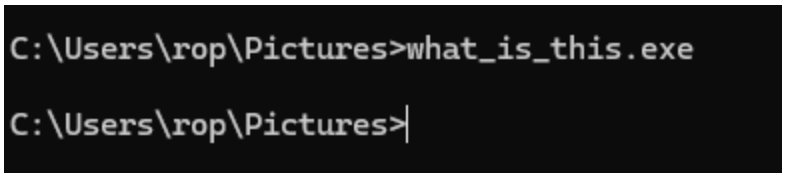
Название:	`Что это такое?`
Категория:	`Реверс`
Уровень:	`Средний`
Очки:	`400`
Описание:	`Сможешь ли ты узнать, что делает этот файл? Да причём так, чтобы получить флаг :)`
Теги:	`C++, дешифрование кода, работа с реестром`
Автор:	`ROP`

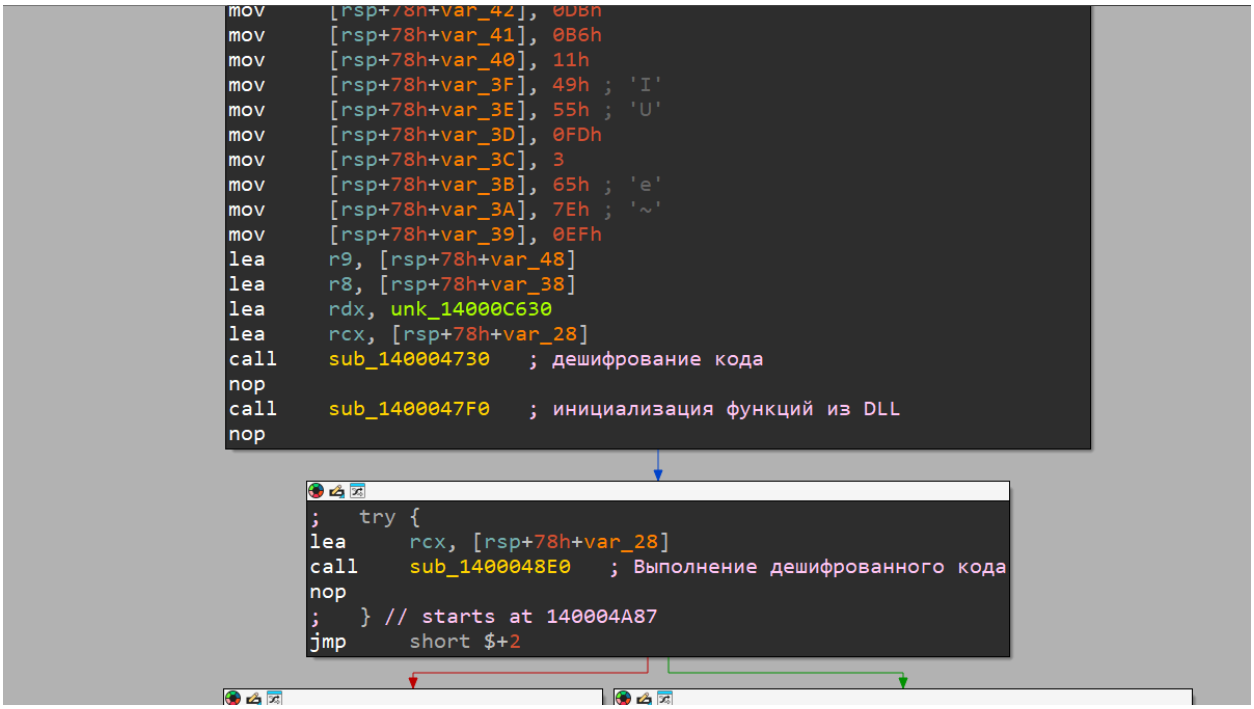
Прохождение:

Нам дан файл на C++.

