1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт кибербезопасности и защиты информации

**ЛАБОРАТОРНАЯ РАБОТА № 1**

«**УДАЛЕННЫЙ ФАЙЛОВЫЙ МЕНЕДЖЕР С ИМПЕРСОНАЦИЕЙ КЛИЕНТА**»

1. по дисциплине «Безопасность современных информационных технологий»
2. Выполнил
3. студент гр. 4851003/00002 Вологдин М.В.

<*подпись*>

1. Преподаватель Дахнович А.Д.

<*подпись*>

1. Санкт-Петербург
2. 2022
3. **Цель работы**

Получить навыки работы с механизмом удаленного вызова процедур (RPC)

1. **Задачи**

Написать программу-сервер и программу-клиент, работающие под Windows 7–10. Сервер должен предоставлять доступ локальным и удаленным клиентам к файлам в своей файловой системе.

Требования:

- Statefull сервер;

- Сервер не должен быть интерактивным (интерфейс командной строки);

- Взаимодействие с клиентами должно осуществляться с помощью механизма RPC;

- При обслуживании клиента должна осуществляться его имперсонация;

- Пользователю должны предоставляться следующие операции: копирование указанного файла с клиента на сервер, загрузка указанного файла с сервера на клиента, удаление указанного файла на сервере;

- Имя файла передается в формате UNC.

В ходе работы должны быть проведены эксперименты с запуском сервера и клиентов под различными учетными записями с демонстрацией работы механизмов контроля доступа. Эксперименты должны показывать, что механизм имперсонации работает как при повышении, так и при понижении прав потока по отношению к процессу сервера.

1. **Ход работы**

1) Первым шагом выполнения лабораторной работы – создание и компиляция MIDL файла. Microsoft Interface Definition Language(MIDL) — текстовый язык описания интерфейсов. Был разработан файл config.idl.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 – Код разработанного интерфейса на языке MIDL

Как можно видеть на рисунке 1, в первой строке записывается uuid – уникальный идентификатор интерфейса, который был сгенерирован с помощью утилиты uuidgen (рисунок 2). Во второй строке записывается текущая версия, а далее идёт тело интерфейса, внутри которого записываются необходимые переменные и прототипы функций.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 – Генерация uuid

Таблица 1 – Функции config.idl

|  |  |
| --- | --- |
| Функция | Описание |
| deleteFile(const char\* path) | Функция удаления файла, расположенного на сервере по пути path |
| copyFile(const char\* path, char\* data, int size, int offset, int mode) | Функция копирования в файл, расположенный на сервере по пути path информации data размера size |
| downloadFile(const char\* path, char data[bufSize], int\* size, int offset) | Функция загрузки файла, расположенного на сервере по пути path размера size |
| authentication(const char\* login, const char\* password) | Функция входа в учётную запись |
| logout(void) | Функция выхода из учетной записи |
| Shutdown(void) | Функция завершения работы |

При компиляции были получены файлы config\_h.h, config\_s.cpp и config\_c.cpp. Второй файл добавляется в проект сервера, третий в проект клиента, а первый файл в оба проекта.

2) Программа-сервер

Сервер реализует все функции интерфейса, которые представлены в MIDL файле, регистрирует интерфейс, а также начинает прослушивать на порту входящие запросы на удалённое выполнение функций. Соединение осуществляется по TCP протоколу.

3) Программа-клиент

Клиент создает дескриптор привязки RPC, запрашивает логин и пароль пользователя, проводит авторизацию и далее выполняет запросы, которые вводятся с помощью командной строки.

1. **Тестирование**

Клиенту предлагается ввести ip-адрес и порт, затем данные. Если они корректны, пользователь видит сообщение об успешном входе и меню выбора доступных функций(рисунок 3), а сервер выводит имя подключенного клиента(рисунок 4). Если же данные некорректны, пользователь получает сообщение об этом и программа завершается(рисунок 5).

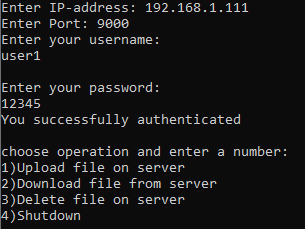


Рисунок 3 – Ввод корректных данных на клиенте



Рисунок 4 – Информация, выводимая сервером, при вводе корректных данных

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5 – Ввод некорректных данных на клиенте



Рисунок 6 – Информация, выводимая сервером, при вводе некорректных данных

1. Попытка скачать файл, на который клиент имеет права

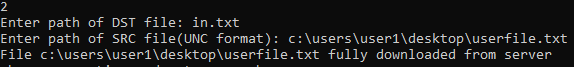


Рисунок 7 – Скачивание файла с сервера

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 8 – Права клиента на userfile.txt

