Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Лабораторная работа №4-5. Разработка защищенных приложений

Выполнил: cтудент гр. 953502

Ганусевич М. В.

Проверил:

Протько М.И.

Минск 2022

# Введение

**Цель работы:** познакомиться с концепцией ролевого управления доступом и способами защиты программного обеспечения от существующих угроз, научиться разрабатывать приложения, которые используют ролевое управление доступом для разграничения полномочий пользователей. Получить навыки защиты разработанной программы от несанкционированного копирования и других угроз, которым может подвергаться программное обеспечение.

**Теоретические сведения**

Перечислим отраженные в приложении атаки.

**Ошибки канонизации.**

Часто приложения принимают решения об авторизации на основе аутентификационной информации и запрошенных ресурсов. Многие уязвимости безопасности относятся к ошибкам канонизации в имени ресурса. Например, хотя приложение может запретить доступ к *\secure\secret.txt*, оно может дать доступ к *\public\..\secure\secret.txt* на основании имени директории. Кодировки в Юникод и в шестнадцатеричном формате относятся к той же категории.

**Принцип минимизации привилегий**

Принцип минимизации привилегий предписывает выделять пользователям только те права доступа, которые необходимы им для выполнения служебных обязанностей. Назначение этого принципа очевидно - уменьшить ущерб от случайных или умышленных некорректных действий пользователей.

**DoS**

DoS — хакерская атака на вычислительную систему с целью довести её до отказа, то есть создание таких условий, при которых добросовестные пользователи системы не смогут получить доступ к предоставляемым системным ресурсам (серверам), либо этот доступ будет затруднён. Отказ «вражеской» системы может быть и шагом к овладению системой (если в нештатной ситуации ПО выдаёт какую-либо критическую информацию — например, версию, часть программного кода и т. д.). Но чаще это мера экономического давления: потеря простой службы, приносящей доход, счета от провайдера и меры по уходу от атаки ощутимо бьют «цель» по карману. В настоящее время DoS и DDoS-атаки наиболее популярны, так как позволяют довести до отказа практически любую плохо написанную систему, не оставляя юридически значимых улик.

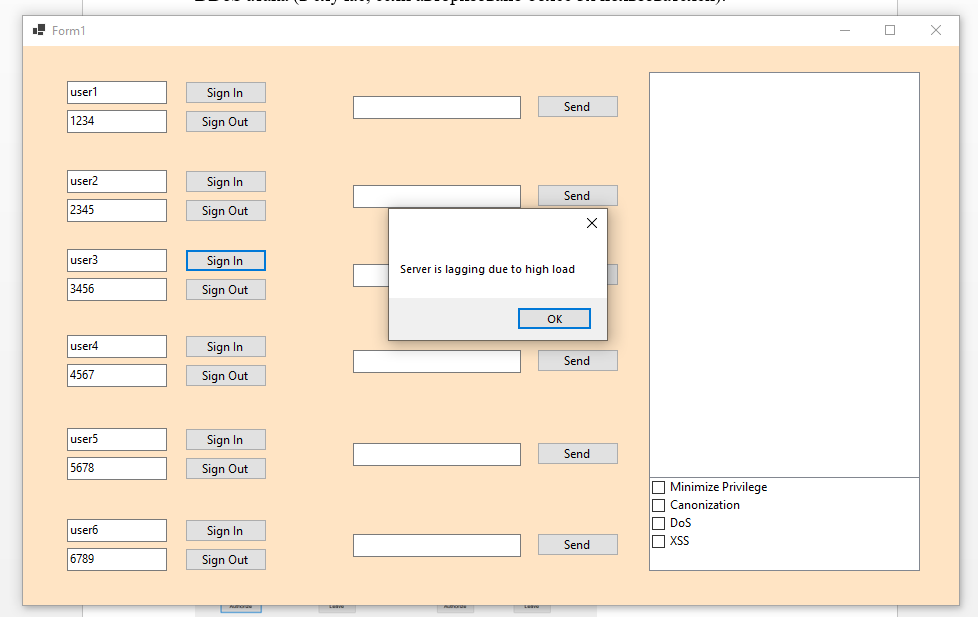
**XSS атака**

XSS (межсайтовый скриптинг) – одна из разновидностей атак на веб-системы, которая подразумевает внедрение вредоносного кода на определенную страницу сайта и взаимодействие этого кода с удаленным сервером злоумышленников при открытии страницы пользователем.

