X ОЛИМПИАДА ПО ИНФОРМАТИКЕ И КОМПЬЮТЕРНОЙ БЕЗОПАСНОСТИ
Вариант 1

Задача 1. Стеганография

Информация в сети передается с помощью пакетов. Каждый из них состоит из заголовка, данных и контрольной суммы:

Заголовок			**Данные**	**Выравнивание до целого числа байт**	**Контрольная сумма**
Адрес источника	Адрес назначения	Размер данных (бит)			1 байт (количество единиц в бинарном представлении по модулю 256)
6 байт	6 байт	2 байта			

Вася обнаружил в исходящем сетевом трафике своего компьютера несколько странных пакетов и подозревает, что в них содержится скрытое сообщение. Помогите Васе определить, что именно было передано?

```
001122332211009988776655005300000000000000000000024
00112233221100998877665500770000000000000000000000000026
0011223322110099887766550060000000000000000000000026
00112233221100998877665500720000000000000000000000000024
00112233221100998877665500640000000000000000000000023
```

Решение.

Структура пакета напоминает сетевой протокол Ethernet, в котором данные передаются в виде пакетов, состоящих из заголовка, данных и контрольной суммы. В заголовках присутствует обязательно MAC-адрес источника (6 байт), MAC-адрес получателя (6 байт), размер пакета (вычисляется автоматически). Контрольная сумма вычисляется автоматически.

Пользователь может повлиять на значение следующих полей пакета:

- поле данных,
- поле размера пакета (вычисляется исходя из размера поля данных).

Если рассматривать поле данных обнаруженных пакетов, то можно заметить следующие особенности:

1) содержимое пакетов повторяется;
2) читаемых данных (учитывая ASCII-таблицу) в содержимом

пакетов нет.

По указанным особенностям можно сделать предположение, что в теле пакетов полезных данных нет. Остается другая часть пакета – заголовок.

В заголовке все поля стандартные и нельзя их использовать для передачи какой-либо информации. Единственное поле, значение которого различное во всех пакетах – это поле размера пакета. Если перевести значение размера пакета по ASCII-таблице в символы, то можно получить читаемые буквы. Исходя из этого, можно сделать предположение о том, что данные были преданы в размере пакета. Каждый пакет передает какой-то символ, а именно: размер пакета – код символа в ASCII-таблице.

В результате получаем сообщение из 5 символов с кодами 0x53, 0x77, 0x6F, 0x72, 0x64, что соответствует слову *«Sword»*.

Ответ: Sword

<u>Задача 2.</u> Вирус

Полиморфный вирус дописывает к заражаемой программе: код расшифровщика, команду безусловного перехода, случайные байты и вредоносный код:

Код расшифровщика	Код заражаемой программы	E9(JMP) (1 байт)	Смещение (2 байта)	Случайные байты	Вредоносный код

При этом вредоносный код записывается в зашифрованном виде. Ниже приведена функция, которая использовалась для шифрования:

```
// crypto_const – неизвестная константа;
char encode(char code, const char crypto_const)
{
     return (code ^ crypto_const);
}
```

Кроме того, известно, что для перехода на начало собственно вредоносного кода применяется команда безусловного перехода *JMP*, которая в незашифрованном виде имеет код E9. После этого следуют 2 байта величины смещения относительно следующей команды. Найдите первые 4 байта расшифрованного вредоносного кода, если известно, что величина этого смещения не больше 250 байт.

Фрагмент кода программы после внедрения вируса:

```
…
db d5 c8 51 b8 fe 94 8b 89 d0 98 b5 b2 b1 d2 dd b1 d1 d6 cb dd ca cc 98 b5
b2 b1 db d5 c8 b1 d9 d4 94 8b 8ad0 98 b5 b2 b1 d2 ddb1 cb dd d9 ca db d0 9
8 b5 b2 b1 db d5 c8 b1 d9 d4 94 8b 8b d0 98 b5 b2 b1 d2 dd b1 dc dd d4 dd
cc dd 98 b5 b2 b1 db d5 c8 b1 d9 d4 94 8b 88 d0 98 b5 b2 b1 d2 ddb1 dd c0
d1 cc 98 b5 b2 b1 d2 d5 c8 b1 df d7 98
…
```

Комментарий. В Вашем распоряжении имеется бинарный файл «*virus.bin*», содержащий указанный фрагмент бинарного кода.

Решение.

Необходимо перебрать все константы от 0x00 до 0xff, выполнив при этом команду «побайтового исключающего ИЛИ» между каждым байтом кода программы и выбранной константой. Перебор осуществляется до тех пор, пока в результирующем бинарном коде не появятся подряд идущие байты «e9» и «00», которые соответствуют коду команды безусловного перехода *JMP* и первому байту смещения. За байтом «00» будет следовать второй, значащий байт со смещением. Необходимо отсчитать это смещение от текущей позиции и взять четыре байта кода начала вируса.

