessageLoop
```

```
0053615F: WinMain(x,x,x,x)+83F
```

Is it starting to make sense already? I decompiled some of the assembler stuff manually to C representation. The next thing that comes to mind is: what is the meaning of dword_081BE98?

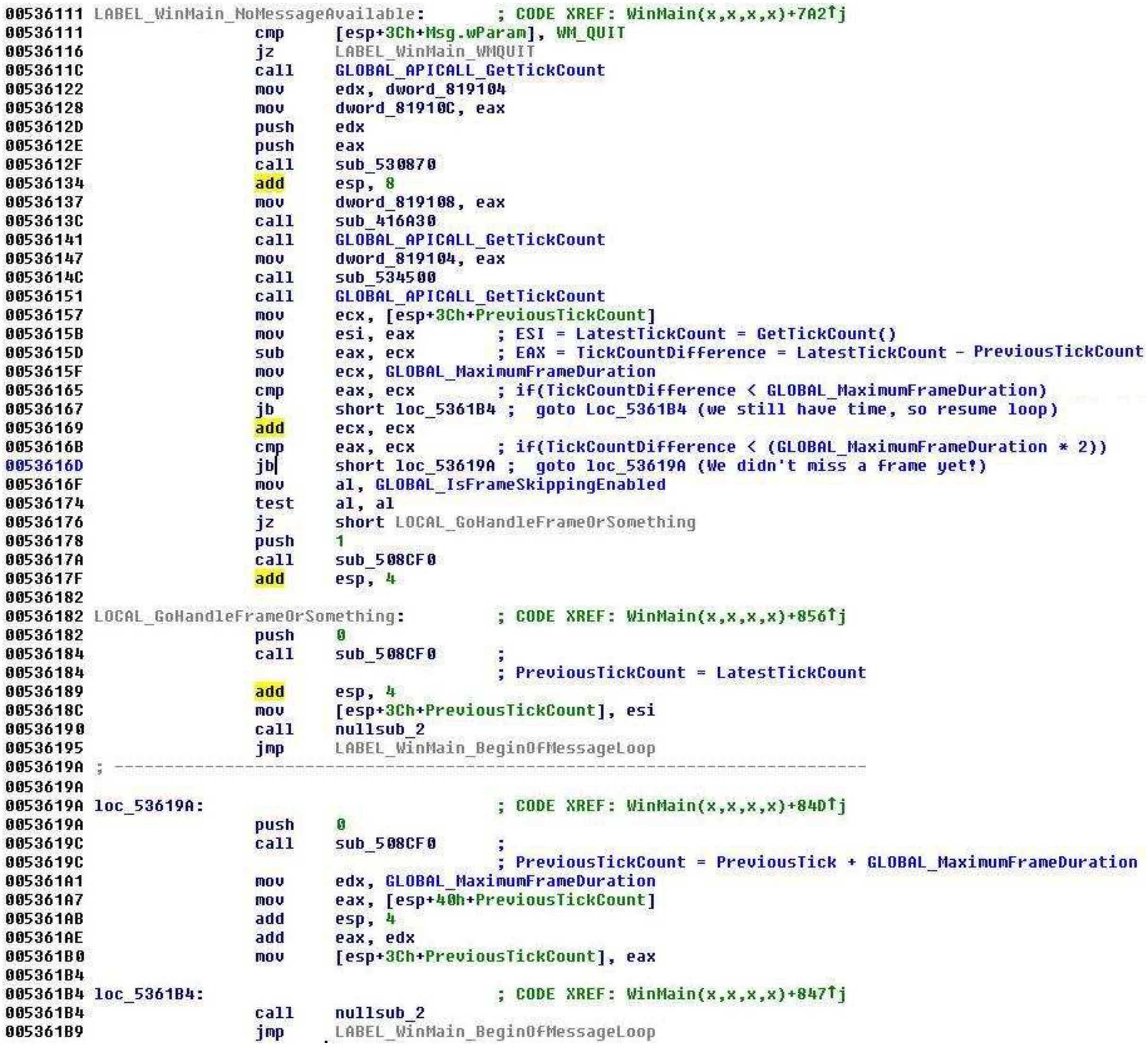
It turns out to be a constant (=fixed value) with a value of 50 decimal or 32 hexadecimal. I programmed games myself once and I too used a value 50 for frame rate control. ☺ My experience came in handy here.