Термин с английского расшифровывается как Cross-Site Scripting, но при этом получил аббревиатуру XSS, чтобы не было путаницы с CSS (каскадные таблицы стилей).

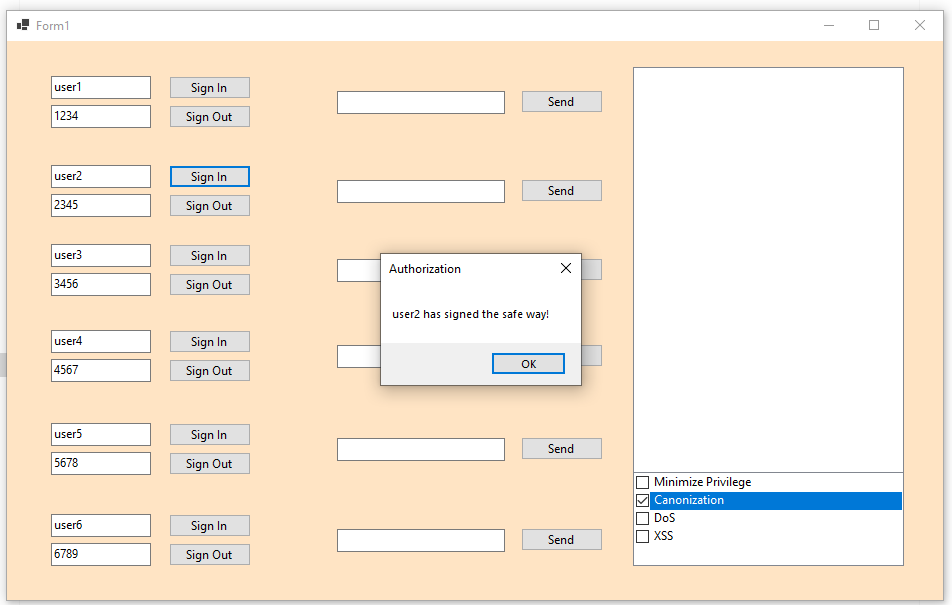
# Результат работы программы

DDoS атака (в случае, если авторизовано более 3х пользователей):

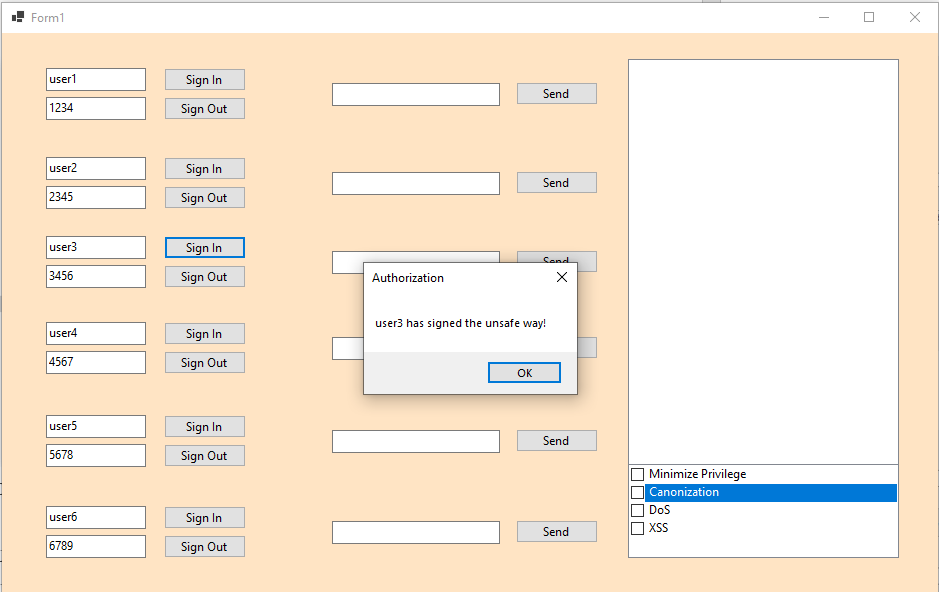


Ошибка канонизации (в данном случае авторизация без хеширования)

