



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++.

```
C:\Users\rop\Pictures>what_is_this.exe  
C:\Users\rop\Pictures>
```

Он ничего не делает.

Изучаем в IDA.

```

mov    [rsp+8n+var_42], 0B6h
mov    [rsp+78h+var_41], 0B6h
mov    [rsp+78h+var_40], 11h
mov    [rsp+78h+var_3F], 49h ; 'I'
mov    [rsp+78h+var_3E], 55h ; 'U'
mov    [rsp+78h+var_3D], 0FDh
mov    [rsp+78h+var_3C], 3
mov    [rsp+78h+var_3B], 65h ; 'e'
mov    [rsp+78h+var_3A], 7Eh ; '~'
mov    [rsp+78h+var_39], 0EEh
lea    r9, [rsp+78h+var_48]
lea    r8, [rsp+78h+var_38]
lea    rdx, unk_14000C630
lea    rcx, [rsp+78h+var_28]
call   sub_140004730 ; дешифрование кода
nop
call   sub_1400047F0 ; инициализация функций из DLL
nop

```

↓

```

; try {
lea    rcx, [rsp+78h+var_28]
call   sub_1400048E0 ; Выполнение дешифрованного кода
nop
; } // starts at 140004A87
jmp   short $+2

```

↓ ↓ ↓

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

```

Function name
#_set_filter_exe
#_set_app_type
#_setusermatherr
#_get_initial_narrow_environment
#_lnterm
#_lnterm_e
#_exit
#_set_fmode
#_p_argc
#_p_argv
#_c_exit
#_register_thread_local_exe_atexit_call
#_configthreadlocale
#_set_new_mode
#_p_commode
#_terminate
#_free
#_memmove
#_memcpy
#_guard_dispatch_icall_nop
#_guard_xfg_dispatch_icall_nop
#_sub_140006AA0

Line 169 of 169, #sub_140006AA0
Graph overview

```

```

loc_14000493A:
mov    rcx, [rsp+58h+arg_0]
call   sub_140004B50
mov    [rsp+58h+var_30], rax
mov    rcx, [rsp+58h+arg_0]
call   sub_140004BC0
mov    rdi, [rsp+58h+lpAddress]
mov    rsi, rax
mov    rax, [rsp+58h+var_30]
mov    rcx, rax
rep    movsb
mov    rax, [rsp+58h+lpAddress]
mov    [rsp+58h+var_28], rax
mov    r8, cs:qword_14000C618
mov    rdx, cs:qword_14000C620
mov    rcx, cs:qword_14000C628
call   [rsp+58h+var_28]
mov    r8d, 800h ; dwFreeType
xor    edx, edx ; dwSize
mov    rcx, [rsp+58h+lpAddress] ; lpAddress
call   cs:VirtualFree
add    rsp, 48h
pop    rdi
pop    rsi
ret

```

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 debugger
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
.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

```

100.0% (-265, 1277) (358, 95) | 00003D84 00007FF7E5944984: sub 7FF7E59448E0+A4 [Synchronized with RIP]

Жмём F7.

IDA - what_is_this.exe C:\Users\rop\Pictures\what_is_this.exe

```

File Edit Jump Search View Debugger Options Windows Help
Local Windows debugger
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
.debug$3.000020888640000 var_28h quword ptr -28h
.debug$3.000020888640000 var_20h quword ptr -20h
.debug$3.000020888640000 var_18h quword ptr -18h
.debug$3.000020888640000 var_10h quword ptr -10h
.debug$3.000020888640000 arg_8 quword ptr 8
.debug$3.000020888640000 arg_10h quword ptr 10h
.debug$3.000020888640000 mov [rsp+arg_10], r8
.debug$3.000020888640000 mov [rsp+arg_8], rdx
.debug$3.000020888640000 mov [rsp+arg_0], rcx
.debug$3.000020888640000 sub [rsp+58h+var_88], 0
.debug$3.000020888640000 movzx eax, [rsp+58h+var_88]
.debug$3.000020888640000 test eax, eax
.jz loc_20888640834c

```

Threads

ID	Base	Name
0000000000000000	000	what_is_this.exe
0000000000000000	000	C:\Windows\SYSTEM32\MSVCP140.dll
0000000000000000	NNN	C:\Windows\SYSTEM32\VCRIINTTM140.dll

IDA - what_is_this.exe C:\Users\rop\Pictures\what_is_this.exe