1. Попытка удаления файла, на который у клиента есть права

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 9 – Удаление файла с сервера

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 10 – Права клиента на fordel.txt

3) Попытка удаления файла, на который у клиента нет прав

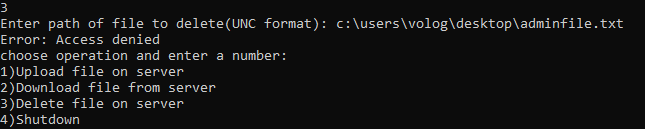


Рисунок 11 – Удаление файла adminfile.txt

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 12 – Права клиента на файл adminfile.txt

1. Копирование файла на сервер в разрешённую директорию

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 13 – Копирование файла input.txt на сервер

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 14 – Результат копирования файла input.txt на сервер

4) Копирование файла на сервер в запрещённую директорию

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 15 – Копирование файла input.txt на сервер

5) Завершение работы клиентом

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 16 – Результат вызова функции Shutdown на клиенте



Рисунок 17 – Результат выполнения функции Shutdown на сервере

1. **Контрольные вопросы**
2. В чем различия между statefull и stateless серверами?

В случае statefull, состояние(или сессия) хранится на сервере, благодаря этому нам не надо постоянно отправлять на сервер логин/пароль и не надо отсылать какие-то дополнительные данные, они хранятся в сессии.

В случае stateless, состояние не хранится на сервере. Т.е. в каждом новом запросе мы передаём свой логин/пароль, а также данные для запроса.

1. Что такое имперсонация?

Это концепт безопасности присущий Windows, что позволяет серверному приложению временно «быть» клиентом для доступа к охраняемому объекту. Имперсонация состоит из трёх уровней: идентификация, позволяющая серверу проверять подлинность клиента, имперсонализация, позволяющая серверу работать от имени клиента, и делегация: то же, что и имперсонализация, только расширена на работу с удалёнными системами, с которыми связывается сервер. То есть сервер временно наделяется правами клиента для совершения каких-либо действий.

1. Что такое LPC? чем этот механизм отличается от RPC?

LPC(Локальное межпроцессное взаимодействие) - это внутреннее недокументированное средство межпроцессного взаимодействия, предоставляемое ядром Microsoft Windows NT для облегченного IPC между процессами на одном компьютере.

Главным отличием LPC является то, что взаимодействие ограничено одним компьютером, всё происходит локально.

1. **Выводы**

В ходе работы были получены навыки работы с механизмом удалённого вызова процедур(RPC). Были реализованы клиент и сервер для файлового менеджера, который позволял копировать данные на сервер, скачивать данные с сервера, а также удалять данные на сервере.

**Приложение А**

Листинг программы “server.cpp”

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include "config\_h.h"

#pragma comment(lib,"rpcrt4.lib")

handle\_t handler = 0;

int deleteFile(

/\* [in] \*/ handle\_t IDL\_handle,

/\* [in] \*/ const unsigned char\* path)

{

printf("deleting %s\n", path);

if (remove((const char\*)path) == -1)

{

switch (errno)

{

case EACCES:

printf("remove error: file only for reading\n");

break;

case ENOENT:

printf("remove error: incorrect path\n");

break;

}

return errno;

}

printf("File successfully deleted!\n");

return 0;

}

int copyFile(

/\* [in] \*/ handle\_t IDL\_handle,

/\* [string][in] \*/ const unsigned char\* path,

/\* [string][in] \*/ const unsigned char\* data,

/\* [in] \*/ int size)

{

FILE\* inputFile = NULL;

inputFile = fopen((const char\*)path, "wb");

if (inputFile == NULL)

{

if (errno == EACCES)

printf("Access denied\n");

else if (errno == ENOENT)

printf("Incorrect path\n");

else

printf("Unknown error\n");

return errno;

}

int wrote = 0, tmp = 0;

while (wrote < size)

{

tmp = fwrite(data, 1, size - wrote, inputFile);

wrote += tmp;

}

fclose(inputFile);

printf("File successfully copied!\n");

return 0;

}

int downloadFile(

/\* [in] \*/ handle\_t IDL\_handle,

/\* [string][in] \*/ const unsigned char\* path,

/\* [out] \*/ unsigned char data[10240],

/\* [out] \*/ int\* size)

{

FILE\* outputFile = NULL;

if ((outputFile = fopen((const char\*)path, "rb")) == NULL)

{

if (errno == EACCES)

printf("Access denied\n");

else if (errno == ENOENT)

printf("Incorrect path\n");

else

printf("Unknown error\n");

return errno;

}

int read = 0, tmp = 0;

while (!feof(outputFile))

