INSIDE THE ULTIMA ONLINE GOLD DEMO - THE PACKET COMMUNICATION - PART 2

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

<u>IDA Pro</u>, a very professional utility, definitely worth buying, Standard version is affordable. <u>HxD</u>, a very neat hex editor and above all, it's free <u>Explorer Suite</u>, it did the job for this project but the tool can be improved

ABOUT ME

I'm just a guy who loves the Ultima universe and knows a bit assembler. Why not combine the two?

I learned programming in BASIC, and then I switched to C and from there to assembler. I think that's the most logical order and I believe every one should learn to program like that. BASIC because it's easy, C because it's a language where other languages are based upon (C++, Java, C#, …) and at the same time you have to stay focused to avoid buffer overruns (which I consider to be an important skill). Assembler will teach you how a program works and it doesn't open a door or a window, it opens a gate to the inner workings of your computer.

SMALL RECAPULATION

In part 1 of The Packet Communication we learned where and how the sending and receiving of the packets is done (at the transfer level, not at the game level). Today's goal is to insert code to enable logging of these packets.

It's my decision to use the log format of the Razor application. Razor is a macro utility for free shards. This log format is clear text so files will become big quickly. I could have chosen to create my own binary format which will be easier to implement but this means that yet another log format would have been born. That my dear reader I wanted to avoid!

WHERE TO START THE PATCH

My first idea was to modify the constructor of the MemoryTransferOC object, however; this object is initialized before the critical section is entered. If we have two log files, one for client to server and one for server to client, this wouldn't be much of a problem. That is however a silly idea.

Remember, we have to keep thread-safety into mind so it's better to insert our code between the EnterCriticalSection and LeaveCriticalSection calls.

```
ecx, [ebp+THIS_MemoryTransfer20]
004E72B5 mov
                                                  ; lpCriticalSection
004E72B8 push
                 PCX
004E72B9 call
                 ds:EnterCriticalSection
                 edx, [ebp+THIS MemoryTransfer20]
004E72BF mov
                 [edx+struct MemoryTransfer20.MemoryTransfer0C LinkedList], 0
004E72C2 CMP
                 short loc_4E72DC
004E72C6 inz
                 eax, [ebp+THIS_MemoryTransfer20]
004E72C8 mov
004E72CB mov
                 ecx, [ebp+VAR MemoryTransfer@C 2]
                 [eax+struct MemoryTransfer20.MemoryTransfer0C LinkedList], ecx
004E72CE MOV
                 edx, [ebp+THIS_MemoryTransfer20]
004F72D1 mov
                 eax, [ebp+VAR MemoryTransfer0C_2]
004E72D4 mov
004E72D7 mov
                 [edx+struct_MemoryTransfer20.MemoryTransfer0C_FirstInLinkedList], eax
004E72DA jmp
                 short loc_4E72F1
AA4F72DC
004E72DC
                                                  ; CODE XREF: FUNC MemoryTransfer Write+851j
004E72DC loc 4E72DC:
                 ecx, [ebp+THIS_MemoryTransfer20]
004E72DC mov
                 edx, [ecx+struct_MemoryTransfer20.MemoryTransfer0C_LinkedList]
004E72DF mov
                 eax, [ebp+VAR_MemoryTransfer@C_2]
004E72E2 mov
004E72E5 mov
                 [edx+struct_MemoryTransfer@C.NextMemoryTransfer@C], eax
                 ecx, [ebp+THIS_MemoryTransfer20]
004E72E8 mov
                 edx, [ebp+VAR_MemoryTransfer0C_2]
004E72EB mov
                 [ecx+struct_MemoryTransfer20.MemoryTransfer0C_LinkedList], edx
884F72FF mov
004E72F1
                                                  ; CODE XREF: FUNC_MemoryTransfer_Write+99<sup>†</sup>j
004E72F1 loc_4E72F1:
                 eax, [ebp+THIS MemoryTransfer20]
004E72F1 mov
                                                  ; 1pCriticalSection
004E72F4 push
                 eax
                 ds:LeaveCriticalSection
004E72F5 call
004E72FB mov
                 eax, [ebp+ARG_BytesToSend]
```

Do you see place to insert code? I didn't.

We can redesign the whole function and optimize it correctly to gain space. However, I took a different approach. I replaced the call to LeaveCriticalSection with a call to inserted code plus I added code to make sure we can use the fastcall.

WHAT IS FASTCALL

Fastcall is a calling convention like any other ©. When using fastcall you are passing variables through registers and not through the stack (which is faster, hence fastcall). Extra reading: http://en.wikipedia.org/wiki/X86_calling_conventions#fastcall.

WHAT TO INSERT

The code we are adding in this patch is really big compared with previous patches so I decided to code and test this in C. Remember, modern C compilers support 64-bit time_t values, but due to its age the uodemo is limited to 32-bit. Luckily you can tell modern C compilers to use the old-style time_t standard.

WARNING! With this patch I could be introducing the <u>Year 2038 bug</u> in the demo because of using 32-bit time_t!

The actual C code is attached at the end of this document!