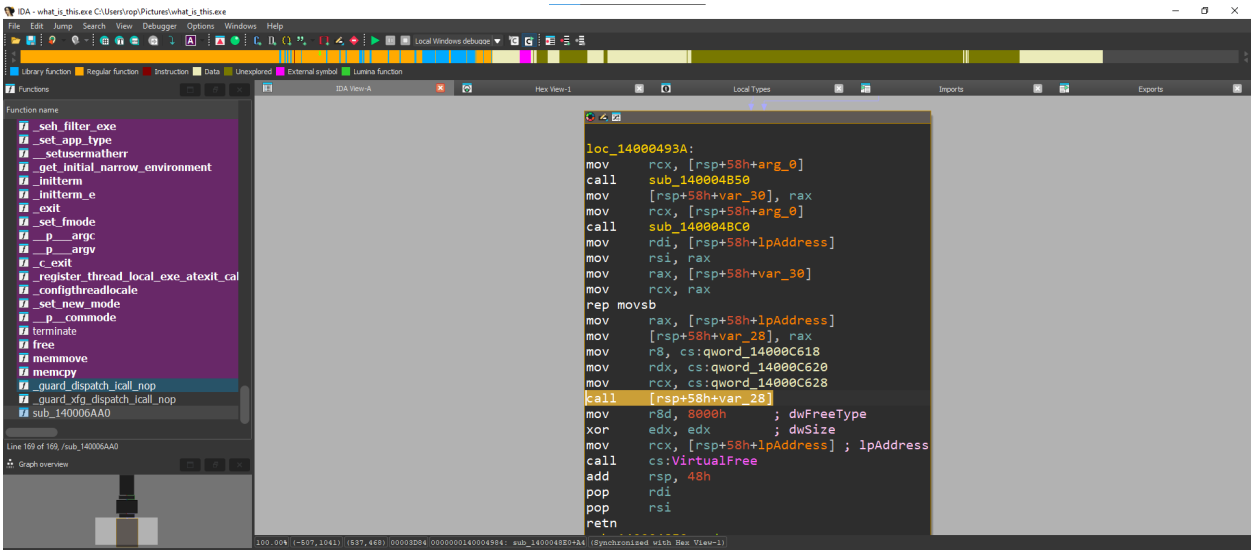


Он ничего не делает.

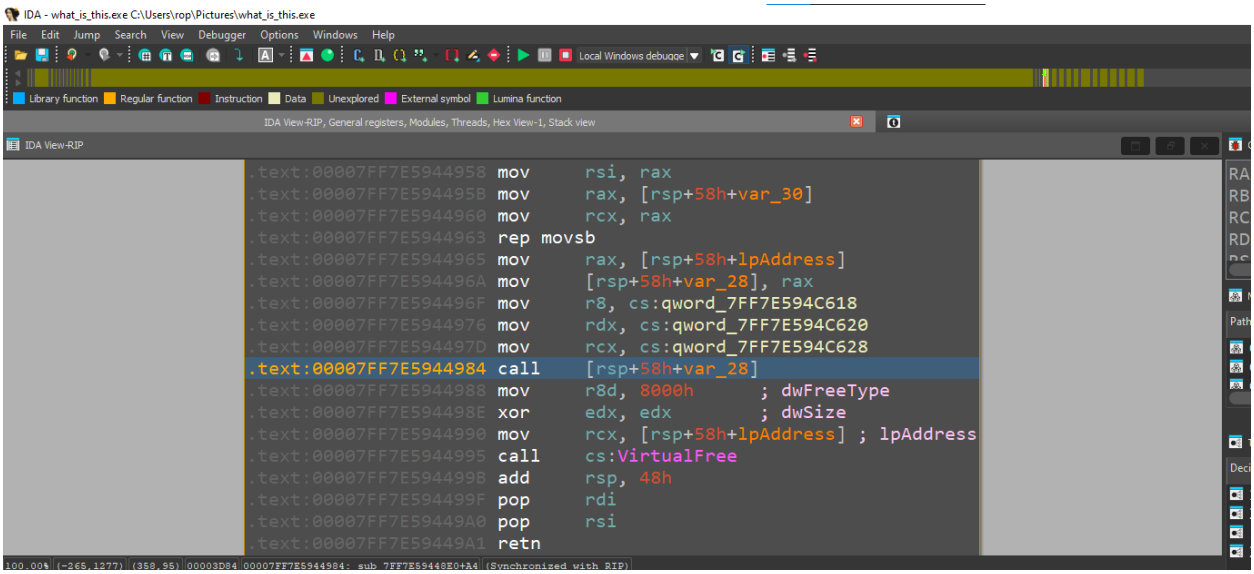
Изучаем в IDA.



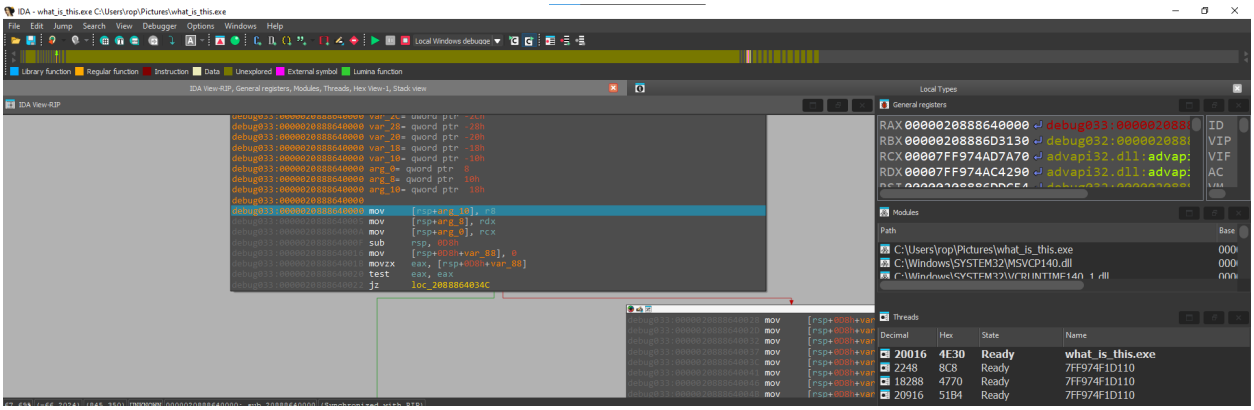
В ходе тестов узнаем, что последовательность такая. Зайдём в функцию, что выполнит код.

