



Если рассматривать поле данных обнаруженных пакетов, то можно заметить следующую особенность: поле данных содержит одинаковую информацию («8»), но при этом для каждого пакета имеет различный размер.

По указанной особенности можно сделать предположение, что полезная информация передавалась путем формирования пакетов определенного размера. Если взять значения, которые необходимо добавить для выравнивания пакета до целого числа байт, то можно заметить, что в ASCII-таблице им соответствуют читаемые буквы. В результате анализа всех пакетов получаем сообщение из 5 символов с кодами 0x53, 0x77, 0x6F, 0x72, 0x64, что соответствует слову «*Sword*».

**Ответ:** «*Sword*»

## Задача 2. 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:

```
...
41 0d 61 01 60 44 69 48 24 28 60 24 2d 2d 41 04 4c 49 05 24 00 28 60 04 41
0d 61 48 4c 04 41 45 20 6c 40 20 20 29 6c 69 41 60 64 04 41 08 20 2c 49 05
2c 49 48 49 49 0d 20 64 49 68 25 84 6d 78 9d 98 68 60 60 28 60 60 68 60
04 20 29 60 24 2d 60 24 2d 01 24 c7 b4 d9 38 6c
...
```

**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 исключения 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:** aa d9 b4 55

### **Задача 3. 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.

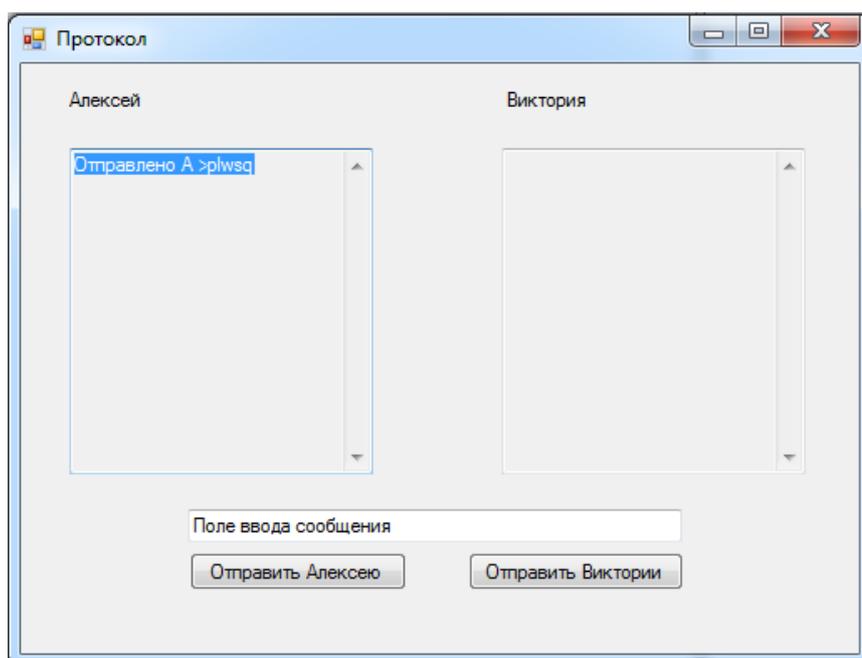
3. Алексей передает в функцию, приведенную ниже, в качестве параметров полученную от Виктории строку и свой исходный ключ:

```
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 "Iptwq".

*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)  $A \rightarrow B : E_{k_a}(k)$
- (2)  $A \leftarrow B : E_{k_b}(E_{k_a}(k))$
- (3)  $A \rightarrow B : D_{k_a}(E_{k_b}(E_{k_a}(k)))$ ,

где  $E$  – коммутирующее шифрующее преобразование:  $E_{k_1}(E_{k_2}(x)) = E_{k_2}(E_{k_1}(x))$  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)  $A \rightarrow C(B) : E_{ka}(k)$

(2)  $A \leftarrow C(B) : E_{ka}(k)$

(3)  $A \rightarrow C(B) : k$

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 «rlwsq» передать Алексею от имени Виктории и прочитать ответ – «mhonk». Это и есть пароль, который Алексей собирался передать Виктории.

**Ответ:** mhonk

#### **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*i*i+i*i*i+9*i+8)%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 исходного файла

**Решение.**

Рассмотрим функцию *Encrypt*(строки 22-32). Чтение блока данных перед шифрованием осуществляет функция *ReadMassFromFile*. В результате ее работы считывается блок данных длиной 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  $i^5+i^3+9i+8$  (строка 16). Эта формула задает следующее отображение

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

Рассмотрим, как действует это преобразование применённое несколько раз:

- 0→8→0 : цикл длинны 2;
- 9→7→1→9 : цикл длинны 3;
- 6→4→2→6: цикл длинны 3;
- 5→3→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 111 для почтового ящика VIP, нарушителю удалось получить к нему доступ. Сообщения от VIP загружаются со скоростью 1 Кбайт/сек, максимальный размер сообщения 100 Кбайт.

Событие	Время
Загрузка обновлений антивируса:	1111
Проверка наличия сообщения от VIP:	19591
Проверка наличия сообщения от VIP:	205664882
Загрузка обновлений антивируса:	317641362
Проверка наличия сообщения от VIP:	411310173
Проверка наличия сообщения от VIP:	616955464
Загрузка обновлений антивируса:	635281613
Проверка наличия сообщения от VIP:	822600755
Загрузка обновлений антивируса:	952921864
Проверка наличия сообщения от VIP:	1028246046
Начало загрузки сообщения (30Кб):	1028246046
Окончание загрузки сообщения (30Кб):	1028246076
Проверка наличия сообщения от VIP:	1233891367
Загрузка обновлений антивируса:	1270562115
Проверка наличия сообщения от VIP:	1439536688
Начало загрузки сообщения (70Кб):	1439536688
Окончание загрузки сообщения (70Кб):	1439536758
Загрузка обновлений антивируса:	1588202366
Проверка наличия сообщения от VIP:	1645182079
Проверка наличия сообщения от VIP:	1850827470
Загрузка обновлений антивируса:	1905842617
Проверка наличия сообщения от VIP:	2056472861
Загрузка обновлений антивируса:	2223482868

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  $T_A = 317640251$ , а «проверка наличия сообщения» запускается через временной промежуток  $T_S = 205645291$  от предыдущей проверки или окончания работы. Тогда нетрудно выписать общий вид времени моментов запуска каждого из процессов:

$$T_A(k) = T_A \cdot k + T_0;$$

$$T_S(n) = T_S \cdot n + T'_0 + (r_1 + r_2 + \dots + r_j);$$

где  $r_i$  –размер  $i$ -ого сообщения от абонента VIP.

Тогда условие одновременного запуска обоих процессов равносильно уравнению:

$$T_A \cdot k + T_0 = T_S \cdot n + T'_0 + (r_1 + r_2 + \dots + r_j)$$

или

$$T_A \cdot k - T_S \cdot n = T'_0 - T_0 + (r_1 + r_2 + \dots + r_j)$$

Левая часть уравнения обязательно делится нацело на наибольший общий делитель (НОД) чисел  $T_A$  и  $T_S$ , а тогда на этот же – НОД( $T_A, T_S$ ) должна делиться и правая часть.

С использованием алгоритма Евклида вычислим  $\text{НОД}(T_A, T_S) = 18181$ .  
 Программный код функции на языке «С», реализующей алгоритм Евклида:

```
int Evclid(int n, int m)
{
    while ((n>0) && (m>0))
    {
        if (m>n)
            m=m%n;
        else
            n=n%m;
    }
    return (n+m);
}
```

Теперь нарушителю необходимо от имени VIP отправлять такие сообщения  $r_i$ , что сумма  $T'_0 - T_0 + (r_1 + r_2 + \dots + r_j)$  дает остаток «0» при делении на  $\text{НОД}(T_A, T_S)$ . То есть отправить сообщения общего размера

$$\text{НОД}(T_A, T_S) - T'_0 + T_0 = 18181 - 19591 + 1111 = 299.$$

Тогда

$$\frac{T_A}{\text{НОД}(T_A, T_S)} \cdot k + \frac{T_S}{\text{НОД}(T_A, T_S)} \cdot n = 1;$$

$$17471 \cdot k + 11311 \cdot n = 1$$

Минимальное время, через которое время запуска процессов совпадет, найдем небольшим перебором с использованием следующей функции:

```
int MinKN(int a, int b)
{
    int k=1;
    while((a*k-1)%b!=0)
    {
        k++;
    }
    return (k);
}
```

Вообще говоря, перебор значения  $k$  не превысит значения  $\frac{T_S}{\text{НОД}(T_A, T_S)} = 11311$ . Найденное  $k = 2881$ ;

Тогда метка времени есть  $T_A \cdot k + T_0 = 915121564\ 242$

**Ответ:** Отправить 3 файла суммарного размера 299; метка времени, при которой возможна атака – 915 121 564 242.