```
mov     dword_81BC3C, 1
mov     dword_81BC60, ebx
mov     dword_81BC64, ebx
mov     dword_81BC68, ebx
mov     dword_81BC48, ebx
mov     dword_81C080, ebx
mov     dword_81BE98, 32h
mov     dword_81BC90, eb
mov     hostshort, si
mov     word_81BC94, 90h
mov     word_81BC8C, ax
mov     word_81BC8A, 3E3
mov     word_81BC88, ax
mov     dword_81BB34, eb
mov     dword_819604, eb
mov     dword_81BC6C, eb
call    sub_4DEDE0
call    sub_4C9250
push    1
push    offset aUo_cfg
mov     byte_819608, bl
mov     dword_819314, eb
call    sub_4D4CA0
mov     ecx, dword 81960
```

Also notice the reference aUo_cfg which is a string “uo.cfg”, so basically this tick count control thing is initialized while loading the configuration file.

We are slowly starting to understand what’s going on at WinMain_NoMessageAvailable. This is the function we need to patch to add some Sleep.



```
00536111 LABEL_WinMain_NoMessageAvailable:       ; CODE XREF: WinMain(x,x,x,x)+7A2↑j
00536111                 cmp     [esp+3Ch+Msg.wParam], WM_QUIT
00536116                 jz      LABEL_WinMain_WMQUIT
0053611C                 call    GLOBAL_APICALL_GetTickCount
00536122                 mov     edx, dword_819104
00536128                 mov     dword_81910C, eax
0053612D                 push    edx
0053612E                 push    eax
0053612F                 call    sub_530870
00536134                 add     esp, 8
00536137                 mov     dword_819108, eax
0053613C                 call    sub_416A30
00536141                 call    GLOBAL_APICALL_GetTickCount
00536147                 mov     dword_819104, eax
0053614C                 call    sub_534500
00536151                 call    GLOBAL_APICALL_GetTickCount
00536157                 mov     ecx, [esp+3Ch+PreviousTickCount]
0053615B                 mov     esi, eax        ; ESI = LatestTickCount = GetTickCount()
0053615D                 sub     eax, ecx        ; EAX = TickCountDifference = LatestTickCount - PreviousTickCount
0053615F                 mov     ecx, GLOBAL_MaximumFrameDuration
00536165                 cmp     eax, ecx        ; if(TickCountDifference < GLOBAL_MaximumFrameDuration)
00536167                 jb      short loc_5361B4 ;  goto Loc_5361B4 (we still have time, so resume loop)
00536169                 add     ecx, ecx
0053616B                 cmp     eax, ecx        ; if(TickCountDifference < (GLOBAL_MaximumFrameDuration * 2))
0053616D                 jb      short loc_53619A ;  goto loc_53619A (We didn't miss a frame yet!)
0053616F                 mov     al, GLOBAL_IsFrameSkippingEnabled
00536174                 test    al, al
00536176                 jz      short LOCAL_GoHandleFrameOrSomething
00536178                 push    1
0053617A                 call    sub_508CF0
0053617F                 add     esp, 4
00536182
00536182 LOCAL_GoHandleFrameOrSomething:         ; CODE XREF: WinMain(x,x,x,x)+856↑j
00536182                 push    0
00536184                 call    sub_508CF0      ;
00536184                                         ; PreviousTickCount = LatestTickCount
00536189                 add     esp, 4
0053618C                 mov     [esp+3Ch+PreviousTickCount], esi
00536190                 call    nullsub_2
00536195                 jmp     LABEL_WinMain_BeginOfMessageLoop
0053619A ; ---------------------------------------------------------------------------
0053619A
0053619A loc_53619A:                             ; CODE XREF: WinMain(x,x,x,x)+84D↑j
0053619A                 push    0
0053619C                 call    sub_508CF0      ;
0053619C                                         ; PreviousTickCount = PreviousTick + GLOBAL_MaximumFrameDuration
005361A1                 mov     edx, GLOBAL_MaximumFrameDuration
005361A7                 mov     eax, [esp+40h+PreviousTickCount]
005361AB                 add     esp, 4
005361AE                 add     eax, edx
005361B0                 mov     [esp+3Ch+PreviousTickCount], eax
005361B4
005361B4 loc_5361B4:                             ; CODE XREF: WinMain(x,x,x,x)+847↑j
005361B4                 call    nullsub_2
005361B9                 jmp     LABEL_WinMain_BeginOfMessageLoop
```

Look at the code paths above, loc_5361B4 is the most interesting one. Because that one is called only when the client has time left. Loc_53619A on the other hand is called when animation needs to be done. Very interesting stuff, also note that we can partially see how frame-skipping is implemented, to further analyze frame-skipping we must look at sub_508CF0 and see what happens what the argument is 1 (see 00536178).

To summarize:
GoHandleFrameOrSomething is called when the tick count difference >= 100.

loc_53619A is called when tick count difference < 100 and >= 50.

loc_5361B4 is called directly when tick count difference < 50 and when tick count difference < 100.

Therefore loc_53619B4 is the most suitable place to patch.

We currently have:

```
  jmp LABEL_WinMain_BeginOfMessageLoop
```

The code must become:

```
  push 1
  call Sleep
  jmp LABEL_WinMain_BeginOfMessageLoop
```

This is the same in binary (opcode representation):

```
  E9 98 FE FF FF
```

To:

```
  6A 01
  FF 15 2C B2 57 00
  E9 ?? ?? ?? ??
```

→ Originally we 5 bytes, our modified version is 11 bytes (2+6+5)

NOTE: the modified jump to LABEL_WinMain_BeginOfMessageLoop cannot easily be coded because we do not know its relative location yet, an alternative form is to store the address in a register and then jump to a register:

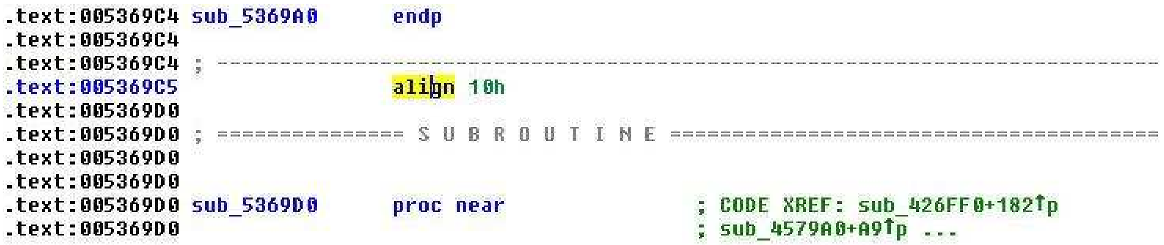
```
  push 1
  call Sleep
  mov eax, offset LABEL_WinMain_BeginOfMessageLoop
  jmp eax
```

This assembles to:

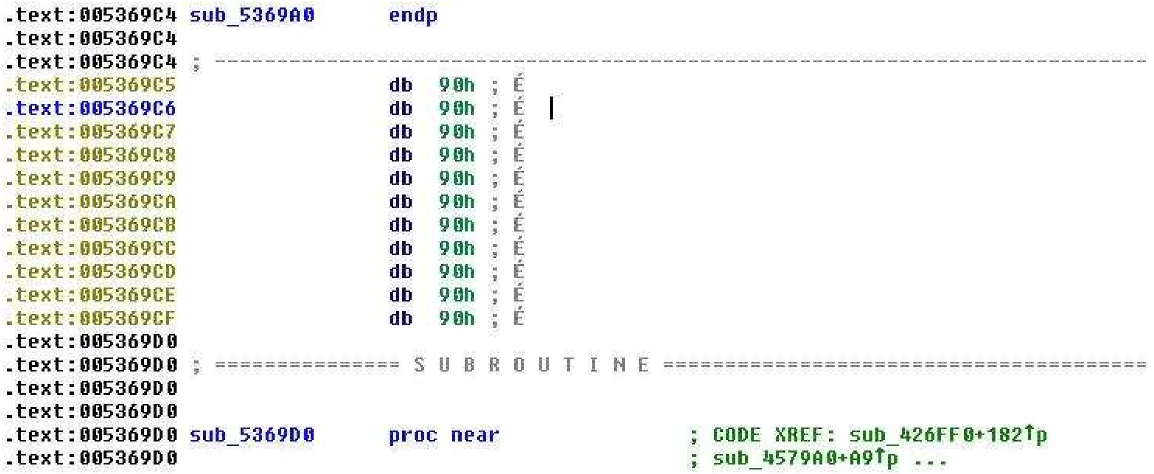
```
  6A 01
  FF 15 2C B2 57 00
  B8 56 60 53 00
  FE E0
```

-> 2 bytes more, thus 13 bytes are needed for such a patch.

To add code into the client we must locate some useable code. We can undefine alignments and see if we have enough space to insert our code.

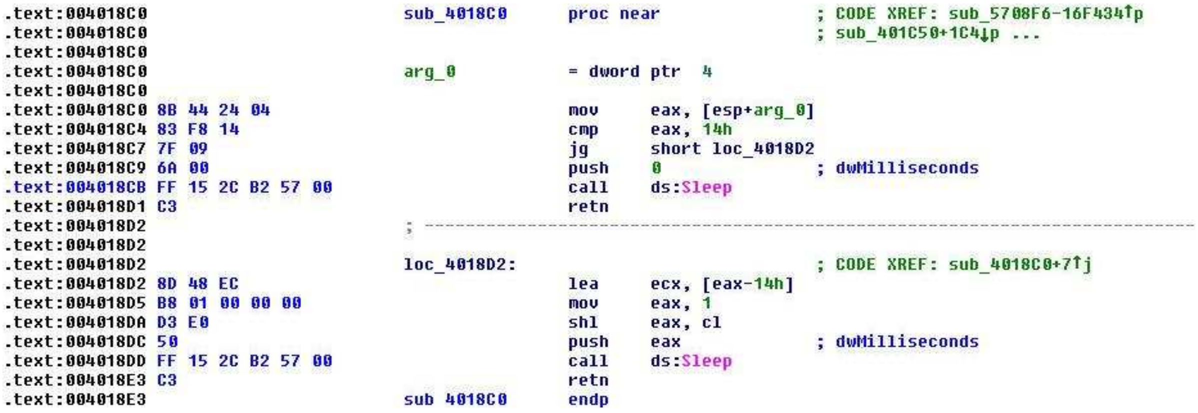


