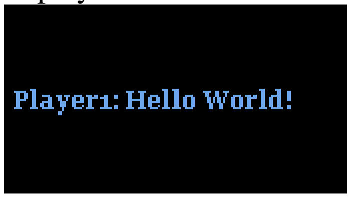
This will be the concluding post of the “Messing with Protocols” series. It will contain some discussion of what was learned and how to mess with the game a bit as a result. The source code provided can be expanded to send any custom chat packets or be used as a starting point in developing a fuzzer. Since the game does not perform integrity checks on parts of the packet such as a valid timer value (this wasn’t discussed but was found while I was reversing recvfrom and onwards), packets can easily be forged by grabbing the session key from any packet. The only field checked is the DWORD value of `06 00 00 00` which was shown to be written in during the building of the chat packet. This means that a custom chat packet can be sent without having to go through the hassle of having to hook the function that increases and writes the timer into the packet (to get the appropriate value if there was a check). This means that writing a custom packet sender is quite easy. The steps would just be: hook sendto to grab the session key and build the packet placing the session key and the `06 00 00 00` bytes in the appropriate offsets. After that, the packet can be filled with whatever data — either garbage data in the case of a fuzzer, or the structure of a legitimate chat packet.
Below is the source to a sample program that can read other players team chat as well as pose as a different player.

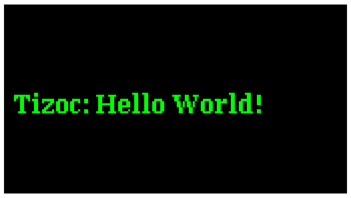
```
#pragma comment(lib, "detours.lib")
#pragma comment(lib, "Ws2_32.lib")

#include <Windows.h>
#include <stdio.h>
#include <include/detours.h>

#define PLAYER_INDEX 20
#define CHAT_FLAG_INDEX 21
#define CHAT_BROADCAST_INDEX 22
#define CHAT_MESSAGE_START_INDEX 37

#define CHAT_FLAG 0xDC

static int (WINAPI *psendto)(SOCKET s, const char *buf, int len, int flags,
const struct sockaddr *to, int tolen) = sendto;
static int (WINAPI *precvfrom)(SOCKET s, char *buf, int len, int flags,
struct sockaddr *from, int *fromlen) = recvfrom;

char *ghost_command = NULL;
char *new_packet_out = NULL;
char *ghost_key = "@ghost";
char *spy_key_on = "@spyon";
char *spy_key_off = "@spyoff";

unsigned char player_to_ghost = 0xFF;
bool is_spy_on = false;

int WINAPI recvfrom_hook(SOCKET s, char *buf, int len, int flags, struct
sockaddr *from, int *fromlen) {
    __asm pushad
    if((buf[CHAT_FLAG_INDEX] & 0xFF) == CHAT_FLAG && is_spy_on == true) {
        memset((buf + CHAT_BROADCAST_INDEX), 0x59, 8);
    }
    int ret = precvfrom(s, buf, len, flags, from, fromlen);
    __asm popad
    return ret;
}

int WINAPI sendto_hook(SOCKET s, const char *buf, int len, int flags, const
struct sockaddr *to, int tolen) {
    __asm pushad
    memcpy(new_packet_out, buf, len);
    if((new_packet_out[CHAT_FLAG_INDEX] & 0xFF) == CHAT_FLAG) {
        if((ghost_command = strstr((new_packet_out +
CHAT_MESSAGE_START_INDEX), ghost_key)) != NULL) {
            player_to_ghost = (ghost_command[strlen(ghost_key)] - 0x30) &
0xFF;
            memset((new_packet_out + CHAT_BROADCAST_INDEX), 0x4E, 8);
        }
        if(strstr((new_packet_out + CHAT_MESSAGE_START_INDEX), spy_key_on) !=
NULL) {
            is_spy_on = true;
            memset((new_packet_out + CHAT_BROADCAST_INDEX), 0x4E, 8);
        }
        else if(strstr((new_packet_out + CHAT_MESSAGE_START_INDEX),
spy_key_off) != NULL) {
            is_spy_on = false;
            memset((new_packet_out + CHAT_BROADCAST_INDEX), 0x4E, 8);
        }
        if(player_to_ghost == 0x00 || player_to_ghost > 0x8)
            new_packet_out[PLAYER_INDEX] = 0xF;
        else
        {
            new_packet_out[PLAYER_INDEX] = player_to_ghost;
            new_packet_out[CHAT_BROADCAST_INDEX + (player_to_ghost - 1)] =
0x4E;
        }
    }
    int ret = psendto(s, new_packet_out, len, flags, to, tolen);
    __asm popad
    return ret;
}

int APIENTRY DllMain(HMODULE hModule, DWORD reason, LPVOID reserved){
    if(reason == DLL_PROCESS_ATTACH) {
        DisableThreadLibraryCalls(hModule);
        new_packet_out = (char *)malloc(256 * sizeof(char));
        (void)DetourTransactionBegin();
        (void)DetourUpdateThread(GetCurrentThread());
        (void)DetourAttach(&(PVOID&)psendto, sendto_hook);
        (void)DetourAttach(&(PVOID&)precvfrom, recvfrom_hook);
        (void)DetourTransactionCommit();
    }
    return TRUE;
}
```

The sample takes in three commands provided through chat, @ghost to imitate a player, @spy_on to enable the ability enemy team chat, and @spy_off to disable it. These all work by replacing outgoing or incoming packets. Chat ghosting works through changing the index of the player sending the chat in outgoing packets. The chat spying works by setting the flags on incoming packets to display in the client. The usage is shown below:



Player1: Hello World!

The chat from the impersonators perspective, who is impersonating player 3.



Tizoc: Hello World!

The chat visible to other players.

After doing all of the reversing, I actually stumbled across a great article which explains the networking code behind the Age of Empire series and provides exclamations into what the counters mean and the general architecture of the protocol.
A downloadble PDF of this post can be found here.