Subject: Re: wwnet
Posted by [EE]pickle-jucer on Fri, 15 Jul 2016 15:32:28 GMT
View Forum Message <> Reply to Message

Okay I've figured it out for the most part, it kind of clicked in my head while I was looking at some documents describing the Quake 3 networking code's delta compression.

I'm fairly certain that the function at 0061BD90 was originally named "PacketManagerClass::Reconstruct_From_Delta" and it reconstructs a packet with the given delta packet patch. Likewise, the function at 0061BB30, which I'm also fairly certain was named "PacketManagerClass::Build_Delta_Packet_Patch" does the opposite of the previous and generates a delta packet patch when given multiple packets.

I don't know if this will be useful to anyone else, but while I was trying to reverse engineer "PacketManagerClass::Reconstruct_From_Delta" I hooked the original function to jump into my code to see if it was working, this is what I ended up with (As far as I can tell with testing, it matches the functionality of the original 1:1):

View Code
```
#include <iostream>
#include <windows.h>
#include <cstdio>
#include <cstdint>
#include <cstring>

// PacketManagerClass::Build_Delta_Packet_Patch
// Original: 0x0061BB30

// PacketManagerClass::Reconstruct_From_Delta
// Original: 0x0061BD90
//


/* NOTES:
 *
 * If the lowest bit of the delta packet header is set (*delta_packet & 1), it signifies what I call QWORD mode for lack of a better name.
 * QWORD mode simply has additional bits in the delta bitmask bytes which tell this function to copy a QWORD (8 bytes) from some position in
 * the srcBuf to the same position in the dstBuf.
 */

int32_t __cdecl my_Reconstruct_From_Delta(uint8_t *srcBuf, uint8_t *dstBuf, uint8_t *delta_packet, int32_t PacketLength, int32_t *delta_bytes_read)
{
 int32_t BitIndex;
 int32_t BitIndex_2;
 uint8_t *delta_byte_data_ptr;
```

```
uint8_t *delta_byte_data_ptr_1;
uint8_t *DiffBytePtr;
int32_t *RelativeByteIndex;
int32_t *CurrentRelativeByteIndex;
int32_t DiffByteCount;
int32_t OrigByteCount;
int32_t RelativeByteIndexes[1024];

// Check for null pointers
if (!srcBuf && !dstBuf && !delta_packet)
{
 return 0;
}

// Check for proper packet length
if (PacketLength > 500)
{
 return 0;
}

// Init some variables.
BitIndex = 0;
BitIndex_2 = 0;
delta_byte_data_ptr = delta_packet + 1;
delta_byte_data_ptr_1 = delta_packet + 1;
DiffByteCount = 0;
OrigByteCount = 0;

if (*delta_packet & 1){
 if (PacketLength - 7 > 0)
 {
  // Get the length needed for delta bitmask bytes
  // This rounds down to the nearest whole number
  uint32_t BitMaskByteCount = PacketLength / 8;

  int32_t currentOffset = 0;
  do
  {
   int8_t isBitSet = ((uint8_t)*delta_byte_data_ptr >> BitIndex++) & 1;
   if (BitIndex == 8)
   {
    ++delta_byte_data_ptr;
    BitIndex = 0;
   }
   if (isBitSet)
   {
    // Copy full QWORD (8 bytes), from
    memcpy(&dstBuf[currentOffset], &srcBuf[currentOffset], 8);
```

```
    OrigByteCount += 8;
   }
   currentOffset += 8;

   --BitMaskByteCount;
  } while (BitMaskByteCount);
 }

 // If there is any remainder of the packet length (because the previous section rounded down)
 if (PacketLength % 8) {
  int8_t isBitSet = ((uint8_t)*delta_byte_data_ptr >> BitIndex++) & 1;
  if (BitIndex == 8)
  {
   ++delta_byte_data_ptr;
   BitIndex = 0;
  }
  if (isBitSet)
  {