Случай без атаки:



Во время атаки:

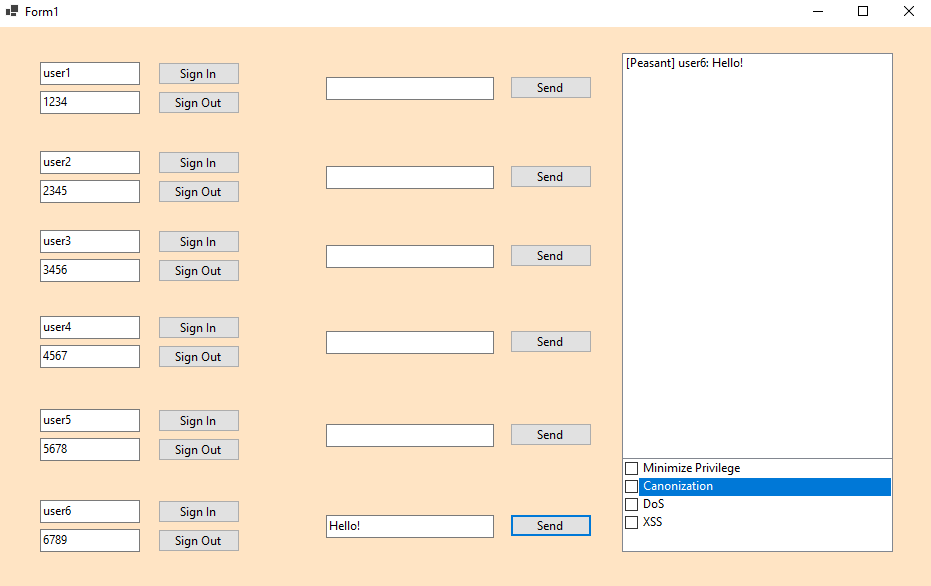


Можем обратить внимание на различия в выводе программы для этих случаев

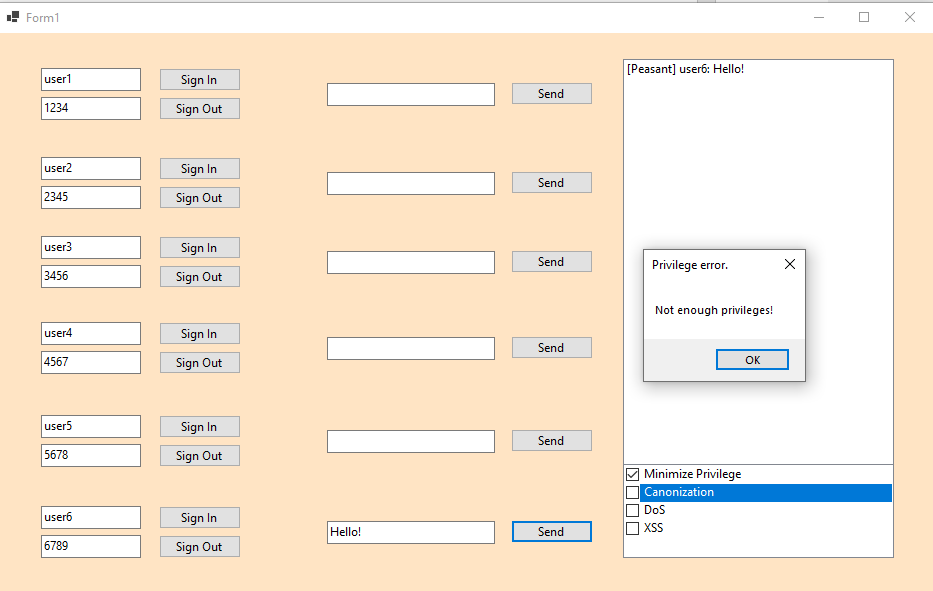
Минимизация привилегий (запрет на отправку сообщений в чат самому низкому уровню):

До атаки:

Сервер позволяет пользователю с привилегиями уровня Peasant писать в чат

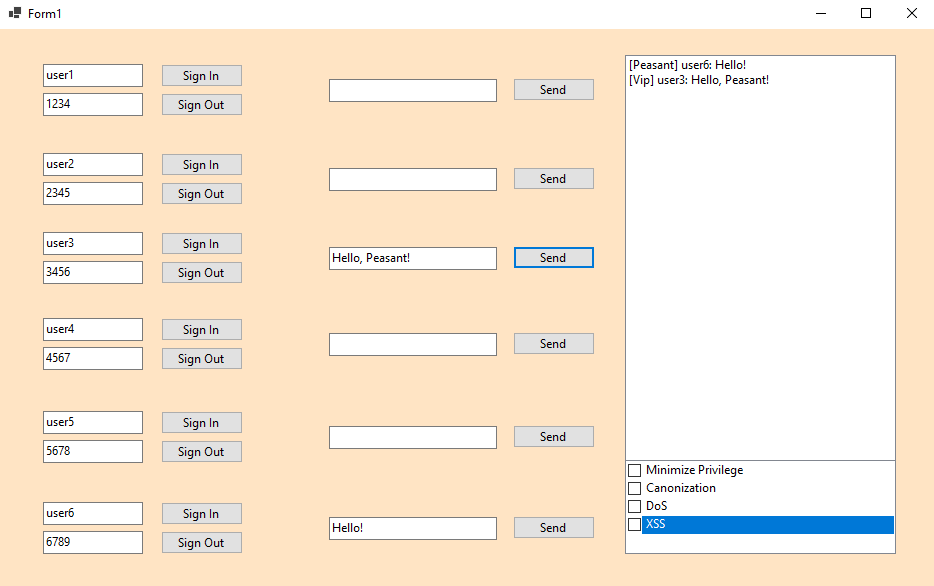


Во время атаки:

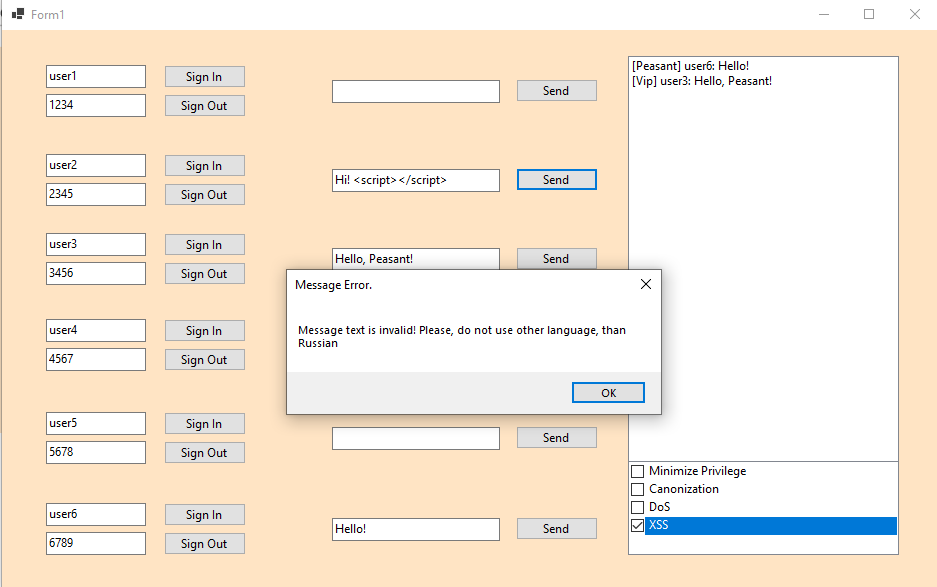
Сервер запрещает пользователю с привилегиями уровня Peasant писать в чат 

XSS (в данном случае – запрет на ввод любых символов кроме цифр и кириллицы):