THE PATCH ENTRY

The screenshot below shows the modified FUNC_MemoryTransfer_Write function, notice the added "XCHG EAX, EDX":

```
004E72C2 83 7A 1C 00
                                           [edx+struct MemoryTransfer20.MemoryTransfer0C LinkedList], 8
                                  CMD
004E72C6 75 14
                                           short loc_4E72DC
                                  jnz
004E72C8 8B 45 E4
                                  mov
                                           eax, [ebp+THIS_MemoryTransfer20]
                                           eax, [ebp+VAR MemoryTransfer20]
ecx, [ebp+VAR MemoryTransfer0C_2]
[eax+struct_MemoryTransfer20.MemoryTransfer0C_LinkedList], ecx
004E72CB 8B 4D F0
                                  mov
004E72CE 89 48
                10
                                  mov
                                           edx, [ebp+THIS_MemoryTransfer20]
004E72D1 8B 55
                                  mov
                                           eax, [ebp+VAR_MemoryTransfer0C_2]
[edx+struct_MemoryTransfer20.MemoryTransfer0C_FirstInLinkedList], eax
004E72D4 8B 45
                                  mov
004E72D7
          89
             42
                18
                                  mov
004E72DA EB
                                           short loc_4E72F2
                                  jmp
004E72DC
004E72DC
004E72DC
                                  1oc_4E72DC:
                                                                               ; CODE XREF: FUNC_MemoryTransfer_Write+85<sup>†</sup>j
004E72DC 8B 4D E4
                                                [ebp+THIS_MemoryTransfer20]
                                  mov
004E72DF
                                           edx, [ecx+struct_MemoryTransfer20.MemoryTransfer0C_LinkedList]
         88
                10
                                  mov
004E72E2
                                                [ebp+VAR MemoryTransfer0C 2]
                                  MOV
                                           eax.
004E72E5 89 42
                                           [edx+struct_MemoryTransfer0C.NextMemoryTransfer0C], eax
BB4F72F8 8B 4D F4
                                  mnu
                                           ecx, [ebp+THIS_MemoryTransfer20]
004E72EB 8B
                                           edx. [ebp+VAR MemoruTransfer@C 2]
                                  MOV
004E72EE 89
                                           [ecx+struct_MemoryTransfer20.MemoryTransfer0C_LinkedList], edx
                                  mov
004E72F1
         92
                                  xchg
004E72F2
                                                                               ; CODE XREF: FUNC_MemoryTransfer_Write+99<sup>†</sup>j
004E72F2
                                  1oc 4E72F2:
                                           edx, [ebp+THIS_MemoryTransfer20]
004E72F2 8B 55 E4
                                  mov
004F72F5 52
                                  push
                                           edx
004E72F6 E8 27 5D 10 00
                                           FUNC LoggerEntry Patch
                                  call
004E72FB 8B
             45 OC
                                  mov
                                           eax, [ebp+ARG_BytesToSend]
```

The EAX will contain a pointer to the MemoryTransfer0C object, remember, that object contains the pointer to the packet data and contains the packet size.

This is the called FUNC_LoggerEntry function (I added __Patch so I know during the debugging sessions that it's a patch that is being called):

```
; CODE XREF: FUNC_MemoryTransfer_Write+B5f;
005ED022
                                 FUNC_LoggerEntry_Patch proc near
005ED022
005ED022
                                 ARG EIPofCaller FarAway= dword ptr 2Ch
005ED 022
                                         ecx, [eax+struct_MemoryTransfer0C.MemorySize]
005ED022 8B 48 04
                                 mov
                                         edx, [eax+struct_MemoryTransfer0C.MemoryBlock]
eax, [esp+ARG_EIPofCaller_FarAway]
005ED025 8B 10
                                 mov
005ED027 8B 44 24 2C
                                 mov
005ED02B FF 15 60 AC 9A 00
                                 call.
                                          ds:GLOBAL_LogFunction
005ED031 FF 25 E0 55 9A 00
                                         ds:LeaveCriticalSection
                                 imp
                                 FUNC_LoggerEntry_Patch endp
005ED 031
```

I will clarify that function a bit; it looks odd if you don't know assembler that well.

We know from the previous screenshot that EAX is used to pass the pointer to the MemoryTransferOC object, so the memory size and memory block (packet) is placed into respectively ECX and EDX (fastcall at work again).

Then ARG_EIPofCaller_FarAway is put into EAX. This EIP is **not** the address of the FUNC_MemoryTransfer_Write. It is the EIP of where FUNC_MemoryTransfer will return to. That EIP will tell us whether it is the server code that is calling the write function or whether it is the client code that is calling. This is important so the logger knows it should record Server->Client or Client->Server. You cannot code ARG_EIPofCaller_FarAway in C without the help of extra variables or assembler. This is the power of assembler at its best because we have direct access to the stack.

GLOBAL_LogFunction is a pointer to the log function (stored in memory). The idea is: the first time the function is called "LOGFILE" is looked up in the environment strings, if not found we will replace GLOBAL_LogFunction with a call to a "nullsub". If found we will replace GLOBAL_LogFunction with a call to the log function. That way we don't have to check every time whether or not the log file has been opened. It's a simply but effective speed optimization technique.

THE LOG FILE INITIALIZATION

By default, GLOBAL_LogFunction points to FUNC_LoggerInit_Patch:

This is the actual function:

```
; CODE XREF: FUNC LoggerEntry Patch+91p
005ED 037
                                         FUNC_LoggerInit_Patch proc near
005ED037
005ED037
                                                                                                 DATA XREF: UVA DATA: GLOBAL LogFunction Lo
                                         EIP Caller= dword ptr 20h
005FD037
005ED037 60
                                         nusha
005ED 038 89 E5
005ED 03A 68 38 E2 4E 00
                                                   ebp, esp
offset a
                                                                                               ; "LOGFILE"
                                                                 LOGFILE
                                         push
005ED03F BB
005ED044 E8
                00 7A 4E 00
07 D4 EF FF
                                         mov
call
                                                    ebx, offset nullsub_1
                                                   _getenv
'a'
005ED049 6A 61
005ED04B 91
                                         push
                                                   eax, ecx
                                         xchq
                                         push
005ED04C 54
                                                                                               ; Mode
005ED04D E3 5D
                                                    short LOCAL_Return
                                         jecxz
                                         push
call
                                                   ecx
_fopen
                                                                                               · Filename
005ED04F 51
005ED050 E8 48 CB EF FF
005ED055 09 C0
005ED057 A3 64
005ED05C 74 4E
                                         nr
                                                    eax. eax
                                         mov
                64 AC 9A 88
                                                    ds:GLOBAL_LogHandle, eax
                                         iz
                                                   short LOCAL Return
005ED05E 83 EC 20
005ED061 97
                                                   esp, 20h
                                         sub
                                         xchq
                                                   eax, edi
005ED062 89 E3
005ED064 6A 00
                                                    ebx, esp
                                         push
005ED064 6H 90
005ED066 54
005ED067 6A 05
005ED069 E8 62 28 F0 FF
005ED06E 6A 00
005ED070 E8 1B DD FE FF
                                         push
                                                    esp
                                         push
                                         call.
                                                     setlocale
                                                                                               ; Time
                                         push
                                                     time
                                         call
005ED075 50
                                         push
005ED076 54
                                         push
call
                                                                                               ; Time
                                                   esp
005ED077 E8 B4 EB EF FF
005ED07C 68 25 63 00 00
                                                        caltime
                                                    _c%,
                                         push
005ED081 89 E1
005ED083 50
                                                                                               ; Tm
; For
; Siz
; Buf
                                         push
                                                    eax
005ED 083 50
005ED 084 51
005ED 085 6A 20
005ED 087 53
005ED 088 E8 03 E0 EF FF
                                                                                                  Format
SizeInBytes
                                         push
                                                     20h ;
                                         push
                                         push
call
                                                    ehx
                                                     strftime
005ED08D 53
                                         push
                                                   ehx
005ED 08E 68 68 5F 60 00
005ED 093 57
                                         push
                                                    offset a_LoggingStartedAt
                                                                                                  "\r\n\r\n\r\n>>>>>> Logging started at %s "...
                                                                                               ; File
                                         push
                                                   _fprintf
                                                   edi
005ED094 E8 07 CD EF FF
                                         call
005ED099 6A 43
                                         push
005ED09B 54
                                         push
                                                   esp
005ED09C 6A 05
                                         push
005ED09E E8 2D 28 F0 FF
005ED0A3 BB B6 D0 5E 00
                                         call
                                                     setlocale
                                                    ebx, offset FUNC LoggerLogPacket Patch
                                         mov
005ED0A8 83 6D 20 06
                                         sub
                                                    [ebp+EIP_Caller],
005ED 0AC
005ED 0AC
                                         LOCAL Return:
                                                                                               ; CODE XREF: FUNC_LoggerInit_
; FUNC_LoggerInit_ Patch+25<sup>†</sup>j
                                                                                                                                       Patch+161j
005ED OAC
                                                    ds:GLOBAL LogFunction, ebx
005ED0AC 89 1D 60 AC 9A 00
005ED0B2 89 EC
                                         mnu
005ED0B4 61
                                         popa
005ED0B5 C3
005ED 0B5
                                         FUNC_LoggerInit_Patch endp
005ED 0B5
```

Notable things going on are:

- 1) EBX points to either "nullsub" or "FUNC_LoggerPacket_Patch" and is placed into GLOBAL_LogFunction at the end
- 2) setlocale(LC_TIME, "") is called, this makes the C API us your computers time settings
- 3) strftime is used to convert the current time to a string
- 4) setlocale(LC_TIME, "C") is called to restore the default C API settings (to minimize interference with the uodemo (if time functions are used that is))
- 5) "SUB [ebp+EIP_Caller], 6" will re-execute "CALL GLOBAL LogFunction"