   return 0;
  }
 }
}

// This 'for' block marks where diff bytes are and places the indicies in the
// RelativeByteIndexes array.
int32_t targetIndex = 0;
int32_t currentIndex = 0;
for (; currentIndex < PacketLength; )
{
 // Checks for QWORD mode:
 // if the bit is set, it was already copied about so do not check for single byte differences.
 if (*delta_packet & 1) {
  int8_t isBitSet = ((uint8_t)*delta_byte_data_ptr_1 >> BitIndex_2++) & 1;
  if (BitIndex_2 == 8)
  {
   BitIndex_2 = 0;
   ++delta_byte_data_ptr_1;
  }
  if (isBitSet)
  {
   currentIndex += 8;
   continue;
  }
 }


 CurrentRelativeByteIndex = &RelativeByteIndexes[DiffByteCount];
 targetIndex = currentIndex + 8;
```

```
 do
 {
  if (currentIndex >= PacketLength)
   break;
  int8_t isBitSet = ((uint8_t)*delta_byte_data_ptr >> BitIndex++) & 1;
  if (BitIndex == 8)
  {
   ++delta_byte_data_ptr;
   BitIndex = 0;
  }
  if (isBitSet)
  {
   // Bit set:
   // Copy the original byte
   dstBuf[currentIndex] = srcBuf[currentIndex];
   ++OrigByteCount;
  }
  else
  {
   // Bit not set:
   // It uses a diff byte at this position, save the index to the array.
   *CurrentRelativeByteIndex = currentIndex;
   ++CurrentRelativeByteIndex;
   ++DiffByteCount;
  }
  ++currentIndex;
 } while (currentIndex < targetIndex);
 currentIndex = targetIndex;
}

 // Get a pointer to the actual diff byte,
 // if BitIndex > 0 then delta_byte_data_ptr still points to a bitmask byte so go 1 byte further.
 DiffBytePtr = delta_byte_data_ptr;
 if (BitIndex > 0){
  DiffBytePtr = delta_byte_data_ptr + 1;
 }

 // I beleive this would come out to the size of all the delta bytes, delta bitmasks, and the delta
header.
 *delta_bytes_read = DiffByteCount + DiffBytePtr - delta_packet;
 if (DiffByteCount > 0)
 {
  // Get the first relative byte index
  RelativeByteIndex = &RelativeByteIndexes[0];

  // Loop over all of the diff bytes and place at the correct relative position in the dst buffer.
  for (int32_t i = DiffByteCount; i > 0; i--){
   dstBuf[*RelativeByteIndex++] = *DiffBytePtr++;
```

```
  }
 }

 // Returns the amount of diff bytes used + original bytes used
 // This should match the original packet length, if it doesn't
 // then something has gone wrong.
 return DiffByteCount + OrigByteCount;

}


BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved){
 if (fdwReason == DLL_PROCESS_ATTACH){
  // While its not reccomended to do alot in DllMain, this doesn't really need a new thread or
anything.

  void* Original_Reconstruct_From_Delta = (void*)0x0061BD90;
  size_t jmp_instruction_size = 5;

  // Change memory permissions
  DWORD flOldProtect;
  VirtualProtect((void*)Original_Reconstruct_From_Delta, jmp_instruction_size,
PAGE_EXECUTE_READWRITE, &flOldProtect);

  // Make a simple jmp patch/hook
  int32_t relativeOffset = (((int32_t)my_Reconstruct_From_Delta -
(int32_t)Original_Reconstruct_From_Delta) - jmp_instruction_size);
  *(BYTE*)Original_Reconstruct_From_Delta = 0xE9;
  *(int32_t*)((uint32_t)Original_Reconstruct_From_Delta + 1) = relativeOffset;

  // Restore memory permissions
  DWORD UnneededPerms;
  VirtualProtect((void*)Original_Reconstruct_From_Delta, jmp_instruction_size, flOldProtect,
&UnneededPerms);

  MessageBoxA(NULL, "Done hooking PacketManagerClass::Reconstruct_From_Delta!", "Done
hooking", 0);
 }
 return TRUE;
}
```