Ответ: 65 20 0d 0a

<u>Задача 3.</u> Протокол

Алексею необходимо передать Виктории пятисимвольный пароль к учетной записи на сайте. Для того, чтобы пароль не был перехвачен, Виктория предлагает использовать следующий способ:

1. Алексей преобразует пароль (параметр *psw*) с помощью приведенной ниже функции, используя при этом известный только ему ключ (параметр *key*). Полученную строку отправляет Виктории.

```
char * E(char psw[5], char key[5])
{
	char *res = new char[5];
	for(int i = 0 ; i<5 ; i++)
	{
		res[i] = (psw[i] + key[i])%256;
	}
	return res;
}
```

2. Виктория с помощью этой же функции преобразует полученную строку, указывая ее в качестве параметра *psw*, но используя свой ключ, известный только ей. Результат преобразования отправляется Алексею.

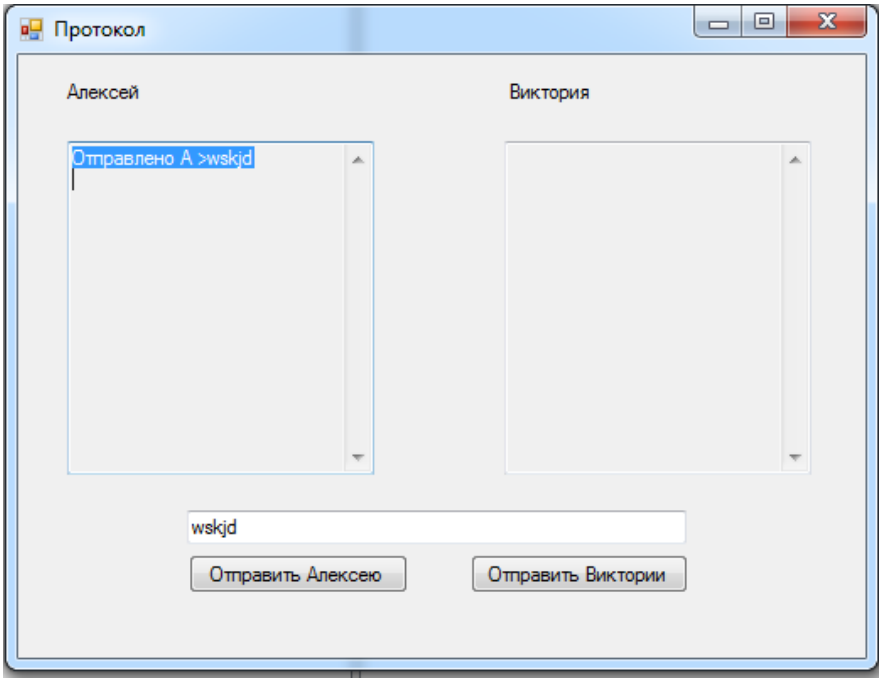
3. Алексей передает в функцию, приведенную ниже, в качестве параметров полученную от Виктории строку и свой исходный ключ:

```
char * D(char msg[5], char key[5])
{
	char *res = new char[5];
	for(int i = 0 ; i<5 ; i++)
	{
		res[i] = (msg[i] - key[i])%256;
	}
	return res;
}
```

4. Возвращаемое функцией значение отправляется Виктории, по которому она восстанавливает пароль.

Алексей отказался от предложения Виктории, сославшись на то, что если не обеспечить подтверждение подлинности абонентов, то нарушитель сможет узнать пароль при перехвате отправляемых по сети строк. Прав ли Алексей? Какой пароль передавался Виктории, если в первом сообщении была перехвачена посланная Алексеем строка ”wskjq”.

Комментарий. В Вашем распоряжении есть программа «Protocol.exe», моделирующая ситуацию, при которой нарушитель может перехватывать посылаемые сообщения. При помощи этой же программы Вы можете посылать любые сообщения Алексею от имени Виктории и Виктории от имени Алексея.



Решение.

Способ передачи пароля, предложенный Викторией, есть не что иное, как протокол Шамира-Ривеста-Адлемана передачи секретного ключа k от абонента A к абоненту B:

(1) A -> B : $E_{ka}(k)$

(2) A <- B: $E_{kb}(E_{ka}(k))$

(3) A -> B: $D_{ka}(E_{kb}(E_{ka}(k)))$,

где E – коммутирующее шифрующее преобразование: $E_{k1}(E_{k2}(x)) = E_{k2}(E_{k1}(x))$ при всех сообщениях x и для любых ключей k_1 и k_2, D – преобразование, которое расшифровывает шифрующее преобразование E.