THE LOGGING

This is the first part of the actual log function. Remember GLOBAL_LogHandle is not checked against NULL because this function can only be called when it has been set.

```
005ED 0B6
                                      FUNC_LoggerLogPacket__Patch proc near ; DATA XREF: FUNC_LoggerInit__Patch+6CTo
005ED 0B6
                                      var_E4= dword ptr -0E4h
var_DA= dword ptr -0DAh
var_78= byte ptr -78h
Dest= byte ptr -38h
SystemTime= _SYSTEMTIME
005ED 0B6
005ED 0B6
BASED BRA
                                                     _SYSTEMTIME ptr -32h
005ED 0B6
                                      var_1C= byte ptr -
arg_8= dword ptr
arg_10= dword ptr
arg_14= dword ptr
005FD 0R6
005ED 0B6
005ED 0B6
005ED 0B6
                                                              18h
005ED 0B6
                                      arg_18= dword ptr
005ED 0B6
                                                ecx, ecx
short LOCAL_Continue
005ED0B6
BOSEDBRR 75
               81
                                      jnz
                                      retn
                                                      005ED 0BB
005ED 0BB
005ED 0BB
                                      LOCAL Continue:
                                                                                         ; CODE XREF: FUNC_LoggerLogPacket__Patch+2<sup>†</sup>j
005ED 0BB 60
                                      pusha
005ED0BC 89 E5
005ED0BE 83 EC 78
                                      mov
                                                esp, 78h
edi, offset a_Server
                                      sub
005ED0C1 BF F9
005ED0C6 81 7D
                                      mov
                                                [ebp+arg_18], offset loc_513FE3
esi, offset a_Client
                   10
                      F3 3F 51 00
                                      CMD
005EDOCD BE
                                      mov
                                                                                         ; "Client"
                                                short LOCAL_LogDirectionOK
BBSFDBD2 75
                                      inz
005ED0D4 87 F7
                                                esi, edi
                                      xchq
005ED 0D6
005ED 0D6
                                      LOCAL LogDirectionOK:
                                                                                         ; CODE XREF: FUNC LoggerLogPacket Patch+1C1j
005ED0D6 A1 64 AC 9A 00
005ED0DB 8D 4D CE
                                                eax, ds:GLOBAL_LogHandle
                                      lea
                                                ecx, [ebp+SystemTime]
005ED0DE 93
                                      xchg
                                                                                         ; lpSystemTime
005ED 0DF 51
                                      push
                                                ecx
005ED0E0 FF
               15 A8 55 9A 00
                                                ds:GetLocalTim
                                      call
005ED0E6 8B
005ED0E9 FF
              45 14
75 18
                                                eax, [ebp+arg_10]
[ebp+arg_14]
                                      mnu
                                      push
005ED OEC OF
                                      .
MOVZX
                                                edx, byte ptr [eax]
005ED0EF 93
                                      xchq
                                                eax, ebx
005ED0F0 52
005ED0F1 0F
                                      push
                                                edx
               B7 4D DC
                                                ecx, [ebp+SystemTime.wMilliseconds]
                                      MOVZX
005ED0F5 56
                                      push
005ED0F6 57
                                      push
                                                edi
005ED0F7 51
                                      push
                                                ecx
BOSEDBER 97
                                      xcha
                                                eax. edi
                                                      [ebp+SystemTime.wSecond]
                                      MOVZX
                                                ecx,
               B7 55 D6
005ED OFD OF
                                      MOVZX
                                                edx, [ebp+SystemTime.wHour]
005ED101 51
                                      push
                                                ecx
005ED102 FF 75 D8
                                                dword ptr [ebp+SystemTime.wMinute]
                                      push
005ED105 52
                                      push
                                                edx
005ED106 68 A0 5F 60 00
                                                offset a_LogPacketHeader
                                      push
                                                                                            Format
                                      push
call
005ED10B 57
                                                edi
                                                                                         ; File
005ED10C E8 8F CC EF FF
                                                  fprintf
                                                esi, offset a_S_
offset a_LogText1
005ED111 BE D5 5F 60 00
005ED116 68 68 AC 9A 00
                                      mov
                                                                                         ; Format
                                                                                                                             6 7
                                      push
                                                                                           Format
005ED11B 56
                                      push
005ED11C 57
                                      push
                                                edi
                                                                                            File
005ED11D E8 7E CC EF FF
005ED122 68 A0 AC 9A 00
005ED127 56
                                                  fprintf
                                      call
                                                                                           Format
                                                offset a_LogText2
                                      push
                                                                                            Format
                                      push
005ED128 57
                                      push
call
                                                edi
                                                                                            File
005ED129 E8 72 CC EF FF
                                                 fprintf
005ED12E 31 C0
005ED130 89 45 0C
                                      xor
                                                [ebp+arg_8], eax
                                      mov
005ED133
005ED133
                                      LOCAL MainLoop:
                                                                                         ; CODE XREF: FUNC LoggerLogPacket Patch+12Fij
```

In the first stage, EDI and ESI are used to point a string either "Server" or "Client". Depending on the EAX register (ARG_EIPofCaller_FarAway) they are swapped or not.

Instead of the C API "time(NULL)", this time the Windows API "GetLocalTime" is used to obtain the current time. Why? Because GetLocalTime returns the current millisecond whereas the C API doesn't return this value at all. In a later stage, EDI is used to contain the log file handle (GLOBAL_LogHandle) and ESI points to a_S_ (which is "%s\r\n").