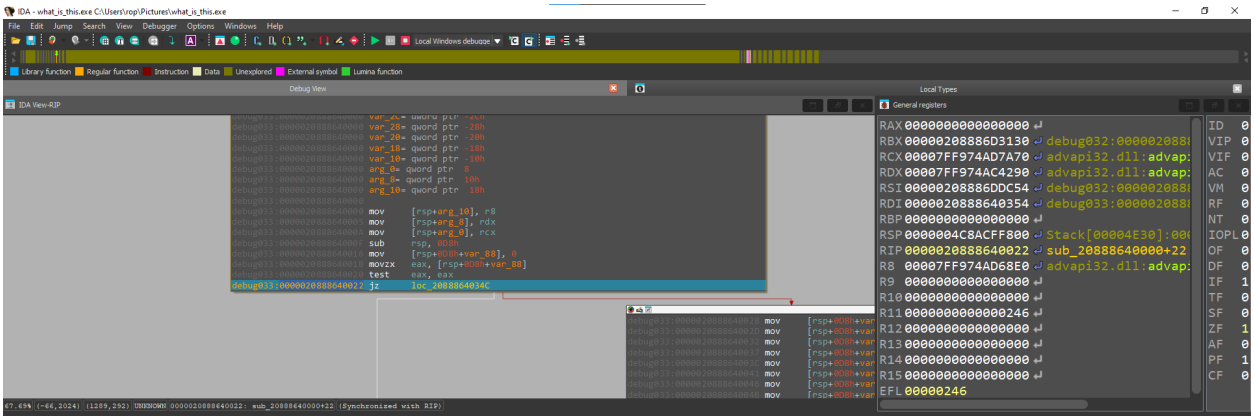


Во время отладки нужно перейти в это место.

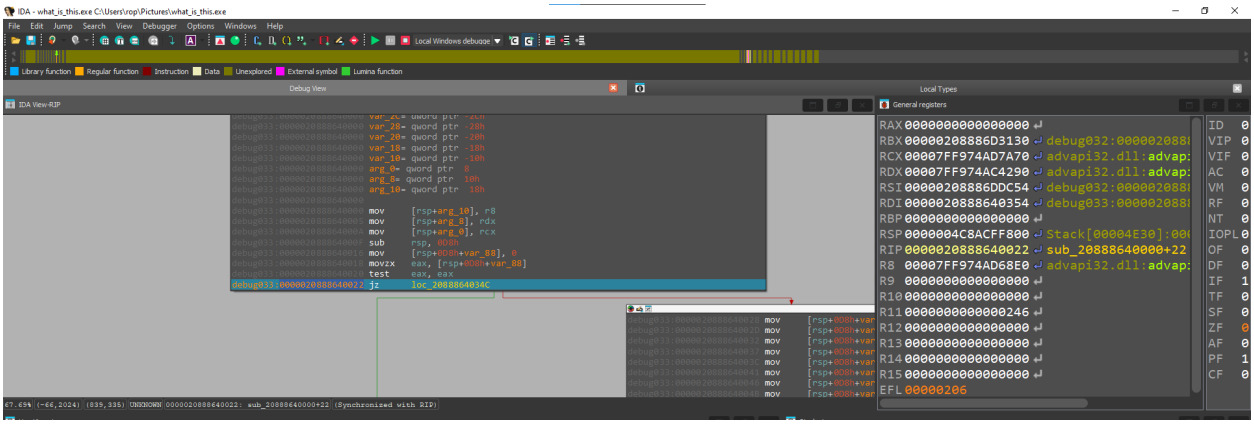

IDA - what_is_this.exe C:\Users\rop\Pictures\what_is_this.exe
File Edit Jump Search View Debugger Options Windows Help
Local Windows debugge
Library function Regular function Instruction Data Unexplored External symbol Lumina function
IDA View-RIP, General registers, Modules, Threads, Hex View-1, Stack view
IDA View-RIP
.text:00007FF7E5944958 mov rsi, rax
.text:00007FF7E594495B mov rax, [rsp+58h+var_30]
.text:00007FF7E5944960 mov rcx, rax
.text:00007FF7E5944963 rep movsb
.text:00007FF7E5944965 mov rax, [rsp+58h+lpAddress]
.text:00007FF7E594496A mov [rsp+58h+var_28], rax
.text:00007FF7E594496F mov r8, cs:qword_7FF7E594C618
.text:00007FF7E5944976 mov rdx, cs:qword_7FF7E594C620
.text:00007FF7E594497D mov rcx, cs:qword_7FF7E594C628
.text:00007FF7E5944984 call [rsp+58h+var_28]
.text:00007FF7E5944988 mov r8d, 8000h ; dwFreeType
.text:00007FF7E594498E xor edx, edx ; dwSize
.text:00007FF7E5944990 mov rcx, [rsp+58h+lpAddress] ; lpAddress
.text:00007FF7E5944995 call cs:VirtualFree
.text:00007FF7E594499B add rsp, 48h
.text:00007FF7E594499F pop rdi
.text:00007FF7E59449A0 pop rsi
.text:00007FF7E59449A1 retn