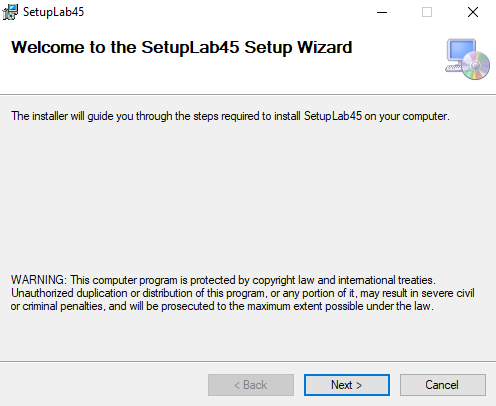
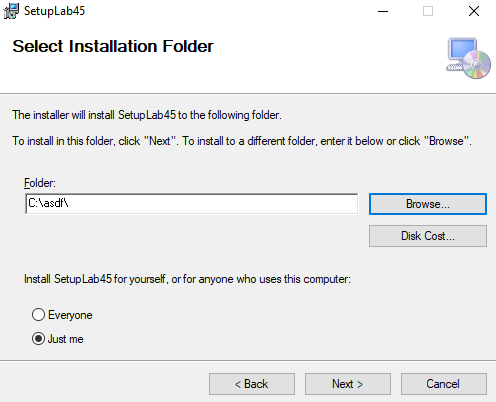
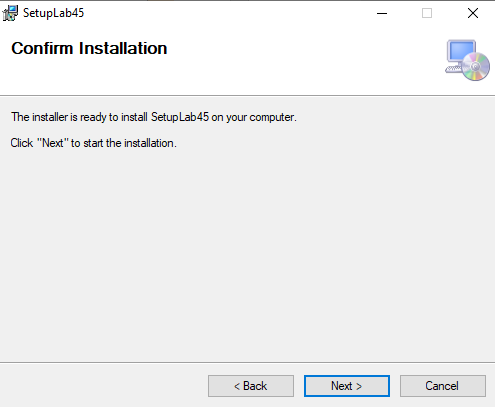
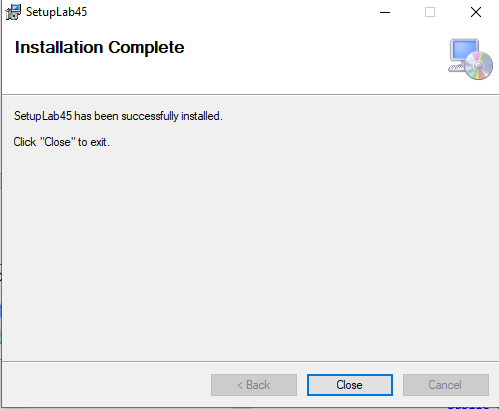
До атаки:



Во время атаки:



**Установщик для данного приложения**

# Код программы

using System.Security.Cryptography;

using System.Text;

using System.Text.RegularExpressions;

namespace Lab45

{

public partial class Form1 : Form

{

private enum Roles

{

Admin,

Vip,

Peasant

}

private enum AuthorizationResult

{

Success,

Fail,

}

private class DefenseSystemSwitcher

{

public bool PrivilegeMinimization;

public bool Canonization;

public bool Xss;

public bool Dos;

}

private class User

{

public string Login;

public string Password;

public Roles Role;

public bool IsSignedIn;

public int SlotIndex;

public User(string login, string password)

{

Login = login;

Password = password;

IsSignedIn = false;

SlotIndex = -1;

}

public override bool Equals(object? obj)

{

return obj is User user &&

Login == user.Login;

}

}

private List<User> usersData = new();

private List<(byte[], byte[])> encryptedUsersData = new();

private readonly SHA256 sha256 = SHA256.Create();

public Form1()

{

InitializeComponent();

for (int i = 1; i < 7; i++)

{

string user = $"user{i}";

string password = $"{i}{i + 1}{i + 2}{i + 3}";

byte[] encryptedUser = ASCIIEncoding.ASCII.GetBytes(user);

byte[] encryptedPassword = ASCIIEncoding.ASCII.GetBytes(password);

usersData.Add(new User(user, password));

encryptedUsersData.Add((sha256.ComputeHash(encryptedUser), sha256.ComputeHash(encryptedPassword)));

this.Controls[$"Login{i}"].Text = user;

this.Controls[$"Password{i}"].Text = password;

loginFields.Add(this.Controls[$"Login{i}"] as TextBox);

passwordsFields.Add(this.Controls[$"Password{i}"] as TextBox);

messageFields.Add(this.Controls[$"Message{i}"] as TextBox);

signInButtons.Add(this.Controls[$"SignIn{i}"] as Button);

sendButtons.Add(this.Controls[$"Send{i}"] as Button);

}

usersData[0].Role = Roles.Admin;

usersData[1].Role = Roles.Admin;

usersData[2].Role = Roles.Vip;

usersData[3].Role = Roles.Vip;

usersData[4].Role = Roles.Vip;

usersData[5].Role = Roles.Peasant;