{

tmp = fread(data + read, 1, 1024, outputFile);

read += tmp;

}

fclose(outputFile);

\*size = read;

printf("File successfully downloaded!\n");

return 0;

}

void Shutdown(

/\* [in] \*/ handle\_t IDL\_handle)

{

RPC\_STATUS ret;

ret = RpcMgmtStopServerListening(NULL);

if (ret)

exit(ret);

ret = RpcServerUnregisterIf(NULL, NULL, FALSE);

if (ret)

exit(ret);

printf("Session successfully ended\n");

}

int authentication(

/\* [in] \*/ handle\_t IDL\_handle,

/\* [string][in] \*/ const unsigned char\* login,

/\* [string][in] \*/ const unsigned char\* password)

{

//пытается авторизовать пользователя на локальном компьютере

if (LogonUserA((LPCSTR)login, NULL, (LPCSTR)password, LOGON32\_LOGON\_INTERACTIVE, LOGON32\_PROVIDER\_DEFAULT, &IDL\_handle) == 0)

{

printf("Error LogonUserA: %d(user %s)\n", GetLastError(), login);

return 1;

}

else

{

//позволяет вызывающему потоку олицетворять контекст безопасности вошедшего в систему пользователя

if (ImpersonateLoggedOnUser(IDL\_handle) == 0)

return 2;

else

{

printf("%s connected\n", login);

return 0;

}

}

}

int printError(const char\* func, int code)

{

printf("%s - Error code: %d\n", func, code);

exit(code);

}

int main()

{

RPC\_STATUS ret;

ret = RpcServerUseProtseqEpA(

(RPC\_CSTR)("ncacn\_ip\_tcp"), //используем TCP/IP протокол

RPC\_C\_PROTSEQ\_MAX\_REQS\_DEFAULT, //размер бэклога TCP/IP

(RPC\_CSTR)("9000"), //порт

NULL); //NULL для TCP/IP

if (ret)

printError("RpcServerUseProtseqEpA", ret);

ret = RpcServerRegisterIfEx(InterfaceRPC\_v1\_0\_s\_ifspec, NULL, NULL, RPC\_IF\_ALLOW\_CALLBACKS\_WITH\_NO\_AUTH, RPC\_C\_LISTEN\_MAX\_CALLS\_DEFAULT, NULL);

if (ret)

printError("RpcServerRegisterIfEx", ret);

ret = RpcServerListen(

1,//рекомендуемое минимальное количество потоков

RPC\_C\_LISTEN\_MAX\_CALLS\_DEFAULT,//рекомендуемое максимальное количество потоков

0);//без ожидания

if (ret)

printError("RpcServerListen", ret);

return 0;

}

void \_\_RPC\_FAR\* \_\_RPC\_API midl\_user\_allocate(size\_t len)

{

return(malloc(len));

}

void \_\_RPC\_API midl\_user\_free(void \_\_RPC\_FAR\* ptr)

{

free(ptr);

}

**Приложение Б**

Листинг программы “client.cpp”

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include "config\_h.h"

#pragma comment(lib,"rpcrt4.lib")

#define BUF\_LEN 128

char username[BUF\_LEN];

char password[BUF\_LEN];

int printError(const char\* func, int code)

{

printf("%s - Error code: %d\n", func, code);

exit(code);

}

void usage()

{

printf(

"choose operation and enter a number:\n"

"1)Upload file on server\n"

"2)Download file from server\n"

"3)Delete file on server\n"

"4)Shutdown\n");

}

void login()

{

printf("Enter your username:\n");

if (!scanf("%100s", username))

exit(-1);

username[strlen(username)] = '\0';

printf("\nEnter your password:\n");

if (!scanf("%100s", password))

exit(-1);

password[strlen(password)] = '\0';

}

void do\_upload(handle\_t binding)

{

printf("Enter path of SRC file: ");

char uploadPath[101];

char n = getchar();

fgets(uploadPath, 100, stdin);

printf("Enter path of DST file(UNC format): ");

char savePath[101];

fgets(savePath, 100, stdin);

uploadPath[strlen(uploadPath) - 1] = '\0';

savePath[strlen(savePath) - 1] = '\0';

printf("\n");

FILE\* inFile = NULL;

inFile = fopen(uploadPath, "rb");

if (inFile == NULL)

return;

char buf[10240] = { 0 };

int read = 0,tmp = 0;

while (!feof(inFile))

{

tmp = fread(buf + read, 1, 1024, inFile);

read += tmp;

}

fclose(inFile);

int ret = copyFile(binding, (const unsigned char\*)savePath, (const unsigned char\*)buf, strlen(buf));

if (!ret)

printf("File %s fully uploaded on servers\n", uploadPath);

else if (ret == EACCES)

printf("Error: Access denied\n");

else if (ret == ENOENT)

printf("Error: Incorrect path\n");