```
.text:005369C4 sub_5369A0      endp
.text:005369C4
.text:005369C4 ; ---------------------------------------------------------------------------
.text:005369C5                 align 10h
.text:005369D0
.text:005369D0 ; =============== S U B R O U T I N E =======================================
.text:005369D0
.text:005369D0
.text:005369D0 sub_5369D0      proc near               ; CODE XREF: sub_426FF0+182↑p
.text:005369D0                                         ; sub_4579A0+A9↑p ...
```



```
.text:005369C4 sub_5369A0      endp
.text:005369C4
.text:005369C4 ; ---------------------------------------------------------------------------
.text:005369C5                 db  90h ; É
.text:005369C6                 db  90h ; É
.text:005369C7                 db  90h ; É
.text:005369C8                 db  90h ; É
.text:005369C9                 db  90h ; É
.text:005369CA                 db  90h ; É
.text:005369CB                 db  90h ; É
.text:005369CC                 db  90h ; É
.text:005369CD                 db  90h ; É
.text:005369CE                 db  90h ; É
.text:005369CF                 db  90h ; É
.text:005369D0
.text:005369D0 ; =============== S U B R O U T I N E =======================================
.text:005369D0
.text:005369D0
.text:005369D0 sub_5369D0      proc near               ; CODE XREF: sub_426FF0+182↑p
.text:005369D0                                         ; sub_4579A0+A9↑p ...
```

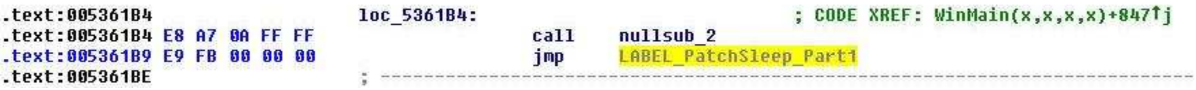
Let’s also look up the usage of existing calls to the Sleep API, the fact that the UO Client does use the Sleep function somehow makes it easier to re-use it for our own purpose.