}

private int GetIndex(Button sender) => Convert.ToInt32((sender).Name.Last().ToString()) - 1;

private bool GetIsServerLagging(int limit = 3) => usersData.FindAll(user => user.IsSignedIn).Count() >= limit;

private DefenseSystemSwitcher GetDefenseSystemSwitcher()

{

DefenseSystemSwitcher defenseCheks = new();

bool[] bools = new bool[4];

foreach (var check in AttackDefensesCheckedListBox.CheckedIndices)

{

bools[(int)check] = true;

}

defenseCheks.PrivilegeMinimization = bools[0];

defenseCheks.Canonization = bools[1];

defenseCheks.Dos = bools[2];

defenseCheks.Xss = bools[3];

return defenseCheks;

}

private void SignIn\_Click(object sender, EventArgs e)

{

int index = GetIndex(sender as Button);

string login = loginFields[index].Text;

string password = passwordsFields[index].Text;

User userByName = usersData.Find(user => user.Login == login);

DefenseSystemSwitcher defense = GetDefenseSystemSwitcher();

if (0 > index || index >= usersData.Count)

{

return;

}

if (GetIsServerLagging())

{

if (defense.Dos)

{

MessageBox.Show("Server is full, please wait in queue");

return;

}

else

{

MessageBox.Show("Server is lagging due to high load");

}

}

if (userByName != null && userByName.IsSignedIn)

{

MessageBox.Show($"User {login} has been already authorized on slot {userByName.SlotIndex + 1}", "Authorization Error");

return;

}

if (defense.Canonization)

{

byte[] loginHash = sha256.ComputeHash(ASCIIEncoding.ASCII.GetBytes(login));

byte[] passwordHash = sha256.ComputeHash(ASCIIEncoding.ASCII.GetBytes(password));

foreach (var user in encryptedUsersData)

{

if (loginHash.SequenceEqual(user.Item1) && passwordHash.SequenceEqual(user.Item2))

{

userByName.IsSignedIn = true;

userByName.SlotIndex = index;

break;

}

}

}

else

{

if (userByName != null && userByName.Password == password)

{

userByName.IsSignedIn = true;

userByName.SlotIndex = index;

}

}

string way = defense.Canonization ? "the safe way" : "the unsafe way";

MessageBox.Show(userByName.IsSignedIn

? $"{userByName.Login} has signed {way}!"

: $"login or password are invalid, please, try again!", "Authorization");

}

private void Send\_Click(object sender, EventArgs e)

{

int index = GetIndex(sender as Button);

string messageText = messageFields[index].Text;

string answer;

var userByIndex = usersData.Find(user => user.SlotIndex == index);

DefenseSystemSwitcher defense = GetDefenseSystemSwitcher();

if (0 > index || index >= usersData.Count)

{

return;

}

if (userByIndex == null)

{

MessageBox.Show("User must be authorized to send messages from this slot!", "User Error.");

return;

}

if (defense.Xss)

{

if (!Regex.IsMatch(messageText, @"^[А-я0-9]\*$"))

{

MessageBox.Show("Message text is invalid! Please, do not use other language, than Russian", "Message Error.");

return;

}

}

if (defense.PrivilegeMinimization && userByIndex.Role == Roles.Peasant)

{

MessageBox.Show("Not enough privileges!", "Privilege error.");

return;

}

messageText = messageText.Insert(0, $"[{userByIndex.Role}] {userByIndex.Login}: ");

if (GetIsServerLagging(4))

{

Thread.Sleep(500);

}

MessagesListBox.Items.Add(messageText);

}

private void SignOut\_Click(object sender, EventArgs e)

{

int index = GetIndex(sender as Button);

var userByIndex = usersData.Find(user => user.SlotIndex == index);

if (userByIndex == null)

{

MessageBox.Show($"This slot is empty. No users signed in.", "Signing out.");

return;

}

if (userByIndex != null)

{

userByIndex.SlotIndex = -1;

userByIndex.IsSignedIn = false;

MessageBox.Show($"User has been signed out", "Signing out.");

}

}

}

}

# Вывод

В рамках данной лабораторной работы я реализовал приложение, презентующее типичные ошибки на ПО и способы защиты от них, а также приложение-установщик для вышеуказанного приложения.