}

void do\_download(handle\_t binding)

{

printf("Enter path of DST file: ");

char n = getchar();

char savePath[101];

fgets(savePath, 100, stdin);

printf("Enter path of SRC file(UNC format): ");

char downloadPath[101];

fgets(downloadPath, 100, stdin);

downloadPath[strlen(downloadPath) - 1] = '\0';

savePath[strlen(savePath) - 1] = '\0';

unsigned char inputBuffer[10240];

memset(inputBuffer, 0, sizeof(inputBuffer));

int size = 0;

int ret = downloadFile(binding, (const unsigned char\*)downloadPath, inputBuffer, &size);

if (!ret)

{

FILE\* inFile = NULL;

inFile = fopen(savePath, "wb");

if (inFile == NULL)

return;

int wrote = 0, tmp = 0;

while (wrote < size)

{

tmp = fwrite(inputBuffer + wrote, 1, size - wrote, inFile);

wrote += tmp;

}

fclose(inFile);

printf("File %s fully downloaded from server\n", downloadPath);

}

else if (ret == EACCES)

printf("Error: Access denied\n");

else if (ret == ENOENT)

printf("Error: Incorrect path\n");

}

void do\_delete(handle\_t binding)

{

printf("Enter path of file to delete(UNC format): ");

char n = getchar();

char deletePath[101];

fgets(deletePath, 100, stdin);

deletePath[strlen(deletePath) - 1] = '\0';

int ret = deleteFile(binding, (const unsigned char\*)deletePath);

if (!ret)

printf("File %s fully deleted from server\n", deletePath);

else if (ret == EACCES)

printf("Error: Access denied\n");

else if (ret == ENOENT)

printf("Error: Incorrect path\n");

}

void do\_shutdown(handle\_t binding)

{

Shutdown(binding);

}

int main()

{

RPC\_STATUS status;

RPC\_CSTR szStringBinding = NULL;

char addr[20];

printf("Enter IP-address: ");

scanf("%s", addr);

char port[5] = "0";

printf("Enter Port: ");

scanf("%s", port);

// Creates a string binding handle.

// This function is nothing more than a printf.

// Connection is not done here.

status = RpcStringBindingComposeA(

NULL, // UUID для привязки

(RPC\_CSTR)("ncacn\_ip\_tcp"),//используем TCP/IP протокол

(RPC\_CSTR)(addr),//TCP/IP адрес

(RPC\_CSTR)(port),//TCP/IP порт

NULL,//параметры сети

&szStringBinding);//указатель на строковое представление дескриптора привязки

if (status)

printError("RpcStringBindingComposeA", status);

RPC\_BINDING\_HANDLE binding;

status = RpcBindingFromStringBindingA(

szStringBinding,

&binding);

if (status)

printError("RpcBindingFromStringBindingA", status);

login();

if (authentication(binding, (const unsigned char\*)username, (const unsigned char\*)password))

{

puts("Wrong login or password");

exit(0);

}

printf("You successfully authenticated\n\n");

RpcTryExcept

{

int input = -1;

while (input != 4)

{

usage();

scanf("%d", &input);

switch (input)

{

case 1:

do\_upload(binding);

break;

case 2:

do\_download(binding);

break;

case 3:

do\_delete(binding);

break;

case 4:

do\_shutdown(binding);

break;

}

}

}

RpcExcept(1)

{

printf("Runtime reported exception %ld", RpcExceptionCode());

}

RpcEndExcept

status = RpcStringFreeA(&szStringBinding);

if (status)

printError("RpcStringFreeA", status);

status = RpcBindingFree(&binding);

if (status)

printError("RpcBindingFree", status);

printf("Session successfully ended\n");

return 0;

}

void \_\_RPC\_FAR\* \_\_RPC\_API midl\_user\_allocate(size\_t len)

{

return(malloc(len));

}

void \_\_RPC\_API midl\_user\_free(void \_\_RPC\_FAR\* ptr)

{

free(ptr);

}

**Приложение В**

Листинг программы “config.idl”

[

uuid(da405304-8b4b-4291-8c76-7045a2734db6),

version(1.0)

]

interface InterfaceRPC

{

const unsigned int bufSize = 10240;

int deleteFile(

[in, string] const char\* path

);

int copyFile(

[in, string] const char\* path,

[in, string] const char\* data,

[in] int size

);

int downloadFile(

[in, string] const char\* path,

[out] char data[bufSize],

[out] int\* size

);

int authentication(

[in, string] const char\* login,

[in, string] const char\* password

);

void Shutdown(void);

}