```
.text:004018C0                      sub_4018C0      proc near               ; CODE XREF: sub_5708F6-16F434↑p
.text:004018C0                                                              ; sub_401C50+1C4↓p ...
.text:004018C0
.text:004018C0                      arg_0           = dword ptr  4
.text:004018C0
.text:004018C0 8B 44 24 04                          mov     eax, [esp+arg_0]
.text:004018C4 83 F8 14                             cmp     eax, 14h
.text:004018C7 7F 09                                jg      short loc_4018D2
.text:004018C9 6A 00                                push    0               ; dwMilliseconds
.text:004018CB FF 15 2C B2 57 00                    call    ds:Sleep
.text:004018D1 C3                                   retn
.text:004018D2                      ; ---------------------------------------------------------------------------
.text:004018D2
.text:004018D2                      loc_4018D2:                             ; CODE XREF: sub_4018C0+7↑j
.text:004018D2 8D 48 EC                             lea     ecx, [eax-14h]
.text:004018D5 B8 01 00 00 00                       mov     eax, 1
.text:004018DA D3 E0                                shl     eax, cl
.text:004018DC 50                                   push    eax             ; dwMilliseconds
.text:004018DD FF 15 2C B2 57 00                    call    ds:Sleep
.text:004018E3 C3                                   retn
.text:004018E3                      sub 4018C0      endp
```

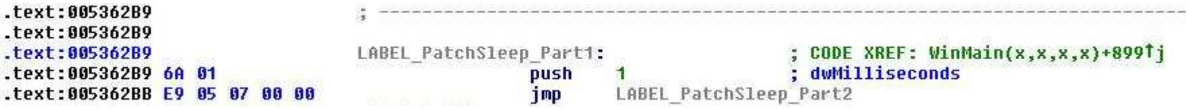
The actual patch, I will now show 3 screenshots with the actual patch applied. Each screenshot shows how a jump is made to the next part of the patch. If you aren't as lazy as me, you can write a utility that will locate 11 unused bytes (by alignment) and then you can put the patch into one single block. But again, I tend to be lazy, sometimes.

PATCHED CODE BLOCK 1:



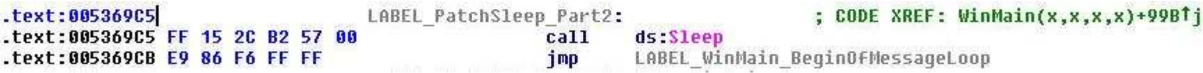
```
.text:005361B4                 loc_5361B4:                             ; CODE XREF: WinMain(x,x,x,x)+847↑j
.text:005361B4 E8 A7 0A FF FF                  call    nullsub_2
.text:005361B9 E9 FB 00 00 00                  jmp     LABEL_PatchSleep_Part1
.text:005361BE                 ; ---------------------------------------------------------------------------
```

PATHCED CODE BLOCK 2:



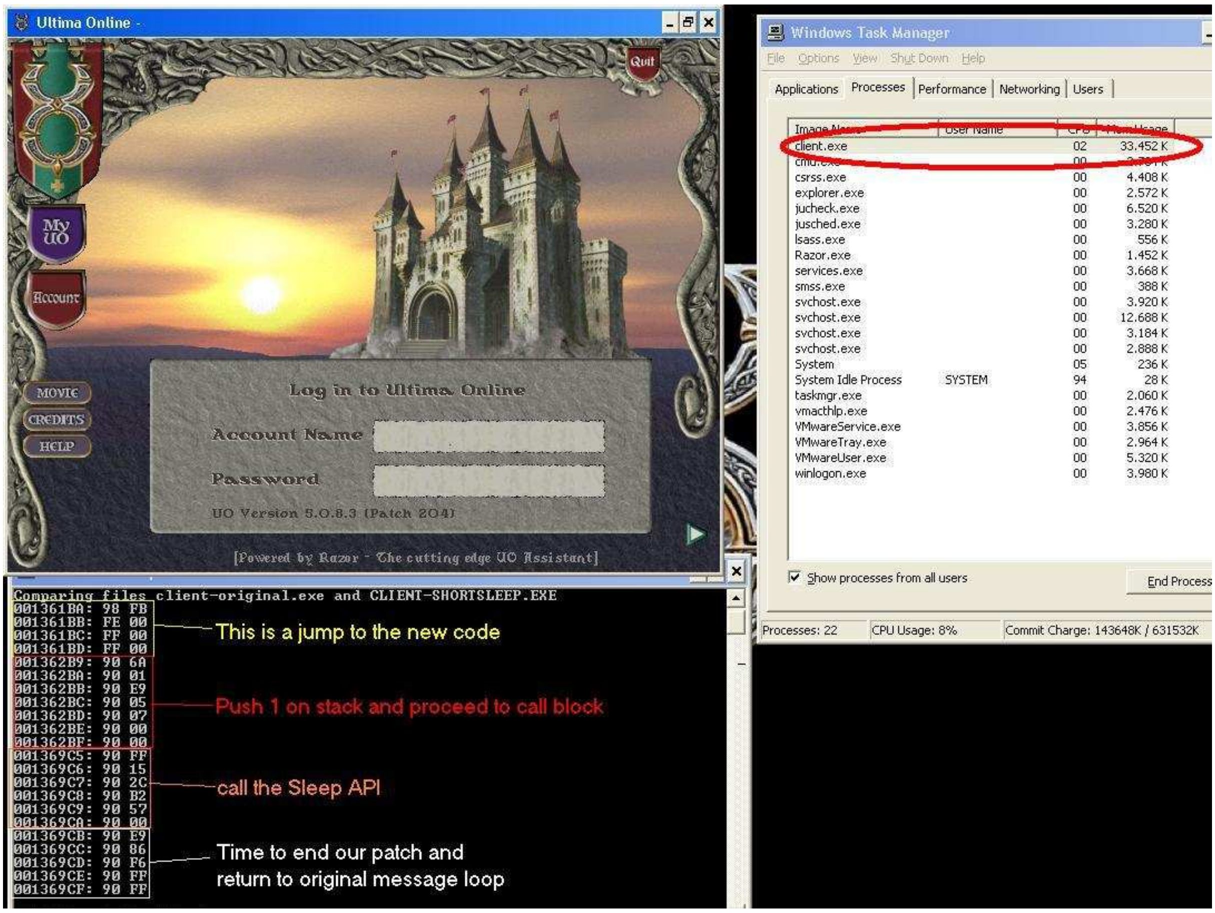
```
.text:005362B9                 ; ---------------------------------------------------------------------------
.text:005362B9
.text:005362B9                 LABEL_PatchSleep_Part1:                 ; CODE XREF: WinMain(x,x,x,x)+899↑j
.text:005362B9 6A 01                           push    1               ; dwMilliseconds
.text:005362BB E9 05 07 00 00                  jmp     LABEL_PatchSleep_Part2
```

PATCHED CODE BLOCK 3:



```
.text:005369C5                 LABEL_PatchSleep_Part2:                 ; CODE XREF: WinMain(x,x,x,x)+99B↑j
.text:005369C5 FF 15 2C B2 57 00               call    ds:Sleep
.text:005369CB E9 86 F6 FF FF                  jmp     LABEL_WinMain_BeginOfMessageLoop
```

And this is a screen shot of a binary comparison of the Sleep patch for the Ultima Online Client Version 5.0.8.3, with proof that it actually works:



Ultima Online -
Quit
My UO
Account
MOVIE
CREDITS
HELP
Windows Task Manager
File Options View Shut Down Help
Applications Processes Performance Networking Users
End Process

Log in to Ultima Online

Account Name [redacted]

Password [redacted]

UO Version 5.0.8.3 (Patch 204)

[Powered by Razor - The cutting edge UO Assistant]

```
Comparing files client-original.exe and CLIENT-SHORTSLEEP.EXE
001361BA: 98 FB
001361BB: FE 00
001361BC: FF 00
001361BD: FF 00
001362B9: 90 6A
001362BA: 90 01
001362BB: 90 E9
001362BC: 90 05
001362BD: 90 07
001362BE: 90 00
001362BF: 90 00
001369C5: 90 FF
001369C6: 90 15
001369C7: 90 2C
001369C8: 90 B2
001369C9: 90 57
001369CA: 90 00
001369CB: 90 E9
001369CC: 90 86
001369CD: 90 F6
001369CE: 90 FF
001369CF: 90 FF
```

This is a jump to the new code

Push 1 on stack and proceed to call block

call the Sleep API

Time to end our patch and
return to original message loop

Image Name	User Name	CPU	Mem Usage
client.exe		02	33.452 K
cmd.exe		00	[illegible]
csrss.exe		00	4.408 K
explorer.exe		00	2.572 K
jucheck.exe		00	6.520 K
jusched.exe		00	3.280 K
lsass.exe		00	556 K
Razor.exe		00	1.452 K
services.exe		00	3.668 K
smss.exe		00	388 K
svchost.exe		00	3.920 K
svchost.exe		00	12.688 K
svchost.exe		00	3.184 K
svchost.exe		00	2.888 K
System		05	236 K
System Idle Process	SYSTEM	94	28 K
taskmgr.exe		00	2.060 K
vmacthlp.exe		00	2.476 K
VMwareService.exe		00	3.856 K
VMwareTray.exe		00	2.964 K
VMwareUser.exe		00	5.320 K
winlogon.exe		00	3.980 K

☑ Show processes from all users

Processes: 22 CPU Usage: 8% Commit Charge: 143648K / 631532K