Алексей прав, говоря о том, что если не обеспечить подтверждение подлинности абонентов, то нарушитель сможет узнать пароль при перехвате отправляемых по сети строк, что подтверждается наличием одной из известных слабостей протокола Шамира-Ривеста-Адлемана, которая основана на симметричности сообщений участников.

Использование атаки, реализующей описанную слабость данного протокола, может выглядеть следующим образом:

(1) A -> C(B) : $E_{ka}(k)$

(2) A <- C(B) : $E_{ka}(k)$

(3) A -> C(B) : k

Это позволяет ответить на второй вопрос задачи – для того, чтобы прочитать пароль, нарушителю достаточно осуществить доступ к сети от имени Виктории и повторить первое сообщение Алексея, а затем прочитать сообщение от Алексея. Это и будет словом, которое Алексей хотел передать Виктории. То есть, используя программу, необходимо сообщение «wskjq» передать Алексею от имени Виктории и прочитать ответ – «tocek». Это и есть пароль, который Алексей собирался передать Виктории.

Ответ: tocek.

Задача 4. Дешифрование

Текстовый файл «*encrypttext.txt*» был получен, применяя 2015 раз функцию *Encrypt* (см. листинг 1) к исходному файлу. Расшифруйте файл «*encrypttext.txt*» по крайней мере в 1000 раз быстрее, чем он был зашифрован.

```
char * ReadMassFromFile(ifstream &inFile, int &outN)                    //1
{
	char tempMass[10];                                                  //2
	char buff;                                                          //3
	int i(0);                                                           //4
	while (!inFile.eof() && (i<10))                                     //5
	{
		inFile.get(buff);                                           //6
		if (inFile) tempMass[i++] = buff;                           //7
	}
	char * outMass = new char [i];                                      //8
	for (int j = 0; j<i; j++) outMass[j] = tempMass[j];                 //9
	outN = i;                                                           //10
	return outMass;                                                     //11
}
void EncryptMass(char * Mass, int n)                                    //12
{
	if (n!=10) return;                                                  //13
	char temp[10];                                                      //14
	for (int i = 0; i < 10; i++)    temp[i] = Mass[i];                  //15
	for(int i=0;i<n;i++) Mass[(i*i*i+2)%10]=temp[i];                    //16
	return;                                                             //17
}
void WriteMassToFile(ofstream &outFile, char * mass, int n)             //18
{
	for (int i=0; i<n; i++) outFile.put(mass[i]);                       //19
	delete [] mass;                                                     //20
	return;                                                             //21
}
int Encrypt(char * inFile, char* outFile)                               //22
{
	ifstream openText(inFile);                                          //23
	ofstream encryptText(outFile);                                      //24
	if (!openText || !encryptText) return -1;                           //25
	while (!openText.eof())                                             //27
```

```
	{
		int n;                                                  //28
		char *buffMass = ReadMassFromFile(openText, n);         //29
		EncryptMass(buffMass, n);                               //30
		WriteMassToFile(encryptText,buffMass, n);               //31
	}
	return 1;                                                   //32
}
```

Листинг 1. Исходный код программы шифрования файла

Решение.

Рассмотрим функцию *Encrypt*(строки 22-32). Чтение блока данных перед шифрованием осуществляет функция *ReadMassFromFile*. В результате ее работы считывается блок данных длинной 10 байт из входного файла (строка 5). Полученный массив передается функции *EncryptMass*, которая преобразует его. Затем функция *WriteMassToFile* записывает преобразованные данные в выходной файл. Таким образом, непосредственное шифрование происходит в функции *EncryptMass,* которая осуществляет перестановку элементов массива. Новый индекс элемента с номером i вычисляется по формуле i^3+2 (строка 16). Эта формула задает следующее отображение