This big screenshot shows the second part of the logger function:

```
005ED133
                                       LOCAL_MainLoop:
                                                                                            ; CODE XREF: FUNC_LoggerLogPacket__Patch+12Fij
005ED133 56
                                       push
push
005ED134 57
                                                  edi
                                                                                            ; Dest
005ED135 6A
                                       push
005ED137 8B
005ED13A 58
               7D 18
                                        mov
                                                  edi, [ebp+arg_14]
                                       pop
                                                  eax
005ED13B 50
005ED13C 39 C7
                                       push
                                       cmp
lea
                                                  edi, eax
                                                  esi, [ebp+Dest]
short LOCAL_Ready1
005ED13E 8D
005ED141 76
                75 C8
                01
                                        ibe
005ED143 97
                                                  eax, edi
005ED144
005ED144
                                       LOCAL_Ready1:
                                                                                            ; CODE XREF: FUNC_LoggerLogPacket__Patch+8B1j
005ED144 53
                                       push
005ED145
                                       LOCAL Loop1:
                                                                                            ; CODE XREF: FUNC_LoggerLogPacket_Patch+A61j
005ED145 OF B6 03
                                        MOVZX
                                                  eax, byte ptr [ebx]
005ED148 50
                                       push
                                                  eax
                                                                                            ; "%02X "
; Dest
                                                  offset a_X_
005ED149 68 EC 5F 60 00
                                       push
005ED14E 56
                                        push
                                                  esi
                                                   sprintf
005ED14F E8 2C B6 EF FF
005ED154 83 C4 0C
                                       add
                                                  esp, OCh
005ED157 43
                                       inc
005ED158 83 C6 03
                                                  esi, 3
                                       add
005ED15B 4F
005ED15C 75 E7
                                       inz
                                                  short LOCAL Loop1
005ED15E 5B
                                       pop
005ED15F 59
                                       pop
lea
                                                  ecx
005ED160 8D 75 E4
                                                  esi, [ebp+var_10]
005ED163 8D
                                       lea
                                                  eax. [ecx+ecx]
005ED166 8B 7D 18
                                       mov
                                                  edi, [ebp+arg_14]
005ED169 50
                                       push
                                                  eax
005ED16A 39 C7
005ED16C 88 26
                                        cmp
                                                  [esi], ah
short LOCAL_Ready2
                                       mov
005ED16E 76
005ED170 97
                                       xchg
                                                  eax, edi
005ED171
                                       LOCAL Ready2:
                                                                                            ; CODE XREF: FUNC_LoggerLogPacket__Patch+B8<sup>†</sup>j
005ED171
005ED171 29 CF
005ED173 76 1D
                                                  edi, ecx
short LOCAL_SkipLoop2
                                       sub
                                       ibe
005ED175 53
005ED176 01 CB
                                                  ebx. ecx
                                       add
005ED178
005ED178
                                       LOCAL Loop2:
                                                                                            ; CODE XREF: FUNC LoggerLogPacket Patch+D911
095ED178 0F B6 03
005ED178 50
005ED17C 68 EC 5F 60 00
005ED181 56
                                       movzx
                                                  eax, byte ptr [ebx]
                                       push
                                                  eax
                                       push
                                                  offset a_X_
                                                                                            ; "%02X "
; Dest
                                       push
                                                  esi
985ED182 E8 F9 B5 EF FF
985ED187 83 C4 9C
985ED18A 43
                                                   sprinte
                                        call
                                       add
                                                  esp, OCh
                                       inc
                                                  ehx
005ED18B 83 C6 03
                                                  esi, 3
                                       add
005ED18E 4F
005ED18F 75 E7
                                       dec
                                                  edi
                                                  short LOCAL_Loop2
                                       jnz
005ED191 5B
005ED192
                                       pop
                                                  ebx
                                                                                            ; CODE XREF: FUNC_LoggerLogPacket__Patch+BDfj
                                       LOCAL_SkipLoop2:
005FD192
005ED192
                                       pop
                                       pop
lea
005ED193 5F
                                                  edi
005ED194 8D
                                                  eax, [ebp+var_10]
005ED197 8D 4D C8
                                       lea
                                                  ecx, [ebp+Dest]
005ED19A 50
                                       push
                                                  eax
005ED19B 51
                                       push
1ea
                                                  PCX
005ED19C 8D 45 88
005ED19F FF 75 0C
                                                  eax, [ebp+var_78]
                                       push
                                                  [ebp+arg_8]
offset a_LogPacketData
005ED1A2 68 DA 5F 60 00
                                       push
                                                                                            : "%84X
                                                                                                         %-255%-265"
                                       push
call
005ED1A7 50
                                                  eax
                                                                                            : Dest
005ED1A8 E8 D3 B5 EF FF
                                                   sprintf
005ED1AD 83 C4 14
005ED1B0 8D 54 05
                                                  esp, 14h
edx, [ebp+eax+var_78]
                                       add
                                       1ea
905ED1B4 8B 4D 18
905ED1B7 39 F1
905ED1B9 76 92
905ED1BB 89 F1
                                       mou
                                                  ecx, [ebp+arg_14]
ecx, esi
                                       cmp
                                                  short LOCAL_Ready3andLoop3
                                       ibe
                                       mov
                                                  ecx, esi
005ED1BD
                                                                                            ; CODE XREF: FUNC_LoggerLogPacket_
; FUNC_LoggerLogPacket__Patch+112↓j
005ED1BD
                                       LOCAL_Ready3andLoop3:
                                                                                                                                         Patch+1031j
005FD1BD
005ED1BD 8A 03
005ED1BF E8 31 01 F0 FF
005ED1C4 88 02
                                                  al, [ebx]
FUNC_LoggerFixCharacter__Patch
[edx], al
                                       cal1
                                       mov
005ED1C6 43
                                       inc
                                                  ehx
005ED1C7 42
                                                                                            ; File
                                                  edx
                                        inc
005ED1C8 E2 F3
005ED1CA 88 0A
005ED1CC 8D 45 88
                                       100p
                                                  LOCAL_Ready3andLoop3
                                       mov
                                                  [edx], cl
                                       lea
                                                  eax, [ebp+var_78]
005ED1CF 50
                                       push
                                                  eax
005ED1D0 68 D5 5F 60 00
005ED1D5 57
005ED1D6 E8 C5 CB EF FF
                                       push
                                                  offset a_S_
                                                                                                "%5\r\n"
                                       push
                                                                                            ; File
                                                   fprintf
                                       call.
005ED1DB 83 C4 0C
005ED1DE 01 75 0C
                                       add
                                                  esp, OCh
                                                  [ebp+arg_8], esi
[ebp+arg_14], esi
                                       add
005ED1E1 29 75 18
005ED1E4 5E
005ED1E5 OF 87 48 FF FF FF
                                                  LOCAL MainLoop
                                                                                            ; Format
```

I will not explain the function completely as it is based on the C code which you can see later in this document.

Finally, the third and final part of the logger function:

```
005ED1EB 8D 46 02
                                        eax, [esi+2]
                                                                           ; Format
                               Tea
005ED1EE 50
                                push
                                        eax
005ED1EF 56
                                                                           ; Format
                                push
                                        esi
005ED1F0 57
                                        edi
                                                                            File
                                push
005ED1F1 E8 AA CB EF FF
                                call.
                                         forintf
005ED1F6 57
                                        edi
                                                                           ; File
                               push
005ED1F7 E8 54 AD EF FF
                                call
                                         FFlush
005ED1FC 89 EC
                                        esp, ebp
                                MOV
005ED1FE 61
                               popa
005ED1FF C3
                                retn
                               FUNC_LoggerLogPacket__Patch endp
005ED1FF
```

Nothing much here, it will write a double "\r\n" and flush the file.

At this stage ESI points to a_S_ (which is "%s\r\n"), thus EAX = ESI + $2 = "\r\n"$. This is equal to "printf(LogHandle, "%s\r\n", "\r\n");".

THE STRINGS

Some of the strings I needed to add for this project were really long and I had to look really well where to add them. Later I will tell you how I did this.