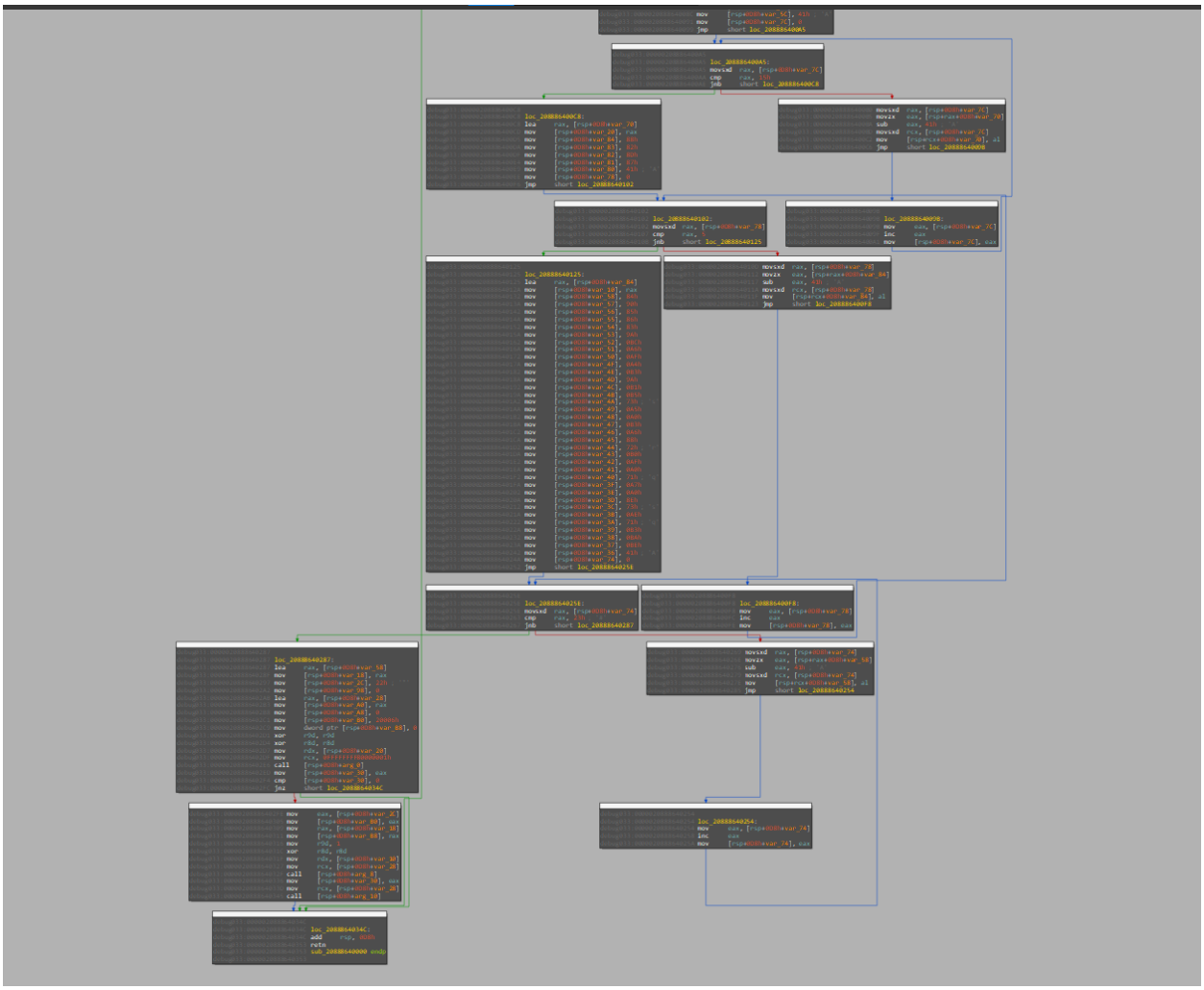
Жмём F7.


IDA - what_is_this.exe C:\Users\rop\Pictures\what_is_this.exe
Local Types
General registers
RAX 0000020888640000
RBX 000002088 6D3130
RCX 00007FF974AD7A70
RDX 00007FF974AC4290
Modules
Path
C:\Users\rop\Pictures\what_is_this.exe
C:\Windows\SYSTEM32\MSVCP140.dll
Threads
Decimal Hex State Name
20016 4E30 Ready what_is_this.exe
2248 8C8 Ready 7FF974F1D110
18288 4770 Ready 7FF974F1D110
20916 51B4 Ready 7FF974F1D110


IDA - what_is_this.exe C:\Users\rop\Pictures\what_is_this.exe
Debug View
Local Types
General registers
RAX 0000000000000000
RBX 000002088 6D3130
RCX 00007FF974AD7A70
RDX 00007FF974AC4290
RSI 000002088 6DDC54
RDI 0000020888640354
RBP 0000000000000000
RSP 0000004C8ACFF800
RIP 0000020888640022 sub_20888640000+22
R8 00007FF974AD68E0
R9 0000000000000000
R10 0000000000000000
R11 0000000000000246
R12 0000000000000000
R13 0000000000000000
R14 0000000000000000
R15 0000000000000000
EFL 00000246

