Задача 1. Парольная комбинация

Для входа в систему используется пароль, состоящий из трёх двузначных чисел, расположенных следующим образом:

xx-xx-xx

Известно, что пароль состоит из *3-х неповторяющихся простых чисел*. При этом, *последняя цифра первого числа равна первой цифре второго числа*, а *последняя цифра второго числа равна первой цифре третьего числа*.

Пример:

x1-17-7y

Задержка между попытками входа в систему равна 1 секунде. За какое *минимальное* время (в секундах) можно *гарантированно* получить пароль, если на ввод пароля время не тратится, и количество попыток ввода пароля не ограничено?

Решение

Для начала необходимо найти все простые числа в диапазоне от 11 до 99 (двузначные числа). *Простым* является такое число, которое делится только на себя и на 1. Сделать это можно перебором, проверяя, не делится ли проверяемое число на какое-либо от 2-х до самого числа (либо половины числа).

Для ускорения напишем программу, которая выводит на экран все простые числа в указанном диапазоне.

Листинг программы на языке Python.

```
# Функция, определяющая, является ли число простым
# ПАРАМЕТР:
#  n - проверяемое число
# ВОЗВРАЩАЕТ:
#  True - число n простое
#  False - число n непростое
def isprime(n):
    if n == 1:
        return True
    for d in range(2, n//2):
        if n % d == 0:
            return False
    return True

# Цикл перебора чисел от 11 до 99
# Для каждого числа вызывается функция isprime()
# Если функция вернула True - число добавляется
# в массив list_input
# Счетчик Count считает количество простых чисел
Count = 0
list_input = []
for x in range(11,100):
    if isprime(x) == True:
        list_input.append(x)
        Count += 1
```

```
print(list_input)
print("Total:", Count)
```

Листинг программы на языке C.

```
// Функция, определяющая, является ли число простым
// ПАРАМЕТР:
//  n - проверяемое число
// ВОЗВРАЩАЕТ:
//  true - число n простое
//  false - число n непростое
bool isprime(int n)
{
   if (n == 1)
        return true;
   for (int d = 2; d < n / 2; d++)
        if (n % d == 0)
             return false;
   return true;
}

int main()
{
// Цикл перебора чисел от 11 до 99
// Для каждого числа вызывается функция isprime()
// Если функция вернула true - число добавляется
// в массив
// Счетчик Count считает количество простых чисел
    int Count = 0;
    int list_input[100] = { 0 };
    // Формирование массива простых чисел
    for (int x = 11; x < 100; x++)
    {
        if (isprime(x) == true)
        {
            list_input[Count] = x;
            Count += 1;
        }
    }
    // Вывод на экран
    for (int i = 0; i < Count; i++)
    {
        printf("%d ", list_input[i]);
    }
    printf("\nTotal: %d\n", Count);
    return 0;
}
```

Результат работы программы:

```
11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
Total: 21
```

Всего 21 простых чисел в диапазоне от 11 до 99. Теперь необходимо выделить все возможные комбинации, удовлетворяющие условию «*последняя цифра первого числа равна первой цифре второго числа*, а *последняя цифра второго числа равна первой цифре третьего числа*».

Сделать это можно перебором всех комбинаций. В цикле перебираем всевозможные числа и проверяем два условия:

- все числа разные;
- последняя цифра первого числа равна первой цифре второго числа, а последняя цифра второго числа равна первой цифре третьего числа.

Если все условия выполнены, то увеличиваем значение счетчика. В результате счетчик будет содержать количество комбинаций, удовлетворяющих заданию. Поскольку задержка между вводом паролей равна 1 секунде, а после ввода последней комбинации задержки нет, то ответ – количество найденных комбинаций минус 1.

Пример программы представлен в листингах ниже.