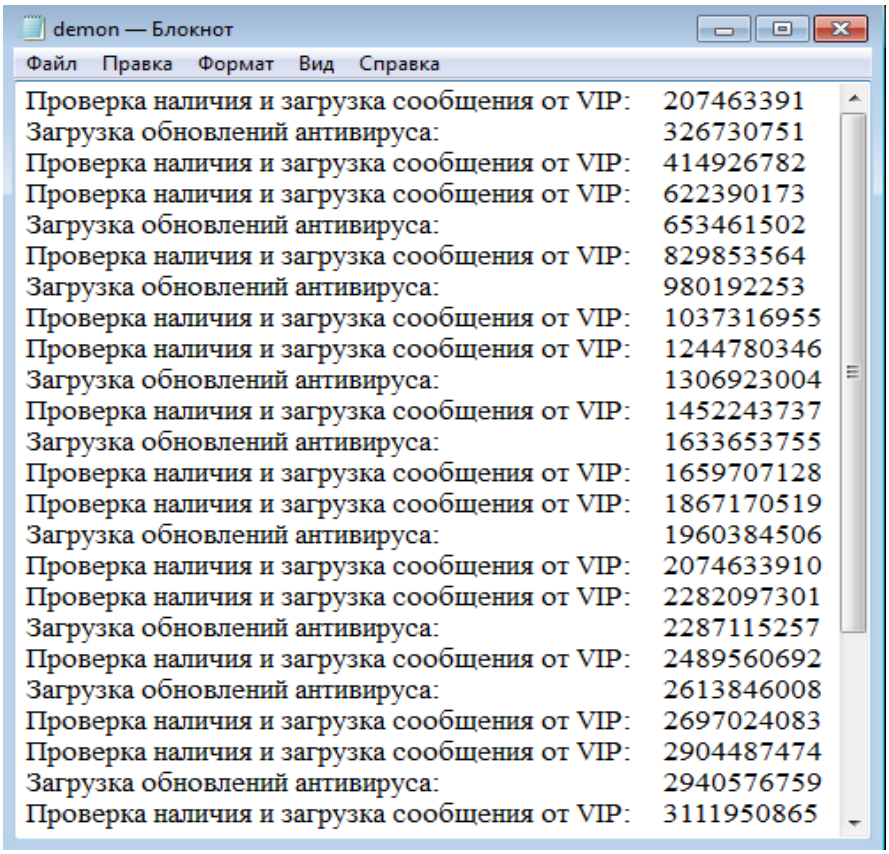
0	1	2	3	4	5	6	7	8	9
2	3	0	9	6	7	8	5	4	1

Рассмотрим, как действует это преобразование применённое несколько раз:
0→2→0 : цикл длинны 2;
1→3→9→1 : цикл длинны 3;
4→6→8→4: цикл длинны 3;
5→7→5: цикл длинны 2.
Таким образом, после применения этого преобразование 6 раз подряд будет получен начальный текст. По условию задачи, преобразование применялось 2015 раз. Так как 2015 = 6*335 + 5, то для получения открытого текста необходимо применить функцию *Encrypt* к файлу *encrypttext.txt* один раз.

Ответ: для получения открытого текста необходимо применить функцию *Encrypt* к файлу *encrypttext.txt* один раз

Задача 5. Антивирус

Нарушителю удалось получить журнал работы двух периодически запускающихся процессов сервера – обновления антивируса и проверки почтовых сообщений. Кроме того, он знает, что если обновление антивируса стартует во время загрузки почтовых сообщений от некоторого абонента VIP, то загружаемое сообщение антивирусом не проверяется. Из-за использования пароля 111для почтового ящика VIP, нарушителю удалось получить к нему доступ.



Опишите возможные действия нарушителя по внедрению на сервер вредоносного кода через почтовые сообщения от VIP. В какой минимальный момент времени может произойти внедрение вредоносного кода?

Решение.

Для внедрения на сервер вредоносного кода нарушитель может воспользоваться отсутствием проверки сообщений от VIP в момент загрузки обновлений антивируса.

Заметим, что «загрузка обновлений антивируса» запускается через один и тот же временной промежуток $T_A = 326730751$, а «проверка наличия сообщения» запускается через временной промежуток $T_S = 207463391$ от предыдущей проверки и загрузки. Тогда нетрудно выписать общий вид времени моментов запуска каждого из процессов:

$$T_A(k) = T_A \cdot k; \qquad T_S(n) = T_S \cdot n.$$

Тогда условие одновременного запуска обоих процессов равносильно уравнению:$T_A \cdot k = T_S \cdot n$.

Тогда первый одновременный запуск произойдет в момент времени равный наименьшему общему кратному $[T_A, T_S]$, вычислить данное значение удобно как произведение $T_A \cdot T_S$ деленное на наибольший общий делитель (НОД) этих чисел. На языке «С» следующим образом реализуем алгоритм Евклида нахождения НОД:

```
int Evclid(int n, int m)
{
	while ((n>0) && (m>0))
	{
```

```
            if (m>n)
                  m=m%n;
            else
                  n=n%m;
      }
      return (n+m);
}
```

Посчитанный НОД = 18181.

(*Дальнейшие вычисления возможны в калькуляторе Windows или используя переменную _int64.*)

Ответ: $\left(\frac{326730751}{18181}\right) * 207463391 = 3728324599661$