For now, I'll just show you the strings added:

```
: DATA XREE: FUNC LongerInit Patch+5710
00605F68 3E 3E 20 4C 6F 67 67*db 0Dh,0Ah
00605F68 69 6E 67 20 73 74 61*db '>>>>>>> Logging started at %s <<<<<<', 0Dh,0Ah
00605F68 72 74 65 64 20 61 74+db 0Dh, 0Ah
00605F68 20 25 73 20 3C 3C 3C+db 0Dh, 0Ah, 0
00605FA0 ; char a_LogPacketHeader[]
, cliar a__Logracketheader db '%02u:%02hu:%02u.%04u: %s -> %s 0x%02X (Length: %u)',0Dh,0Ah
006:05FA0 32 68 75 3A 25 30 32+ ; DATA XREF: FUNC_LoggerLogPacket_Patch+5
                                                                                                ; DATA XREF: FUNC_LoggerLogPacket__Patch+50fo
00605FD4 00
                                         a_Unused_QuestionMark db
00605FD5
                                         ; char a_S_[]
a_S_ db '%s',0Dh,0Ah,0
                                                                                                ; DATA XREF: FUNC_LoggerLogPacket__Patch+5Bfo
; FUNC_LoggerLogPacket__Patch+11Afo
00605FD5 25 73 0D 0A 00
ARKASEDS
                                         ; char a__LogPacketData[]
00605FDA 25 30 34 58 20 20 20+4 LogPacketData db '%04X
00605FDA 25 2D 32 35 73 25 2D+
                                                                               %-255%-265',0
                                                                                                ; DATA XREF: FUNC LoggerLogPacket Patch+ECTo
                                         ; char a_X_[]
a_X_ db '%02X ',0
00605FEC
                                                                                                ; DATA XREF: FUNC_LoggerLogPacket__Patch+93fo
; FUNC_LoggerLogPacket__Patch+06fo
00605FEC 25 30 32 58 20 00
                                                                                                ; DATA XREF: FUNC_LoggerLogPacket_Patch+17<sup>†</sup>o
; DATA XREF: FUNC_LoggerLogPacket_Patch+B<sup>†</sup>o
00605FFF2 43 6C 69 65 6E 74 00 a Client db 'Client',0
00605FF9 53 65 72 76 65 72 00 a Server db 'Server',0
```

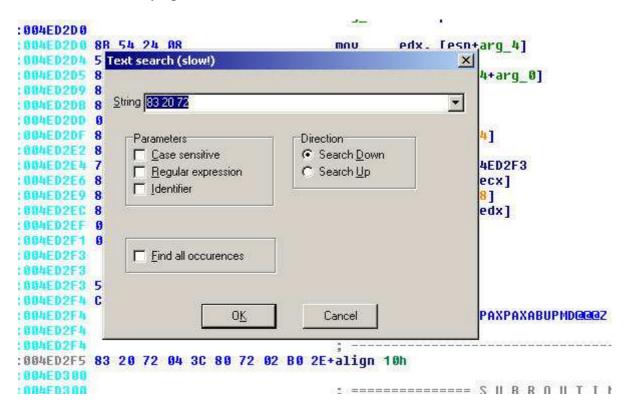
FUNC_LoggerFixCharacter__Patch

The main loop contained code to replace non-standard ASCII strings with a dot, same as Razor is doing. However, I needed more bytes so I had to place that code elsewhere and I replaced it with a single call.

```
Before:
89 F1
                                      ecx, esi
                              mov
                              loc_5ED1AA:
                                                                       ; CODE XREF: .text:005ED1BAij
8A 03
                                      al, [ebx]
                              mov
                              loc 5ED1AC:
                                                                       ; CODE XREF: .text:005ED1A61j
                                      al, 20h; ''
30 20
                              CMP
 72 04
                                      short loc_5ED1B4
                              jb
 30 80
                               cmp
                                      short loc_5ED1B6
                              loc_5ED1B4:
mov al, 2Eh ; '.'
                                                                       ; CODE XREF: .text:005ED1AETj
B0 2E
                              loc_5ED1B6:
                                                                       ; CODE XREF: .text:005ED1B2fj
 43
                              inc
                                       ehx
 88 02
                                       [edx], al
                              mov
 42
                              inc
                                      edx
                                      loc 5ED1AA
 E2 EE
                              loop
                              mov
                                      [edx], cl
After:
89 F1
                        mov
                                  ecx, esi
                        LOCAL Ready3andLoop3:
8A 03
                        mov
                                  al, [ebx]
E8 31 01 F0 FF
                                  FUNC_LoggerFixCharacter__Patch
                        call
88 02
                        mov
                                  [edx], al
43
                        inc
42
                        inc
                                  edx
E2 F3
                                  LOCAL_Ready3andLoop3
                        100p
88 OA
                        mov
                                  [edx], cl
```

I searched the EXE for empty space to place the code sequence "3c 20 72 04 3c 80 72 02 b0 2e c3". I used the search string "c3 90 90 90 90 90 90 90 90 90 90 955". This is a long NOP sequence (alignment) between a RET and a PUSH.

After editing the EXE I used IDA Pro's text search function to locate the modified code (I had no clue where exactly I placed the function):



After removing the alignment directive and documenting:

```
FUNC LoggerFixCharacter proc near
004ED2F5
004ED2F5 3C 20
                                        cmp
                                                 al, 20h; '
                                                 short LOCAL Change
004ED2F7 72 04
                                        jb
                                                 al, 80h ; 'C'
004ED2F9 3C 80
                                        cmp
                                                 short LOCAL ChangeDone
004ED2FB 72 02
                                        jb
004ED2FD
004ED2FD
                                        LOCAL Change:
004ED2FD B0 2E
                                                 al, 2Eh; '.'
004ED2FF
004ED2FF
                                        LOCAL ChangeDone:
004ED2FF C3
                                        retn
                                        FUNC LoggerFixCharacter endp
004ED2FF
004ED2FF
```

Looks better, right?