Листинг программы на языке Python.

```
# Функция возвращает список всех комбинаций чисел,
# удовлетворяющих заданному условию паролей
# ПАРАМЕТР:
#    list_input - список чисел, из которых будет
#                 формироваться комбинация
# ВОЗВРАЩАЕТ:
#    list_combination - список комбинаций
#             из 3-х чисел, удовлетворяющих условию
def combination(list_input):
  list_combination = []
  for a in list_input:
    for b in list_input:
      for c in list_input:
        if a != b and b != c and a != c:
          if str(a)[-1] == str(b)[0] and
             str(b)[-1] == str(c)[0]:
            list_combination.append(str(a) +
                    "-" + str(b) + "-" + str(c))
  return list_combination

# Заполнение списка всеми простыми числами
# в заданном диапазоне
list_input = []
for x in range(11,100):
    if isprime(x) == True:
        list_input.append(x)
# Список, содержащий все комбинации,
# удовлетворяющие заданному условию
result_list = combination(list_input)
# Вывод всех комбинаций
for i in result_list:
    print(i)
# Вывод количества комбинаций
print("Total:", len(result_list))
```

Листинг программы на языке C.

```
// Функция выводит на экран все комбинации,
// удовлетворяющие условию
// ПАРАМЕТРЫ:
```

```
   //    list_input - массив чисел, из которых будут
   //                 строится комбинации
   //    count - количество чисел в массиве list_input
   // ВОЗВРАЩАЕТ:
   //    выводит на экран комбинации, удовлетворяющие
   //    условию
   //  выводит на экран количество комбинаций
   void combination(int list_input[], int count)
   {
      int last_digit1;
      int first_digit2;
      int last_digit2;
      int first_digit3;
      int total = 0;
      for (int i = 0; i < count; i++)
           for (int j = 0; j < count; j++)
                for (int k = 0; k < count; k++)
                {
                     if ( list_input[i] != list_input[j] &&
                          list_input[j] != list_input[k] &&
                          list_input[i] != list_input[k] )
                     {
                          // последняя цифра числа
                          last_digit1 = list_input[i] % 10;
                          // первая цифра числа
                          first_digit2 = list_input[j] / 10;
                          last_digit2 = list_input[j] % 10;
                          first_digit3 = list_input[k] / 10;
                          if ( last_digit1 == first_digit2 &&
                               last_digit2 == first_digit3 )
                          {
                               printf("%d-%d-%d\n",
                                         list_input[i],
list_input[j],
                                         list_input[k]);
                               total++;
                          }
                     }
                }
      // Вывод общего количества комбинаций
      printf("Total: %d\n", total);
   }
```

В результате выполнения программы получим следующие данные:

11-13-31, 11-13-37, 11-17-71, 11-17-73, 11-17-79, 11-19-97, 13-31-11, 13-31-17, 13-31-19, 13-37-71, 13-37-73, 13-37-79, 17-71-11, 17-71-13, 17-71-19, 17-73-31, 17-73-37, 17-79-97, 19-97-71, 19-97-73, 19-97-79, 23-31-11, 23-31-13, 23-31-17, 23-31-19, 23-37-71, 23-37-73, 23-37-79, 29-97-71, 29-97-73, 29-97-79, 31-11-13, 31-11-17, 31-11-19, 31-13-37, 31-17-71, 31-17-73, 31-17-79, 31-19-97, 37-71-11, 37-71-13, 37-71-17, 37-71-19, 37-73-31, 37-79-97, 41-11-13, 41-11-17, 41-11-19, 41-13-31, 41-13-37, 41-17-71, 41-17-73, 41-17-79, 41-19-97, 43-31-11, 43-31-13, 43-31-17, 43-31-19, 43-37-71, 43-37-73, 43-37-79, 47-71-11, 47-71-13, 47-71-17, 47-71-19, 47-73-31, 47-73-37, 47-79-97, 53-31-11, 53-31-13, 53-31-17, 53-31-

```
19, 53-37-71, 53-37-73, 53-37-79, 59-97-71, 59-97-73, 59-97-79,
61-11-13, 61-11-17, 61-11-19, 61-13-31, 61-13-37, 61-17-71, 61-17-
73, 61-17-79, 61-19-97, 67-71-11, 67-71-13, 67-71-17, 67-71-19,
67-73-31, 67-73-37, 67-79-97, 71-11-13, 71-11-17, 71-11-19, 71-13-
31, 71-13-37, 71-17-73, 71-17-79, 71-19-97, 73-31-11, 73-31-13,
73-31-17, 73-31-19, 73-37-71, 73-37-79, 79-97-71, 79-97-73, 83-31-
11, 83-31-13, 83-31-17, 83-31-19, 83-37-71, 83-37-73, 83-37-79,
89-97-71, 89-97-73, 89-97-79, 97-71-11, 97-71-13, 97-71-17, 97-71-
19, 97-73-31, 97-73-37
  Total: 126
```