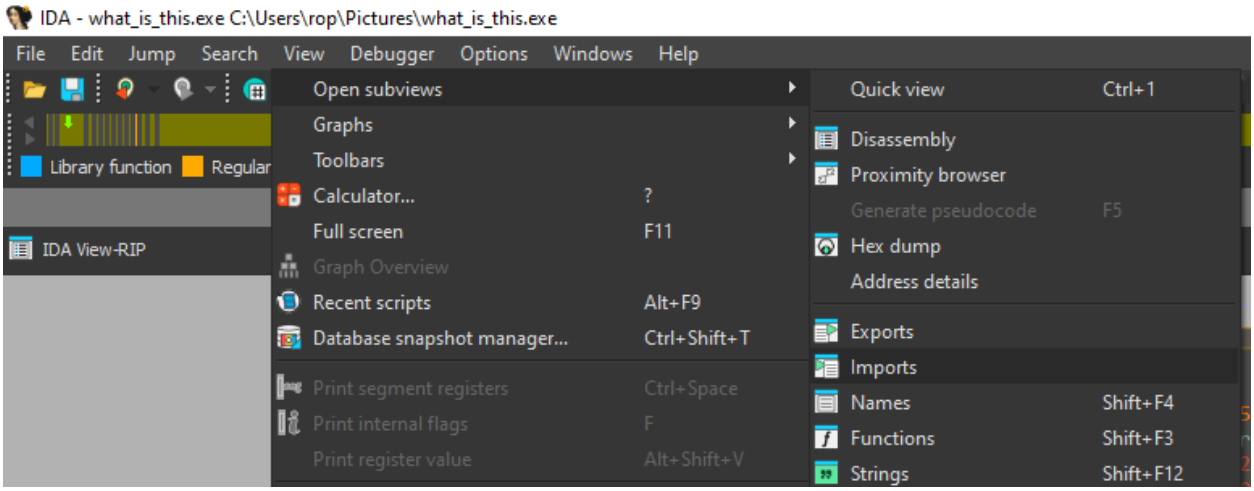
Тут в ходе тестов понимаем, что нужно поменять ZF. Иначе код не выполнится.

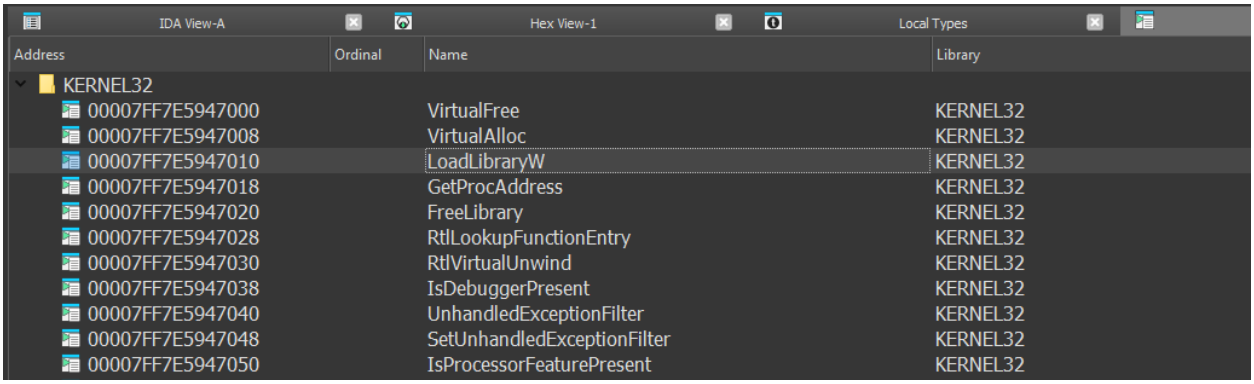




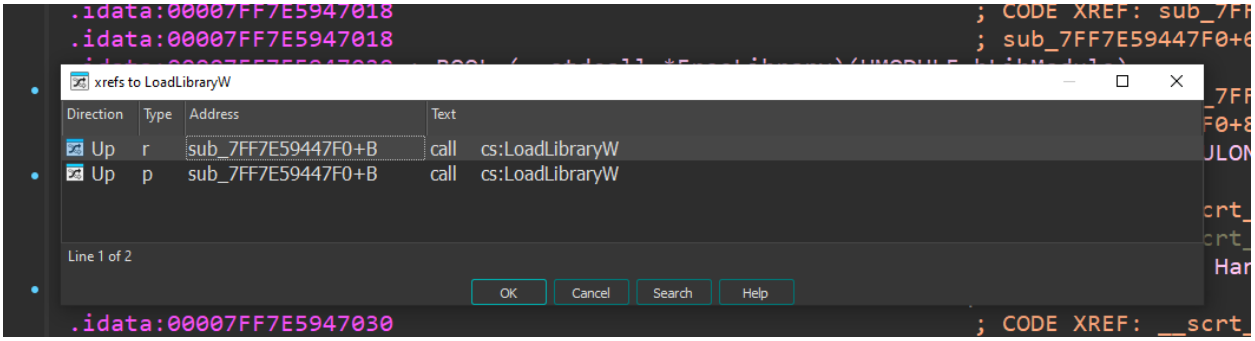
Видим дешифровку массивов и выполнение функций, переданных в дешифрованный код.

