Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Информационные сети. Основы безопасности

ОТЧЕТ

к лабораторной работе №2

на тему

**ИДЕНТИФИКАЦИЯ И АУТЕНТИФИКАЦИЯ ПОЛЬЗОВАТЕЛЕЙ. ПРОТОКОЛ KERBEROS.**

Студент А. В. Скворцов

Преподаватель Е. А. Лещенко

Минск 2024

СОДЕРЖАНИЕ

[Введение 3](#_Toc160192830)

[1 Результат выполнения 4](#_Toc160192831)

[Заключение 5](#_Toc160192832)

[Приложение А (обязательное) Листинг кода 6](#_Toc160192833)

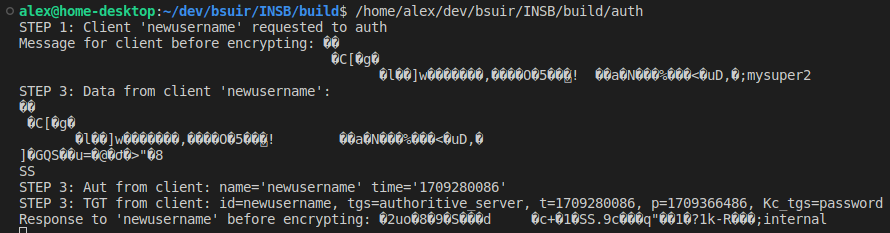
**ВВЕДЕНИЕ**

В ходе лабораторной работы нужно разработать программные средства, реализующие протокол Kerberos, такие как клиентское приложение, сервер аутентификации и сервер для получения данных. Сервер для получения данных подразумевает собой целевую точку, куда должен аутентифицироваться клиент.

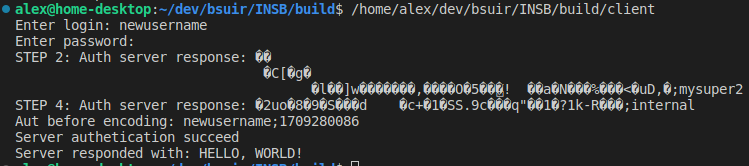
Также в лабораторной работе необходимо реализовать шифрование, использующее алгоритм DES. Этот вид шифрования должен использоваться для безопасной пересылки сообщений между приложениями.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

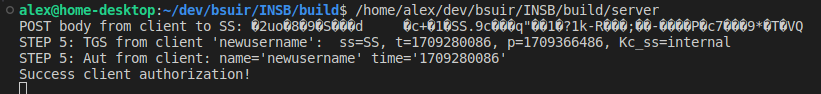
В результате работы было создано три приложения, которые взаимодействуют по протоколу Kerberos. Они представлены в виде двух TCP серверов и одного клиента. На представленных ниже рисунках можно увидеть то, как они взаимодействуют между собой (рисунок 1).



а



б



в

а – сервер аутентификации; б – клиент; в – сервер

Рисунок 1 – Протокол Kerberos

Чтобы увидеть, как общаются программы по протоколу HTTP была использована утилита Wireshark (рисунок 2).

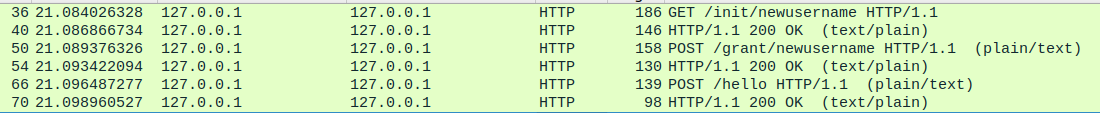


Рисунок 2 – HTTP запросы программ

# ЗАКЛЮЧЕНИЕ

В результате выполнения данной лабораторной работы были разработано программное средство для шифрования с помощью алгоритма DES на языке программирования *С++*.