Всего таких комбинаций будет 126, следовательно, максимальное затраченное время будет равно

$$(126 - 1) * 1 \text{ сек} = 125 \text{ сек}.$$

Ответ: минимальное время, за которое можно гарантированно получить пароль, равно **125 секунд.**

Задача 2. Сетевой трафик

Был получен фрагмент сетевого трафика пользователя при взаимодействии с игровым сервером. Известно, что сервер работает по протоколу UDP и его порт назначения равен 8229. Структура UDP-дейтаграммы представлена ниже:

2 байта	2 байта	2 байта	2 байта	…
UDP-порт отправителя	UDP-порт получателя	Длина UDP-дейтаграммы	Контрольная сумма	Данные

Длина UDP-дейтаграммы включает в себя размер заголовка и размер данных в байтах.

Дамп трафика:

```
  0B D7 20 25 00 16 1D DC 47 45 54 20 43 4F 4D 4D 41 4E 44 3A 20
25 20 25 0B D7 00 12 69 AF 53 45 54 20 43 4F 4F 52 44 3A 20 25 0B
D7 00 12 25 C0 20 25 28 33 34 2C 35 34 29 00 0B D7 20 25 00 14 2C
8F 43 4F 4D 4D 41 4E 44 20 2D 20 4F 4B
```

Определите, какие данные *сервер отправил клиенту*.

Решение

Из условия задачи известно, что взаимодействие происходит между клиентом и сервером. Для адресации используется поле UDP-порт отправителя и UDP-порт получателя (на каждый из них выделяется по 2 байта). При этом, если порт сервера равен 8229, то в дейтаграммах, отправленных клиентом, поле «UDP-порт получателя» будет равно 8229. В дейтаграммах, отправленных сервером, поле «UDP-порт отправителя» будет равно 8229.

Число $8229_{10} = 2025_{16}$.

Анализируя трафик, можно заметить, что 3-й и 4-й байты равны «2025» – это поле «UDP-порт получателя». Значит, первая дейтаграмма отправлена клиентом серверу. Необходимо найти размер этой дейтаграммы (5-6 байты):

$$0016_{16} = 22_{10}.$$

Следующая дейтаграмма начинается с 23-го байта:

```
2025 0BD7 0012 69AF …
```

Как видно, 1-2 байты дейтаграммы равны 20 25 – это поле «UDP-порт отправителя». Это означает, что данная дейтаграмма была отправлена сервером клиенту. Размер данной дейтаграммы: $0012_{16} = 18_{10}$.

Вся дейтаграмма выглядит следующим образом:

```
2025 0BD7 0012 69AF 53 45 54 20 43 4F 4F 52 44 3A
```

Поле данных (10 байт): `53 45 54 20 43 4F 4F 52 44 3A`.

В соответствии с ASCII-таблицей, каждому байту соответствует символ. Получаем следующую строку:

SET COORD:

Анализируя дальнейший дамп трафика, можно заметить, что следующая дейтаграмма также начинается с байтов 2025, это означает, что следующая дейтаграмма также отправлена сервером клиенту:

```
2025 0BD7 0012 25C0 20 25 28 33 34 2C 35 34 29 00
```

Размер дейтаграммы равен $0012_{16} = 18_{10}$.

Поле данных (10 байт): `20 25 28 33 34 2C 35 34 29 00`

В соответствии с ASCII-таблицей, каждому байту соответствует символ. Получаем следующую строку:

_%(34,54)

Следующая дейтаграмма:

```
0BD7 2025 0014 2C8F 43 4F 4D 4D 41 4E 44 20 2D 20 4F 4B
```

Поле «UDP-порт назначения» равно 2025 – эта дейтаграмма отправлена клиентом серверу. Размер дейтаграммы $0014_{16} = 20_{10}$.

Больше дейтаграмм в дампе нет.

Таким образом, от сервера клиенту было отправлено две дейтаграммы со следующим содержимым:

SET COORD:_%(34,54)

Ответ: SET COORD:_%(34,54).

Задача 3. Шифрование

В системе используется следующий алгоритм шифрования текстовых сообщений: значение каждого следующего байта циклически сдвигается побитно влево *N* раз, где *N* – значение предыдущего зашифрованного байта. Первый байт сообщения не шифруется.