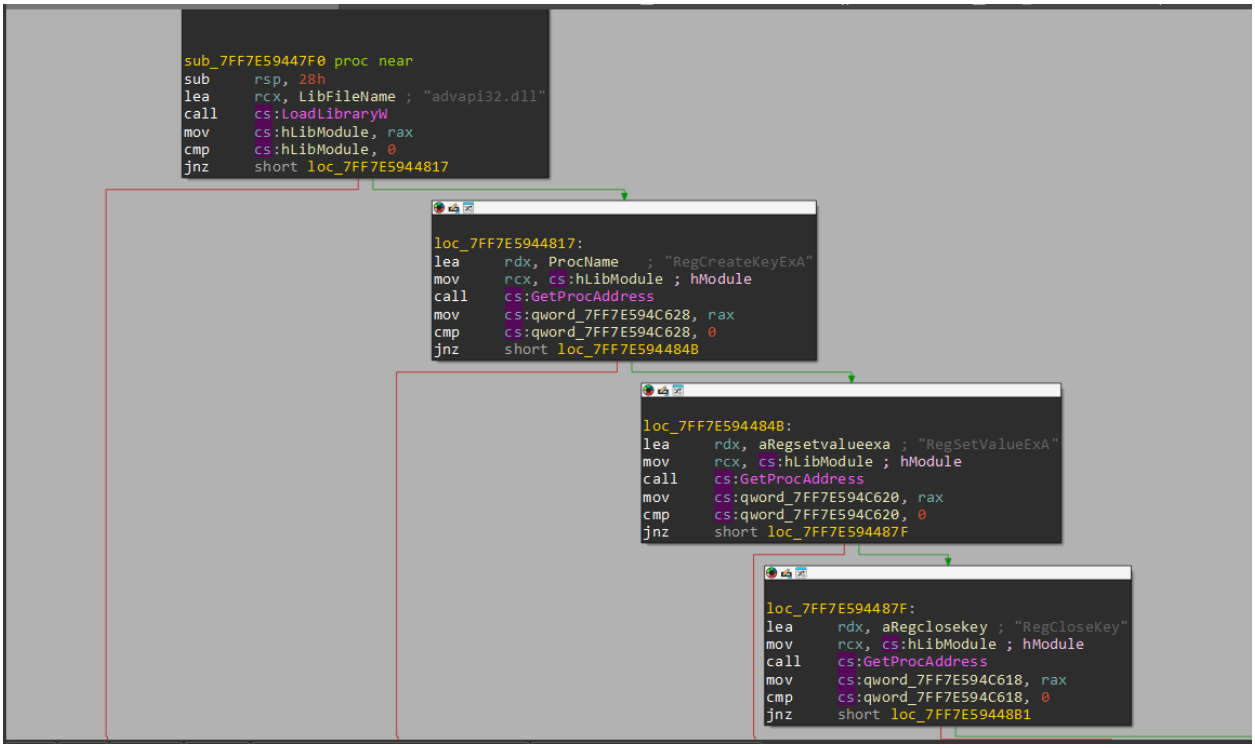
Какие именно это функции можно найти в перекрёстных ссылках в этом окне.





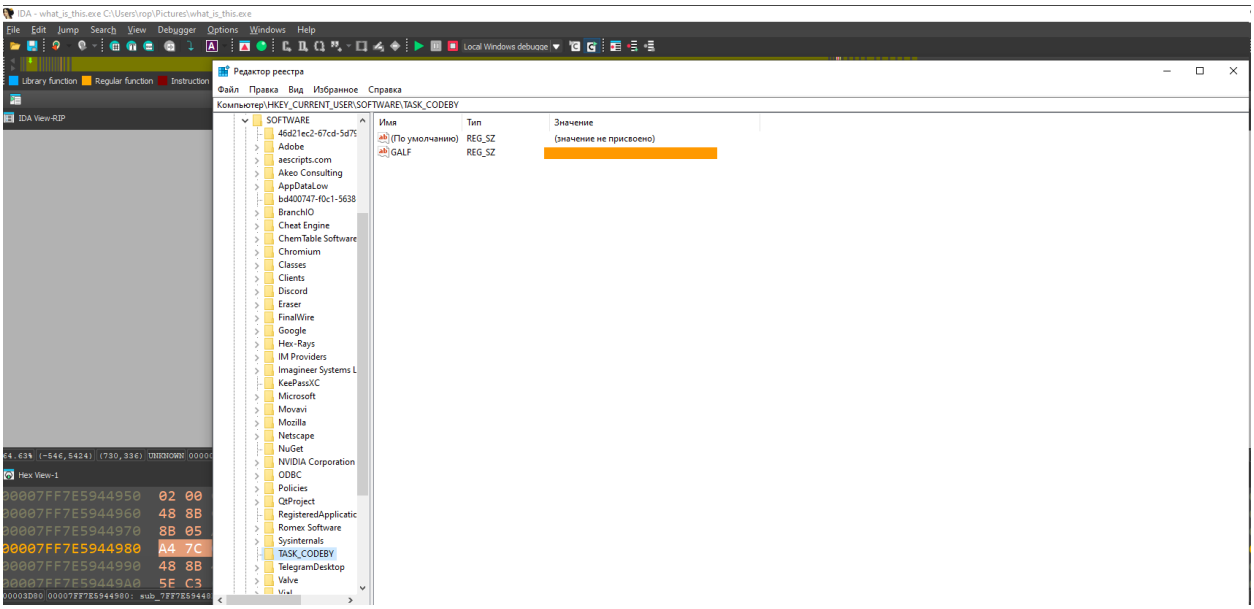
Комбинация из функций LoadLibrary, VirtualFree, VirtualAlloc, FreeLibrary, GetProcAddress уже настораживает. Посмотрим, где используется LoadLibrary. Нажмём 2 раза на неё, затем X.





Видим, какие функции будут вызваны в дешифрованном коде.