THE C TEST CODE

```
#define _USE_32BIT_TIME_T
#include <windows.h>
#include <stdio.h>
#include <locale.h>
#include <time.h>
#pragma warning( disable : 4996 )
static FILE *logfile = NULL;
enum DIRECTION
 RazorToServer, ClientToServer, ServerToClient
void LogPacket(DIRECTION PacketDirection, size_t PacketLength, const BYTE *PacketData)
 const char *LogText1 = "
                              0 1 2 3 4 5 6 7 8 9 A B C D E F";
 const char *LogText2 = "
                              char LogLine[100];
 char Log8_1[8 * 3 + 1], Log8_2[8 * 3 + 1];
  // We must have an open logfile
 if(logfile == NULL)
   return;
  // Do not log empty packets
 if(PacketLength == 0)
   return;
  // Part 1
   // Get the current time as string
   time_t t = time(NULL);
   GetLocalTime((LPSYSTEMTIME) LogLine); // (use LogLine also for SYSTEMTIME)
   strftime(LogLine, sizeof(LogLine), "%X", localtime(&t));
   // Verify the packet direction
   char *LogDirection1, *LogDirection2;
   switch(PacketDirection)
     case ClientToServer: LogDirection1 = "Client";
                        LogDirection2 = "Server";
                        break;
     case ServerToClient: LogDirection1 = "Server";
                        LogDirection2 = "Client";
                        break;
     default: return;
   ((LPSYSTEMTIME) LogLine)->wMilliseconds, LogDirection1, LogDirection2, PacketData[0],
PacketLength, LogText1, LogText2);
  // Part 2
 unsigned int per16counter = 0;
   register unsigned int Log8len, MaxCounter;
   Log8len = 0;
   MaxCounter = PacketLength < 8 ? PacketLength : 8;</pre>
   for(register unsigned int Counter = 0; Counter < MaxCounter; Counter ++)</pre>
     sprintf(Log8_1 + Log8len, "%02X ", PacketData[Counter]); // TEST
     Log8len += 3;
```

```
if(PacketLength > 8)
     Log8len = 0;
     MaxCounter = PacketLength < 16 ? PacketLength - 8 : 8;</pre>
      for(register unsigned int Counter = 0; Counter < MaxCounter; Counter ++)</pre>
       sprintf(Log8_2 + Log8len, "%02X ", PacketData[Counter + 8]); // TEST
       Log8len += 3;
     }
    else
     Log8_2[0] = ' \0';
    // sprintf returns the number of characters printed
    // KNOW YOUR API'S AND RULE THE BINARY WORLD
   rnsigned int len = sprintf(LogLine, "%04X %-25s%-26s", per16counter, Log8_1, Log8_2);
   MaxCounter = PacketLength < 16 ? PacketLength : 16;</pre>
    for(Log8len = 0; Log8len < MaxCounter; Log8len ++)</pre>
      if(PacketData[Log8len] < 0x20 || PacketData[Log8len] >= 0x80)
       LogLine[len + Log8len] = '.';
      else
        LogLine[len + Log8len] = PacketData[Log8len];
    LogLine[len + Log8len] = '\0';
    fprintf(logfile, "%s\n", LogLine);
   per16counter += 16;
   PacketData += 16;
   PacketLength -= 16;
 while((int) PacketLength > 0);
  // Part 3
    fprintf(logfile, "\n\n");
   fflush(logfile);
}
void OpenLog(void)
 // Open the log file
 register char *envstring = getenv("LOGFILE");
 if(envstring != NULL)
   logfile = fopen(envstring, "a");
  // Write log text
 if(logfile != NULL)
   char LogTime[25];
   // Get the current time as string
   time_t t = time(NULL);
   struct tm *tm = localtime(&t);
   setlocale(LC_TIME, "");
   strftime(LogTime, sizeof(LogTime), "%c", tm);
   setlocale(LC_TIME, "C");
   fprintf(logfile, "\n\n>>>>>> Logging started %s <<<<<<\n\n\n", LogTime);</pre>
}
```

ABOUT OPTIMAZATION TECHNIQUES

I had lot of problems with fitting the code inside the binary. So I had to relook how I implemented to log function and I rewrote it a few times before all the bytes I needed fitted in.

Remember, I optimized for size, not for speed!

Let's take a look at some techniques I used:

1) Replacing "MOV reg32, 0xXXXXXXXXX" with "PUSH 0xXX / POP reg32"

For example, "MOV EAX, 8" is assembled to "B8 08 00 00 00" (5 bytes) Now, "PUSH 8 / POP EAX" becomes "6A 08 58" (3 bytes) (but slower!)

2) Not cleaning up the stack after every call

This is actually a technique also used by modern compilers, because it's faster and uses less code bytes. I only clean-up the stack inside the MainLoop because inside there the stack is actually important and used with care.

3) locale(LC_TIME, "C")

By default, this becomes:

. .

PUSH offset a_C 68 XX XX XX XX

PUSH 5; LC_TIME 6A 05

CALL locale E8 XX XX XX XX

ADD ESP, 4 83 C4 04

. . .

a_C DB 'C', 0 43 00

 \rightarrow 5+2+5+3+2 = 17 bytes

I implemented this with:

. . .

PUSH 'C' 6A 43 PUSH ESP 54 PUSH 5 ; LC TIME 6A 05

CALL locale E8 XX XX XX XX

• • •

 \rightarrow 2+1+2+5 = 10 bytes

The trick is that "C" is a string consisting of 1 byte and 1 0-terminator. If you push 'C' (43 hex) on the stack then the value is zero extended on the stack and the stack will actually contain "43 00 00 00". Then ESP is pushed because ESP points to the freshly created string on the stack.

The same trick is applied when calling "localtime(&t)". Instead of placing the return-value of time(NULL) into a variable and then calling localtime with a reference to that variable, localtime is called directly after pushing EAX and pushing ESP (which points to the pushed EAX). I hope I could explain?

4) I also replaced comparisons against a constant with a comparison against a register, this takes one byte less in memory but you have to make sure the register contains the right constant (without adding extra code to put a constant into the register!).