Расшифруйте предоставленный зашифрованный фрагмент текста:

```
53 2B 73 23 01 C2 D5 8E 1A 80 72 95 2E 5D AC 37 3A
```

Решение

Приведенный в задании алгоритм подразумевает, что первый байт не шифруется – $53_{16} = 83_{10}$, что соответствует символу '*S*'.

Следующий символ был зашифрован путем циклического сдвига влево 83 раза. Стоит отметить, что если значение байта циклически сдвинуть влево 8 раз, получится исходное значение байта. Таким образом, сдвиг влево 83 раза равен сдвигу влево 3 раза (83 % 8 = 3, где % – остаток от деления).

Для того, чтобы получить исходное значение байта, сдвинутого влево 3 раза, необходимо его еще сдвинуть влево 8 – 3 = 5 раз (см. таблицу).

Сдвиг	Значение байта в 2-ом формате	Значение байта в 16-ом формате
0	00101011	2B
1	01010110	56
2	10101100	AC
3	01011001	59
4	10110010	B2
5	01100101	65

Сверившись с ASCII-таблицей, коду 65_{16} соответствует символ '*e*'.

Следующий байт был сдвинут влево $2B_{16} = 43_{10}$ раз, что эквивалентно сдвигу влево 3 раза (43 % 8 = 3).

Чтобы восстановить значение байта, необходимо сдвинуть его влево 5 раз.

Сдвиг	Значение байта в 2-ом формате	Значение байта в 16-ом формате
0	01110011	73
1	11100110	E6
2	11001101	CD
3	10011011	9B
4	00110111	37
5	01101110	6E

Сверившись с ASCII-таблицей, коду $6E_{16}$ соответствует символ '*n*'.

Для ускорения процесса расшифрования можно написать программу. Листинги программы представлены ниже.

Листинг программы на языке Python.

```
# Функция циклического сдвига влево
# ПАРАМЕТРЫ:
#    s - число (байт)
#    n - сколько раз сдвигать число s
# РЕЗУЛЬТАТ:
#    res - циклически сдвинутое влево число s n раз
def shift(s, n):
    res = int(s)
    for i in range(0, n):
        t = res
```

```
            # сдвиг влево на 1 бит
            res = (t << 1) & 0xFF
            # на место младшего бита записываем старший,
            # который был до сдвига
            res = res | (t >> 7)
        return res

    # Функция расшифрования массива чисел
    # ПАРАМЕТР:
    #    cmass - массив чисел
    # ВОЗВРАЩАЕТ:
    #    расшифрованную строку
    def decipher(cmass):
        # Результирующий текст
        text = ""
        # Копируем первый символ без изменений
        text += chr(cmass[0])
        # Цикл по оставшимся числам
        for i in range(1, len(cmass)):
            # Сдвигаем очередной символ столько раз,
            # чтобы суммарный сдвиг с учетом шифрования
            # был кратен 8
            n = 8 - (cmass[i-1] % 8)
            s = shift(cmass[i], n)
            # s преобразуем в символ
            # и добавляем в конец строки результата
            text += chr(s)
        return text

    mass = [0x53, 0x2B, 0x73, 0x23, 0x01, 0xC2, 0xD5, 0x8E, 0x1A, 0x80,
0x72, 0x95, 0x2E, 0x5D, 0xAC, 0x37, 0x3A]
    text = decipher(mass)
    print(text)
```

Листинг программы на языке C.

```
    // Функция циклического сдвига влево
    // ПАРАМЕТРЫ:
    //  s - число(байт)
    //  n - сколько раз сдвигать число s
    // РЕЗУЛЬТАТ:
    //  res - циклически сдвинутое влево число s n раз
    char shift(unsigned char a, int n)
    {
       char res = a;
       for (int i = 0; i < n; i++)
       {
            res = a << 1;
            res |= a >> 7;
            a = res;
       }
       return res;
    }

    // Функция расшифрования массива чисел
    // ПАРАМЕТР:
    //    cmass - массив чисел
    //    len - размер массива
    // ВОЗВРАЩАЕТ:
    //    расшифрованную строку
```

```
char* decipher(unsigned char* cmass, int len)
{
   int n;
   // выделение памяти
   char* text = new char[len+1];
   // первый элемент без изменений
   text[0] = cmass[0];
   // цикл по остальным числам из cmass
   for (int i = 1; i < len; i++)
   {
         // Сдвигаем очередной символ столько раз,
         // чтобы суммарный сдвиг с учетом шифрования
         // был кратен 8
         n = 8 - (cmass[i-1] % 8);
         text[i] = shift(cmass[i], n);
   }
   // добавляем к text нуль-символ конца строки
   text[len] = '\0';
   return text;
}
```

При подаче на вход программы последовательности из задания, получится следующий результат:

Send auth request

Данная фраза и есть ответ.