```

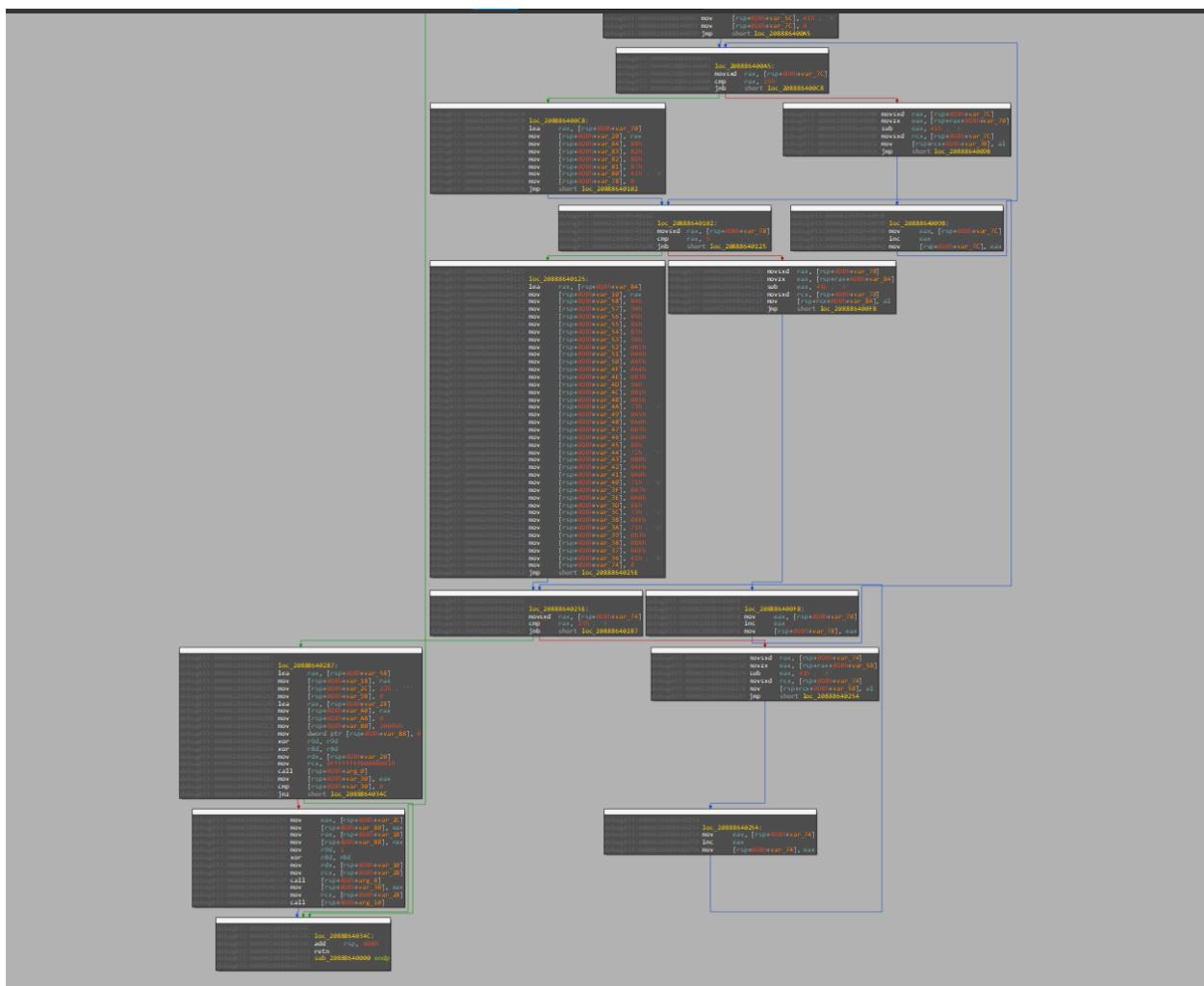
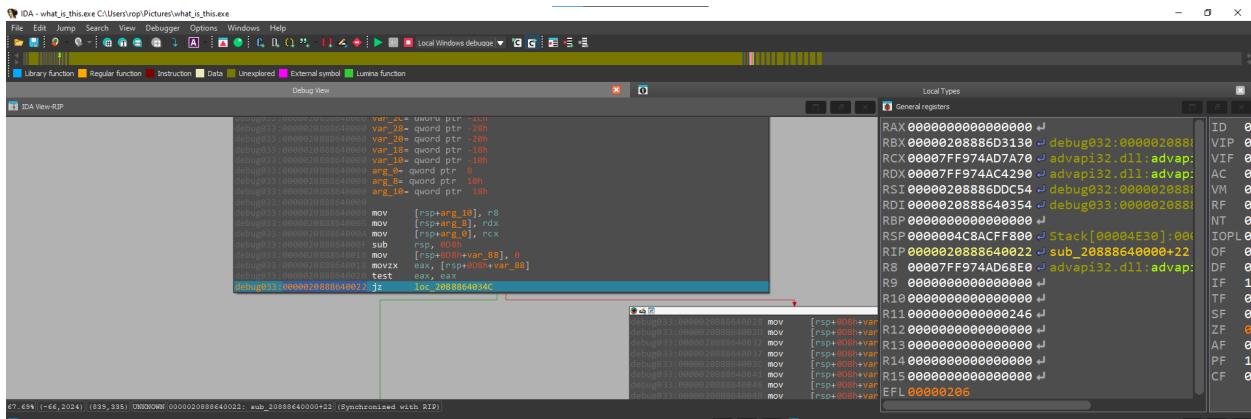
File Edit Jump Search View Debugger Options Windows Help
Local Windows debugger
Library function Regular function Instruction Data Unexplored External symbol Lumina function
IDA View-RIP, General registers, Modules, Threads, Hex View-1, Stack view
IDA View-RIP
.debug$3.000020888640000 var_28h quword ptr -28h
.debug$3.000020888640000 var_20h quword ptr -20h
.debug$3.000020888640000 var_18h quword ptr -18h
.debug$3.000020888640000 var_10h quword ptr -10h
.debug$3.000020888640000 arg_8 quword ptr 8
.debug$3.000020888640000 arg_10h quword ptr 10h
.debug$3.000020888640000 mov [rsp+arg_10], r8
.debug$3.000020888640000 mov [rsp+arg_8], rdx
.debug$3.000020888640000 mov [rsp+arg_0], rcx
.debug$3.000020888640000 sub [rsp+58h+var_88], 0
.debug$3.000020888640000 movzx eax, [rsp+58h+var_88]