Можем выполнить код, он создаст запись в реестре и тат будет флаг.



В строках таска есть пасхалки :)

Исходник для понимания:

```
#include <windows.h>
#include <iostream>
#include <vector>
#include <iomanip> // Для std::hex и std::setfill

extern "C"
{
#include "aes.h"
}

#pragma comment(lib, "user32.lib")

#define _CRT_SECURE_NO_DEPRECATE

unsigned char hmstr[] = {0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x2(
```

```
char flag[] = "CODEBY <- remember, this is where the flag starts
std::vector<uint8_t> encrypted_data = { 0xde, 0x12, 0x7a, 0x9c,

/*void static addPKCS7Padding(std::vector<uint8_t>& buffer, size
    size_t padding_length = block_size - (buffer.size() % block_
    buffer.insert(buffer.end(), padding_length, static_cast<uint
}

std::vector<uint8_t> encryptFunction(const std::vector<uint8_t>&
    struct AES_ctx ctx;
    AES_init_ctx_iv(&ctx, key, iv);

    std::vector<uint8_t> encrypted_data = func_data;

    // Добавляем PKCS7 padding к данным функции
    addPKCS7Padding(encrypted_data, 16);

    // Шифруем данные функции
    AES_CBC_encrypt_buffer(&ctx, encrypted_data.data(), encrypte

    return encrypted_data;
}*/

void removePKCS7Padding(std::vector<uint8_t>& data) {
    hmstr[0] = 0x20;
    flag[0] = 'C';
    if (data.empty()) {
        exit(0);
    }

    uint8_t padding_length = data.back();
    if (padding_length > data.size() || padding_length == 0) {
        exit(0);
    }

    for (size_t i = data.size() - padding_length; i < data.size(
```

```
        if (data[i] != padding_length) {
            exit(0);
        }
    }

    data.resize(data.size() - padding_length);
}

std::vector<uint8_t> decryptFunction(const std::vector<uint8_t>&
    struct AES_ctx ctx;
    AES_init_ctx_iv(&ctx, key, iv);

    std::vector<uint8_t> decrypted_data = encrypted_data;

    // Расшифровываем данные функции
    AES_CBC_decrypt_buffer(&ctx, decrypted_data.data(), decrypte

    // Удаляем PKCS7 padding из данных функции
    removePKCS7Padding(decrypted_data);

    return decrypted_data;
}

/*size_t getFunctionLength(const uint8_t* func) {
    size_t length = 0;
    while (func[length] != 0xC3) {
        length++;
    }
    return length + 1; // Учитываем байт 0xC3
}

void* getRealFunctionAddress(void* func) {
    uint8_t* codePtr = reinterpret_cast<uint8_t*>(func);

    // Проверяем, является ли это JMP инструкцией
    if (codePtr[0] == 0xE9) { // 0xE9 - код для JMP (near jump)
```

```
        int32_t offset = *reinterpret_cast<int32_t*>(codePtr + 1
        uint8_t* realAddress = codePtr + 5 + offset;
        return realAddress;
    }

    // Если это не JMP, возвращаем оригинальный адрес
    return func;
}*/

// Тип функции MessageBox
typedef int (WINAPI* MessageBoxW_t)(HWND, LPCWSTR, LPCWSTR, UINT
/*typedef int (WINAPI* p_RegCreateKeyExA) (
    _In_ HKEY hKey,
    _In_ LPCSTR lpSubKey,
    _Reserved_ DWORD Reserved,
    _In_opt_ LPSTR lpClass,
    _In_ DWORD dwOptions,
    _In_ REGSAM samDesired,
    _In_opt_ CONST LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    _Out_ PHKEY phkResult,
    _Out_opt_ LPDWORD lpdwDisposition
    );
typedef int (WINAPI* p_RegSetValueExA) (
    _In_ HKEY hKey,
    _In_opt_ LPCSTR lpValueName,
    _Reserved_ DWORD Reserved,
    _In_ DWORD dwType,
    _In_reads_bytes_opt_(cbData) CONST BYTE* lpData,
    _In_ DWORD cbData
    );
typedef int (WINAPI* p_RegCloseKey) (
    _In_ HKEY hKey
    );*/

typedef LONG(WINAPI* p_RegCreateKeyExA)(HKEY, LPCSTR, DWORD, LPS
typedef LONG(WINAPI* p_RegSetValueExA)(HKEY, LPCSTR, DWORD, DWOR
```

```
typedef LONG(WINAPI* p_RegCloseKey)(HKEY);

HMODULE hAdvApi32;
MessageBoxW_t pMessageBoxW;
p_RegCreateKeyExA dp_RegCreateKeyExA;
p_RegSetValueExA dp_RegSetValueExA;
p_RegCloseKey dp_RegCloseKey;

void init_close_dynamics() {
    // Загрузка библиотеки user32.dll
    hAdvApi32 = LoadLibraryW(L"advapi32.dll");
    if (hAdvApi32 == NULL) {
        // Обработка ошибки загрузки библиотеки
        return;
    }

    // Получение адреса функции RegCreateKeyExA
    dp_RegCreateKeyExA = (p_RegCreateKeyExA)GetProcAddress(hAdv
    if (dp_RegCreateKeyExA == NULL) {
        // Обработка ошибки получения адреса функции
        FreeLibrary(hAdvApi32);
        return;
    }

    // Получение адреса функции RegSetValueExA
    dp_RegSetValueExA = (p_RegSetValueExA)GetProcAddress(hAdvAp
    if (dp_RegSetValueExA == NULL) {
        // Обработка ошибки получения адреса функции