Ответ: Send auth request.

Задача 4. Стеганография

Аналитику удалось обнаружить папку с графическими изображениями, в которой скрыто осмысленное кодовое слово. Помогите определить кодовое слово, если известно, что для его сокрытия содержимое файлов не менялось.

К задаче прилагается: папка с файлами (1.jpg, 2.jpg, … , 32.jpg) (см. рисунок).

Рисунок. Прилагаемые изображения

Решение

Содержимое изображений не изменялось. Следовательно, содержимое файлов детально изучать нецелесообразно.

Все изображения представляют собой различные смайлики. Внимательно просмотрев все изображения, можно заметить, что изображенные смайлики либо грустные, либо веселые. Предположим, что каждая из картинок – бит информации, тогда можно соотнести одну категорию с единицей, а вторую – с нулем.

Всего 32 изображения – получается комбинация из 32-х нулей и единиц (32 бита).

Зная, что в байте 8 бит, а также тот факт, что каждому символу в ASCII-таблице соответствует 1 байт, можно предположить, что в картинках скрыто 4-символьное сообщение.

Предположим, что грустному смайлику соответствует «0», а веселому – «1». Получается следующая комбинация:

```
1000 1000
1001 1110
1001 0110
1000 1011
```

Переведем в 16-й формат: `88 9E 96 8B`.

Согласно ASCII-таблице, эти коды не соответствуют ни буквам, ни цифрам. Такое сообщение не несет смысл. Значит, исходное предположение было неверным.

Предположим теперь наоборот: грустному смайлику соответствует «1», а веселому – «0». Получается следующая комбинация:

```
0111 0111
0110 0001
0110 1001
```

0111 0100

Переведем в 16-й формат: 77 61 69 74.

Согласно ASCII-таблице, эти коды соответствуют следующим символам: *wait* – что является осмысленным словом и ответом на задачу.

Ответ: wait.

Задача 5. Web-сайт

На компьютере нарушителя было найдено множество архивов, каждый из которых защищён некоторым паролем. Проведённый анализ показал, что только *три* архива содержат полезную информацию: *один* содержит «скрытое сообщение», а *два других* – фрагменты пароля к этому архиву. Остальные архивы пустые и не содержат важной информации.

Всю необходимую информацию для доступа к архивам и секретному сообщению нарушитель спрятал на Web-странице. Определите «скрытое сообщение».

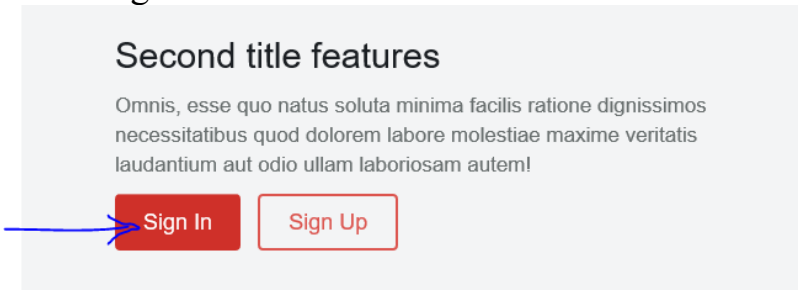
К задаче прилагается:

1) *папка с содержимым web-страницы*,
2) *папка с архивами (архив1.rar, архив2.rar, … , архив30.rar).*

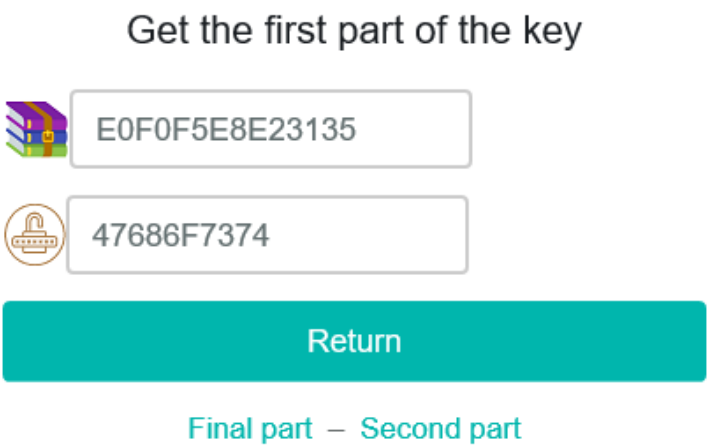
Решение

Обращаем внимание, что среди архивов ровно 3 имеют отличные от остальных дату и время модификации: *архив8.rar*, *архив15.rar*, *архив21.rar*.