.debug$3.000020888640000 test eax, eax
.debug$3.000020888640022 jz loc_20888640834c

```

Registers

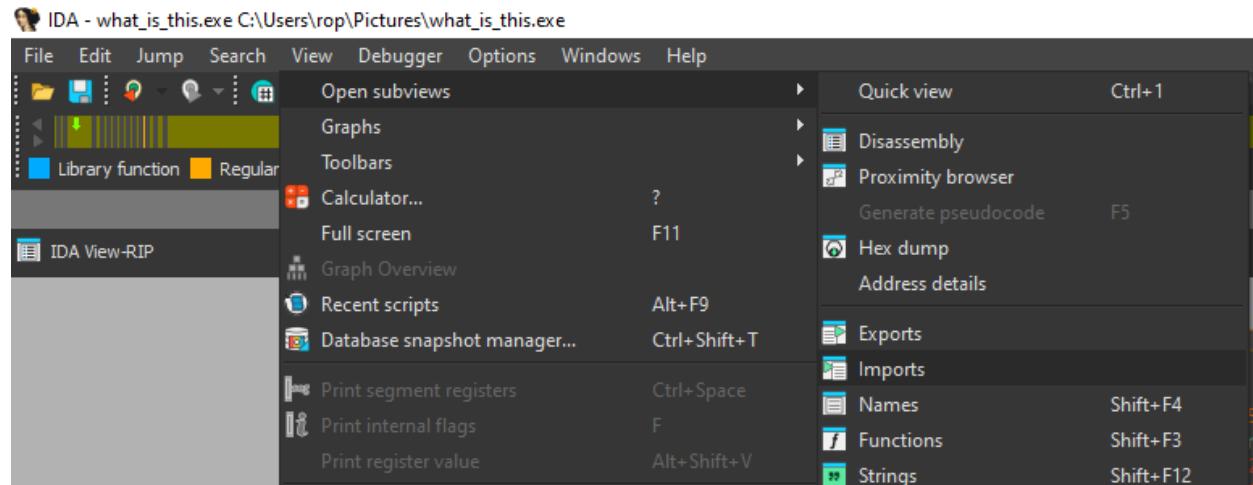
ID	Value	State
RAX	0000000000000000	debug\$3:000002088640000
RBX	0000000000000000	debug\$3:000002088640000
RCX	0000000000000000	debug\$3:000002088640000
RDX	0000000000000000	debug\$3:000002088640000
R8	0000000000000000	debug\$3:000002088640000
R9	0000000000000000	debug\$3:000002088640000
R10	0000000000000000	debug\$3:000002088640000
R11	0000000000000000	debug\$3:000002088640000
R12	0000000000000000	debug\$3:000002088640000
R13	0000000000000000	debug\$3:000002088640000
R14	0000000000000000	debug\$3:000002088640000
R15	0000000000000000	debug\$3:000002088640000
RF	0	
RBP	0000000000000000	debug\$3:000002088640000
RSP	0000000048ACFF800	Stack[00004E30]:000
RIP	0000000020888640022	sub_20888640000+22
OF	0	
DF	0	
IF	1	
SF	0	