Также было проведено изучение протокола Kerberos и выполнена его упрощенная реализация в виде трех приложений. Сделан пример аутентификации пользователя по этому протоколу, состоящий из шагов, описанных в условии лабораторной работы.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл *auth.cpp*

#include "httplib.h"

#include <map>

#include <iostream>

#include <chrono>

#include "des/des.h"

#include "utils.h"

using namespace httplib;

using namespace std;

using namespace chrono\_literals;

struct ClientInfo {

string key;

string tgs\_key;

string c\_ss;

vector<string> allowed\_servers;

};

struct ServerInfo {

string tgs\_key;

};

const string MASTER\_PASSWORD = "password";

const string TGS = "authoritive\_server";

map<string, ClientInfo> clients = {

{"newusername", {

.key = "password",

.tgs\_key = "mysuper2",

.c\_ss = "internal",

.allowed\_servers = {

"SS",

},

}},

};

map<string, ServerInfo> servers = {

{"SS", {

.tgs\_key = "serverkc",

}}};

void initMessage(const Request& request, Response& response) {

string id = request.path\_params.at("id");

if (clients.find(id) == clients.end()) {

response.status = 403;

response.set\_content("Forbidden", 10, "text/plain");

return;

}

cout << "STEP 1: Client '" << id << "' requested to auth" << endl;

DES c;

auto current\_time = chrono::system\_clock::now();

auto end\_time = current\_time + 24h;

string TGT = id + ";" + TGS + ";" + to\_string(chrono::system\_clock::to\_time\_t(current\_time)) + ";" + to\_string(chrono::system\_clock::to\_time\_t(end\_time)) + ";" + clients[id].key;

c.setKey(MASTER\_PASSWORD);

string TGT\_encrypted = c.encryptAnyString(TGT);

string answer = TGT\_encrypted + ";" + clients[id].tgs\_key;

c.setKey(clients[id].key);

cout << "Message for client before encrypting: " << answer << endl;

string encrypted = c.encryptAnyString(answer);

response.set\_content(encrypted.c\_str(), encrypted.size(), "text/plain");

response.status = 200;

}

void grant(const Request& request, Response& response) {

string id = request.path\_params.at("id");

if (clients.find(id) == clients.end()) {

response.status = 403;

response.set\_content("Forbidden", 10, "text/plain");

return;

}

auto data = split(request.body, ';');

cout << "STEP 3: Data from client '" << id << "':" << endl;

for (const auto& cur : data) {

cout << cur << endl;

}

DES c;

c.setKey(clients[id].tgs\_key);

auto aut = split(c.decryptAnyString(data[1]), ';');

cout << "STEP 3: Aut from client: name='" << id << "' time='" << aut.back() << "'" << endl;

string server\_id = data.back().c\_str();

if (

find(clients[id].allowed\_servers.begin(), clients[id].allowed\_servers.end(), server\_id) == clients[id].allowed\_servers.end() ||

aut.front().substr(0, 8) == id) {

response.status = 403;

const char\* resp = "Forbidden (incorrect user)";

response.set\_content(resp, strlen(resp), "text/plain");

return;

}

auto ticket\_creation = chrono::system\_clock::from\_time\_t(std::atoll(aut.back().c\_str()));

c.setKey(MASTER\_PASSWORD);

auto tgt = split(c.decryptAnyString(data.front()), ';');

cout << "STEP 3: TGT from client" << ": ";

cout << "id=" << tgt[0] << ", ";

cout << "tgs=" << tgt[1] << ", ";

cout << "t=" << tgt[2] << ", ";

cout << "p=" << tgt[3] << ", ";

cout << "Kc\_tgs=" << tgt[4] << endl;

if (

chrono::system\_clock::from\_time\_t(std::atoll(tgt[2].c\_str())) > ticket\_creation ||

chrono::system\_clock::from\_time\_t(std::atoll(tgt[3].c\_str())) < ticket\_creation) {

response.status = 403;

const char\* resp = "Forbidden (ticket expired)";

response.set\_content(resp, strlen(resp), "text/plain");

return;