        FreeLibrary(hAdvApi32);
        return;
    }

    // Получение адреса функции RegCloseKey
    dp_RegCloseKey = (p_RegCloseKey)GetProcAddress(hAdvApi32, "R
    if (dp_RegCloseKey == NULL) {
        // Обработка ошибки получения адреса функции
```

```
        FreeLibrary(hAdvApi32);
        return;
    }

}

void disinit_close_dynamics() {
    FreeLibrary(hAdvApi32);
}

/*void func_reg(p_RegCreateKeyExA RegCreateKeyE, p_RegSetValueEx

    bool flg = 0;

    if (flg) {
        unsigned char path[] = { 0x94,0x90,0x87,0x95,0x98,0x82,0

        for (int i = 0; i < sizeof(path); i++) {
            path[i] -= 0x41;
        }

        LPCSTR subkey = (LPCSTR)path;

        unsigned char value[] = { 0x88,0x82,0x8d,0x87,0x41 };

        for (int i = 0; i < sizeof(value); i++) {
            value[i] -= 0x41;
        }

        LPCSTR valueName = (LPCSTR)value;

        unsigned char key_data[] = { 0x84,0x90,0x85,0x86,0x83,0x

        for (int i = 0; i < sizeof(key_data); i++) {
            key_data[i] -= 0x41;
        }
```

```
        LPCSTR data = (LPCSTR)key_data;

        int data_size = sizeof(key_data) - 1;

        HKEY hKey;

        // Открытие или создание ключа реестра
        LONG result = RegCreateKeyE(
            HKEY_CURRENT_USER,  // Основной ключ
            subkey,             // Подключ
            0,                  // Зарезервировано, должно быть
            NULL,               // Класс, обычно NULL
            REG_OPTION_NON_VOLATILE, // Параметры
            KEY_WRITE,          // Права доступа
            NULL,               // Без защиты
            &hKey,              // Указатель на созданный/откры
            NULL                // Указатель на переменную, полу
        );

        if (result == ERROR_SUCCESS) {
            // Установка значения ключа
            result = RegSetValueE(
                hKey,           // Открытый ключ
                valueName,      // Название значения
                0,              // Зарезервировано, должно быть
                REG_SZ,         // Тип данных
                (const BYTE*)data, // Данные для записи
                data_size // Размер данных (включая завершающий
            );

            // Закрытие ключа
            RegCloseK(hKey);
        }
    }
}*/
```

```
typedef void (*func_t)(p_RegCreateKeyExA, p_RegSetValueExA, p_Re

void executeFunctionFromBytes(const std::vector<uint8_t>& code)
    if (code.empty()) {
        exit(0);
    }

    // Выделение исполняемой памяти
    void* exec_mem = VirtualAlloc(nullptr, code.size(), MEM_COMM
    if (!exec_mem) {
        exit(0);
    }

    // Копирование байт кода в выделенную память
    std::memcpy(exec_mem, code.data(), code.size());

    // Приведение указателя на память к типу функции
    func_t func = reinterpret_cast<func_t>(exec_mem);

    // Вызов функции
    func(dp_RegCreateKeyExA, dp_RegSetValueExA, dp_RegCloseKey),

    //func_reg2(subkey, valueName, data, strlen(data));

    // Освобождение памяти
    VirtualFree(exec_mem, 0, MEM_RELEASE);
}

int main() {
    //void* realFuncAddress = getRealFunctionAddress(reinterpret
    //uint8_t* func_bytes = reinterpret_cast<uint8_t*>(realFuncA

    //size_t func_size = getFunctionLength(reinterpret_cast<cons
    size_t func_size = encrypted_data.size();
    //std::vector<uint8_t> func_data(func_bytes, func_bytes + fu
```

```
    // Определяем ключ и вектор инициализации
    const unsigned char key[16] = {
        0x88, 0x23, 0x3B, 0xEF, 0x62, 0x16, 0x11, 0x00, 0xE2, 0x
        0x7D, 0xE8, 0x73, 0xE7
    };

    const unsigned char iv[16] = {
        0x40, 0x42, 0x31, 0xBC, 0xFF, 0xAA, 0xDB, 0xB6, 0x11, 0x
        0x03, 0x65, 0x7E, 0xEF
    };

    // Шифруем функцию
    //std::vector<uint8_t> encrypted_data = encryptFunction(func
    std::vector<uint8_t> deccrypted_data = decryptFunction(encry

    init_close_dynamics();

    /*std::cout << "Encrypted data: ";
    for (size_t i = 0; i < encrypted_data.size(); ++i) {
        std::cout << "0x" << std::hex << std::setw(2) << std::se
        if (i < encrypted_data.size() - 1) {
            std::cout << ", ";
        }
    }
    std::cout << std::endl;*/


    try {
        executeFunctionFromBytes(deccrypted_data);
    }
    catch (const std::exception& e) {
        //std::cerr << "Exception: " << e.what() << std::endl;
        exit(0);
    }
    disinit_close_dynamics();
```

```
    return 0;
}
```