ZF	1	
AF	0	
PF	1	
CF	0	

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



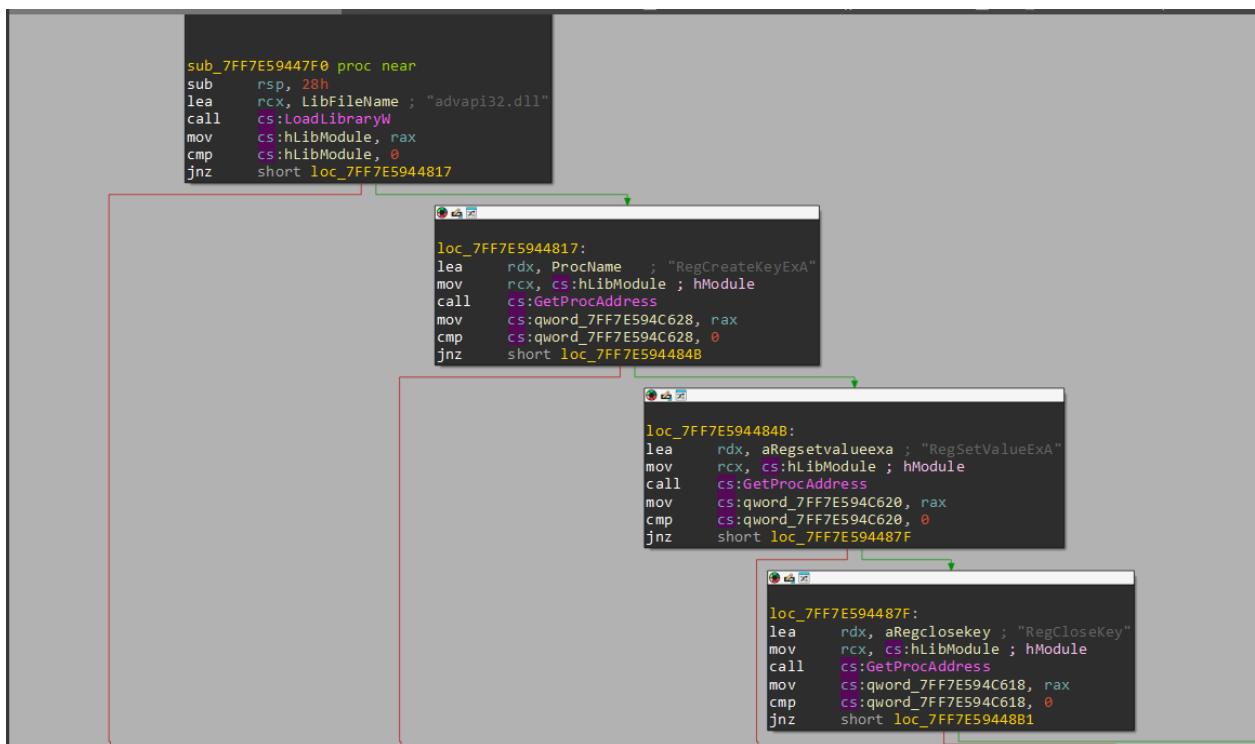
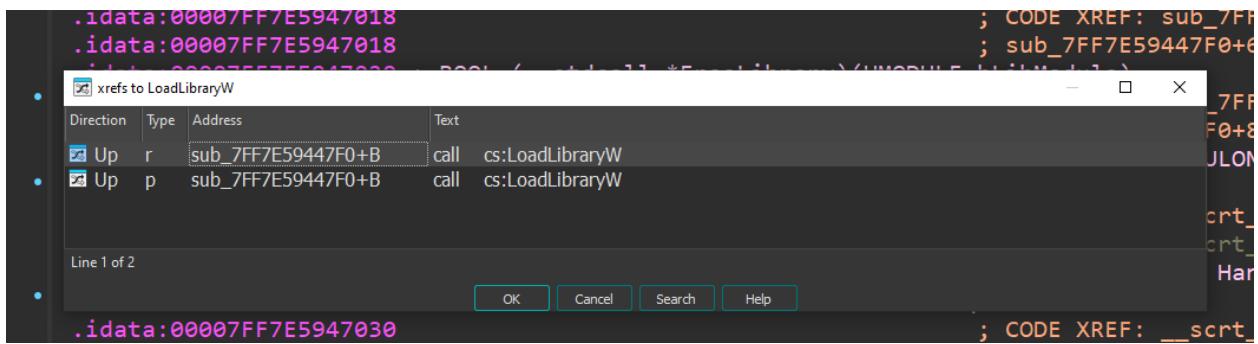
Видим дешифровку массивов и выполнение функций, переданных в дешифрованный код.

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



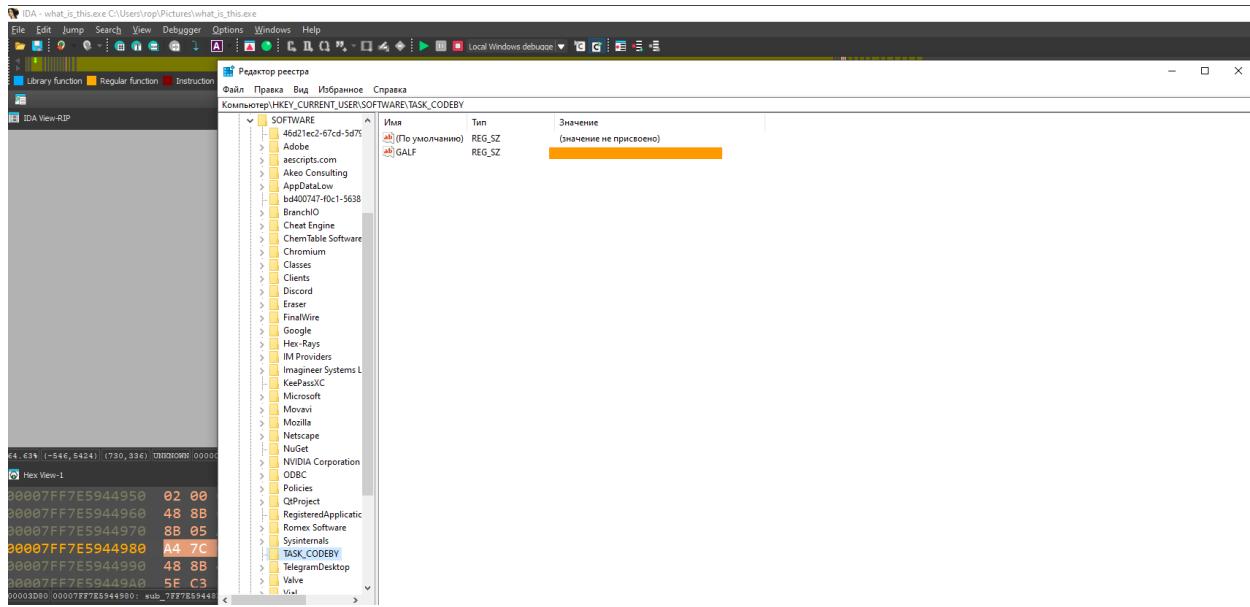
Address	Ordinal	Name	Library
00007FF7E5947000		VirtualFree	KERNEL32
00007FF7E5947008		VirtualAlloc	KERNEL32
00007FF7E5947010		LoadLibraryW	KERNEL32
00007FF7E5947018		GetProcAddress	KERNEL32
00007FF7E5947020		FreeLibrary	KERNEL32
00007FF7E5947028		RtlLookupFunctionEntry	KERNEL32
00007FF7E5947030		RtlVirtualUnwind	KERNEL32
00007FF7E5947038		IsDebuggerPresent	KERNEL32
00007FF7E5947040		UnhandledExceptionFilter	KERNEL32
00007FF7E5947048		SetUnhandledExceptionFilter	KERNEL32
00007FF7E5947050		IsProcessorFeaturePresent	KERNEL32

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 тут 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, 0x20}
```

```
char flag[] = "CODEBY <- remember, this is where the flag starts";
std::vector<uint8_t> encrypted_data = { 0xde, 0x12, 0x7a, 0x9c, 0x00 };

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

std::vector<uint8_t> encryptFunction(const std::vector<uint8_t>& func_data) {
    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(), encrypted_data.size());

    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(); ++i)
        data[i] = hmstr[padding_length];
}
```

```
        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(), decrypted_data.size());

    // 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, LPSECURITY_ATTRIBUTES);
typedef LONG(WINAPI* p_RegSetValueExA)(HKEY, LPCSTR, DWORD, DWORD, CONST BYTE*, DWORD);