}

c.setKey(servers[server\_id].tgs\_key);

auto current\_time = chrono::system\_clock::now();

auto end\_time = current\_time + 24h;

string tgs\_server = id + ";" + server\_id + ";" + to\_string(chrono::system\_clock::to\_time\_t(current\_time)) + ";" + to\_string(chrono::system\_clock::to\_time\_t(end\_time)) + ";" + clients[id].c\_ss;

string tgs\_server\_encrypted = c.encryptAnyString(tgs\_server);

string answer = tgs\_server\_encrypted + ";" + clients[id].c\_ss;

cout << "Response to '" << id << "' before encrypting: " << answer << endl;

c.setKey(clients[id].tgs\_key);

string answer\_encrypted = c.encryptAnyString(answer);

response.body = answer\_encrypted;

response.status = 200;

}

int main() {

Server svc;

svc.Get("/init/:id", initMessage);

svc.Post("/grant/:id", grant);

svc.listen("0.0.0.0", 8080);

return 0;

}

Листинг 2 – Файл *client.cpp*

#include <iostream>

#include "des/des.h"

#include "httplib.h"

#include "utils.h"

#include <chrono>

#include <termios.h>

using namespace std;

using namespace httplib;

constexpr const char\* server = "SS";

void disableEcho() {

termios term;

tcgetattr(STDIN\_FILENO, &term);

term.c\_lflag &= ~ECHO;

tcsetattr(STDIN\_FILENO, TCSANOW, &term);

}

void enableEcho() {

termios term;

tcgetattr(STDIN\_FILENO, &term);

term.c\_lflag |= ECHO;

tcsetattr(STDIN\_FILENO, TCSANOW, &term);

}

int main() {

Client cli("http://127.0.0.1:8080");

string username, password;

cout << "Enter login: ";

cin >> username;

char ch;

disableEcho();

cout << "Enter password: ";

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

std::getline(std::cin, password);

cout << endl;

enableEcho();

auto response = cli.Get("/init/" + username);

if (response->status != 200) {

cout << "Auth server answered with " << response->status << " code" << endl;

return -1;

}

string resp = response->body;

DES c;

c.setKey(password);

string result = c.decryptAnyString(resp);

cout << "STEP 2: Auth server response: " << result << endl;

string tgs\_key = split(result, ';').back().substr(0, 8);

string aut = username + ";" + to\_string(chrono::system\_clock::to\_time\_t(chrono::system\_clock::now()));

c.setKey(tgs\_key);

string aut\_encrypted = c.encryptAnyString(aut);

string request\_to\_tgs = split(result, ';').front() + ";" + aut\_encrypted + ";" + server;

response = cli.Post("/grant/" + username, request\_to\_tgs, "plain/text");

if (response->status != 200) {

cout << "Auth server answered with non 200 code: " << response->status << endl;

return -1;

}

result = c.decryptAnyString(response->body);

cout << "STEP 4: Auth server response: " << result << endl;

Client hello("http://localhost:8081");

auto tgs = split(result, ';').front();

auto current\_time = chrono::system\_clock::to\_time\_t(chrono::system\_clock::now());

aut = username + ";" + to\_string(current\_time);

cout << "Aut before encoding: " << aut << endl;

c.setKey(string(split(result, ';').back()));

auto request\_to\_server = tgs + ";" + c.encryptAnyString(aut);

response = hello.Post("/hello", request\_to\_server, "plain/text");

if (response->status != 200) {

cout << "Server answered with non 200 code: " << response->status << endl;

return -1;

}

auto answer = split(c.decryptAnyString(response->body), ';');

if (std::atoll(answer.back().c\_str()) != current\_time + 1) {

cout << "Server verification failed!" << endl;

return 1;

}

cout << "Server authetication succeed" << endl;

cout << "Server responded with: " << answer.front() << endl;

return 0;

}