5) CMP equals SUB

This is something you should know, the CMP instruction is actually a SUB instruction but with that difference the target register isn't changed. That's why you are seeing Jxx instructions right after a SUB instruction.

```
mainloop: ...

CMP EAX, 16

JBE endloop

SUB EAX, 16

JMP mainloop

endloop: ...

becomes:

mainloop: ...

SUB EAX, 16

JA mainloop ; Jump if Above (=JNBE) (Jump if Not Below or Equal)

endloop: ...
```

NOTE: also the TEST instruction is actually an AND instruction (not used here)

6) Remember that the C API and Windows API preserve the EBX, EDI, ESI and EBP registers. Make use of that fact! Never ever trust the contents of ECX or EDX after calling an API. Know that most functions put their return values into EAX and sometimes into the EDX:EAX pair.

Let's take a closer look at the following code inside the main loop:

```
005ED15F 59
                                pop
                                        ecx
005ED160 8D 75 E4
                                        esi, [ebp+var 10]
                                lea
005ED163 8D 04 09
                                lea
                                        eax, [ecx+ecx]
005ED166 8B 7D 18
                                        edi, [ebp+arg 14]
                                mov
005ED169 50
                                push
                                        eax
005ED16A 39 C7
                                        edi, eax
                                cmp
005ED16C 88 26
                                        [esi], ah
                                mov
005ED16E 76 01
                                jbe
                                        short LOCAL Ready2
005ED170 97
                                        eax, edi
                                xchq
005ED171
005ED171
                               LOCAL Ready2:
005ED171 29 CF
                                sub
                                        edi, ecx
005ED173 76 1D
                                        short LOCAL SkipLoop2
                                ibe
005ED175 53
                               push
                                        ebx
005ED176 01 CB
                                add
                                        ebx, ecx
005ED178
                               LOCAL_Loop2:
005ED178
```

If you are looking at that code to understand it, I think you're in hell.

I will help you.

"POP ECX" will put 8 into ECX. It is a constant I pushed on the stack earlier, at address 005ED13B to be exact. ESI points to a string on the stack (Log8_2 in the C Version). EAX is then calculated by adding ECX to itself. EAX = ECX + ECX = 8 + 8 = 16. I could use "MOV EAX, 16" (5 bytes) or the "PUSH / POP" trick (also 3 bytes), but I decided to use the LEA version. Then EAX (16) is pushed on the stack **for later use**, it will be popped into ESI at address 005ED192.

So, we know that ECX is 8 and EAX is 16. Then EDI ([EBP+ arg_14] aka PacketSize) is compared against EAX (16). We know that EAX is 16, so AL is 16 and AH is 00. Thus "MOV [ESI], AH" equals "MOV BYTE PTR [ESI], 0" thus meaning "*Log8_2 = 0;" or "Log8_2[0] = 0;". The "XCHG EAX, EDI" will put the value of 16 into EDI. After this function, when we reach LOCAL_Ready2, the EAX register is no longer used. "SUB EDI, ECX" equals "SUB EDI, 8", the loop will be skipped if EDI was below or equal to 8.

By the way, EBX is pointing to the packet data.

The readable (but binary longer) version of the code above is:

```
MOV ESI, offset Log8_2
MOV EDI, PacketSize
CMP EDI, 16
MOV BYTE PTR [ESI], 0
JBE LOCAL_Ready2
MOV EDI, 16
LOCAL_Ready2:
CMP EDI, 8
JBE LOCAL_SkipLoop2
SUB EDI, 8
PUSH EBX
ADD EBX, 8
```

WHERE TO ADD THE CODE AND TEXT

Looking for empty space in an executable can be difficult but can also be easy if the executable is compiled by following the rules.

You have to open the EXE with the Explorer Suite and view the section headers.

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations	Linenumber	Characteristics
00000178	00000180	00000184	00000188	0000018C	00000190	00000194	00000198	0000019A	0000019C
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	001EC022	00001000	001EC200	00000400	00000000	00000000	0000	0000	60000020
.rdata	00017F68	001EE000	00018000	001EC600	00000000	00000000	0000	0000	40000040
.data	0039ED57	00206000	00037A00	00204600	00000000	00000000	0000	0000	C0000040
.idata	00001826	005A5000	00001A00	0023C000	00000000	00000000	0000	0000	C0000040
UVA_DATA	00003C60	005A7000	00003E00	0023DA00	00000000	00000000	0000	0000	C0000040
rsrc	00091000	005AB000	00091000	00241800	00000000	00000000	0000	0000	40000040

The Raw Size is the number of bytes a certain section takes inside the file. The Virtual Size is the number of bytes in memory. If the Virtual Size is lower than the Raw Size, then it means there is unused space (if the compiler is following the rules!).

So actually, you can see that have quite some unused space inside the UoDemo executable!

After inserting your code inside that space, do not forget to edit the section headers so it will contain accurate values, which is important if you want to add code later again.

This is a screenshot of the modified section headers:

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations	Linenumber	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	001EC200	00001000	001EC200	00000400	00000000	00000000	0000	0000	60000020
.rdata	00018000	001EE000	00018000	001EC600	00000000	00000000	0000	0000	40000040
.data	0039ED57	00206000	00037A00	00204600	00000000	00000000	0000	0000	C0000040
.idata	00001826	005A5000	00001A00	0023C000	00000000	00000000	0000	0000	C0000040
UVA_DATA	00003CD8	005A7000	00003E00	0023DA00	00000000	00000000	0000	0000	C0000040
.rsrc	00091000	005AB000	00091000	00241800	00000000	00000000	0000	0000	40000040

The red squares show the modified virtual sizes.

Note that the ".text" and ".rdata" are now completely full and can no longer grow! If we want to add code in a future patch we either have to either remove code or insert an extra section!