```

```
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(hAdvApi32, "RegCreateKeyExA");
    if (dp_RegCreateKeyExA == NULL) {
        // Обработка ошибки получения адреса функции
        FreeLibrary(hAdvApi32);
        return;
    }

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

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

```
        FreeLibrary(hAdvApi32);
        return;
    }

}

void disinit_close_dynamics() {
    FreeLibrary(hAdvApi32);
}

/*void func_reg(p_RegCreateKeyExA RegCreateKeyE, p_RegSetValueExA RegSetValueE)
{
    bool flg = 0;

    if (flg) {
        unsigned char path[] = { 0x94, 0x90, 0x87, 0x95, 0x98, 0x82, 0x84, 0x8d, 0x87, 0x41 };

        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, 0x87, 0x8d, 0x87, 0x41 };

        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/открытый
    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
    RegCloseKey(hKey);
}

}
}/*

```

```
typedef void (*func_t)(p_RegCreateKeyExA, p_RegSetValueExA, p_RegCloseKeyA);

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

    // Выделение исполняемой памяти
    void* exec_mem = VirtualAlloc(nullptr, code.size(), MEM_COMMIT, PAGE_EXECUTE_READWRITE);
    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_RegCloseKeyA);

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

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

int main()
{
    //void* realFuncAddress = getRealFunctionAddress(reinterpret_cast<func_t>(func));
    //uint8_t* func_bytes = reinterpret_cast<uint8_t*>(realFuncAddress);

    //size_t func_size = getFunctionLength(reinterpret_cast<const func_t>(func));
    size_t func_size = encrypted_data.size();
    //std::vector<uint8_t> func_data(func_bytes, func_bytes + func_size);
}
```

```

// 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(function);
//std::vector<uint8_t> deccrypted_data = decryptFunction(encrypted_data);

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::setbase(16)
    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;  
}
```