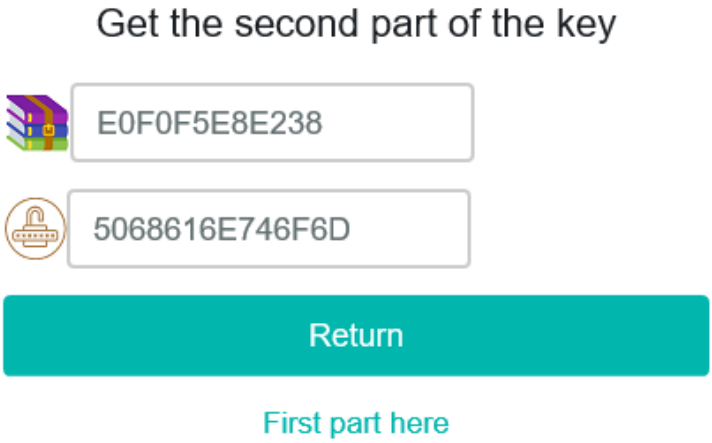
При анализе содержимого web-страницы обращаем внимание, что при переходе по ссылке «Sign-In»



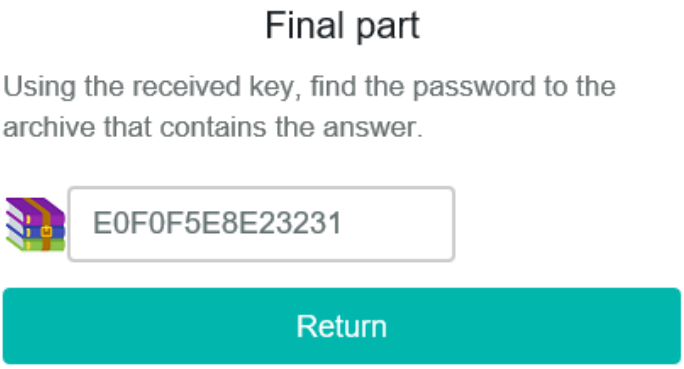
открывается форма, в который поля, обычно предназначенные для ввода логина и пароля, заполнены какой-то полезной информацией.



Перейдя по ссылкам «Second Part», получаем похожую картину:



Переход по ссылке «Final part» даёт информацию только об имени архива:



Исходя из условия задачи, можно предположить, что первые две части относятся к архивам, содержащим фрагменты пароля к финальному архиву с искомым «скрытым сообщением».

Проанализируем полученную информацию.

На первой форме (SignIn) содержится 2 сообщения:

- `0xE0F0F5E8E23135 (E0 F0 F5 E8 E2 31 35)`, что по ASCII-таблице соответствует строке «*архив15*»,

- в поле «пароль» указан пароль `0x47686F7374 (47 68 6F 73 74)`, то соответствует строке «*Ghost*».

Можно предположить, что к файлу архив15.rar паролем является Ghost (с учетом регистра). В архиве содержится файл с изображением первой части пароля от архива со «скрытым сообщением»: *World*.

На второй форме (Second Part) так же содержатся 2 сообщения:

- `0xE0F0F5E8E238` = «*архив8*»,
- `0x5068616E746F6D` = «*Phantom*», что является паролем к файлу архив8.rar.

В архиве8 содержится файл с изображением второй части пароля от архива со «скрытым сообщением»: *Map*.

На третьей форме (Final Part) содержится только одно сообщение: `0xE0F0F5E8E23231` = «*архив21*». Можно сделать вывод, что ответ содержится в файле архив21.rar, пароль к которому «складывается из двух частей» – *WorldMap*.

В архиве содержится файл, внутри которого можно найти «скрытое сообщение» – «*Sense of security*», которое и является ответом.

Ответ: Sense of security