Задачи с решениями
(11 класс)

Задача 1. Гербы

Вариант 1

Аналитику удалось обнаружить папку с графическими изображениями и текстовым файлом. Известно, что в изображениях скрыто кодовое слово. Помогите определить кодовое слово, если известно, что для его сокрытия изменили содержимое всех файлов.

К задаче прилагается:

1) 6 файлов-изображений (**.jpg*),

2) текстовый файл bytes.txt.

Решение

Файлы с изображениями содержат гербы городов России (рисунок 1.1-1):

1 – Ульяновск

2 – Казань

3 – Владивосток

4 – Москва

5 – Тула

6 – Рязань

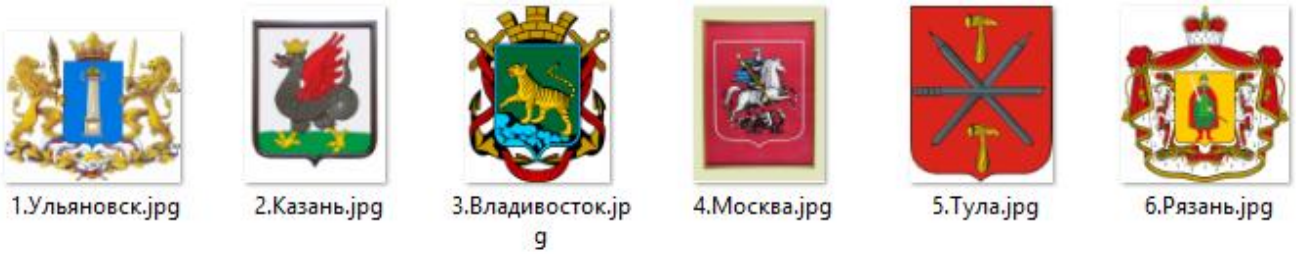

Рисунок 1.1-1 – Изображения гербов городов России

В представленном текстовом файле содержится следующая информация (рисунок 1.1-2):

1 - 4388, 10086

2 - 373B

3 - 5941B, 5941C

4 - 1B1DD, 23167, 33129

5 - 558A

6 - D9FBA

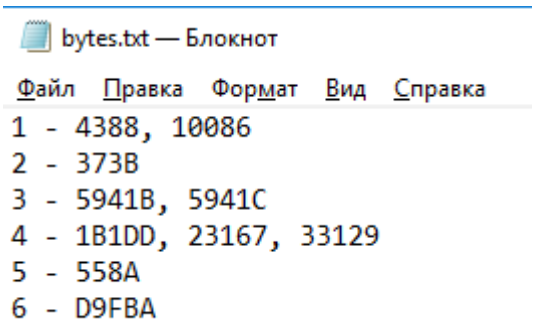

Рисунок 1.1-2 – Содержимое файла bytes.txt

Можно сделать предположение, что поскольку имеется 6 файлов с изображениями и в текстовом файле представлено 6 строчек, то каждая строчка в файле bytes.txt относится к соответствующему изображению.

Второе предположение – числа, записанные в строчках файла bytes.txt могут являться номерами (адресами) байт в соответствующих файлах. Проверим это предположение на примере первого файла – *1.Ульяновск.jpg*. В строке 1 указаны 2 числа: *4388, 10086*. С учетом остального

содержимого файла можно предположить, что эти числа в 16-ой системе счисления. Посмотрим содержимое байтов с адресами `0x00004388` и `0x00010086` (рисунок 1.1-3). Воспользуемся для этого приложением HexEditor.

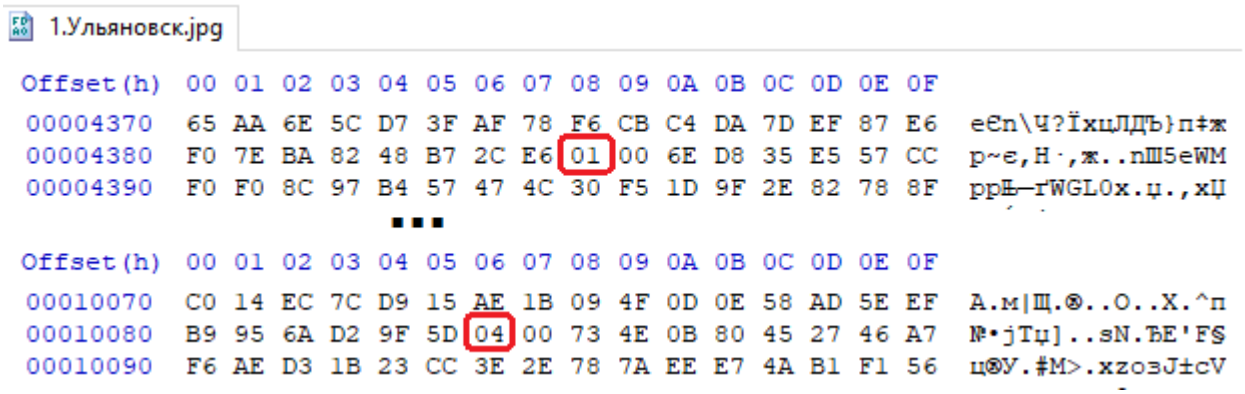

Рисунок 1.1-3 – Содержимое файла 1.Ульяновск.jpg

Байту с номером 4388 соответствует значение «01».
Байту с номером 10086 соответствует значение «04».

Аналогичным образом рассмотрим содержимое байтов и для остальных файлов.
Для файла 1.Ульяновск.jpg – значение «01» и «04».
Для файла 2.Казань.jpg – значение «03».
Для файла 3.Владивосток.jpg – значения «01» и «05».
Для файла 4.Москва.jpg – значения «01», «02» и «03».
Для файла 5.Тула.jpg – значение «01».
Для файла 6.Рязань.jpg – значение «06».

Из полученных значений можно сделать вывод: числа в разных файлах повторяются, но их значения не превышают 6. На коды символов ASCII-таблицы это непохоже. На номера букв в алфавите тоже – зачем тогда разделение на файлы.

Можно сделать предположение, что каждое значение – это номер буквы в имени файла, а конкретно, в названии города. Получается:

Ульяновск	–	01,04	–	УЯ
Казань	–	03	–	З
Владивосток	–	01,05	–	ВИ
Москва	–	01,03,03	–	МОС
Тула	–	01	–	Т
Рязань	–	06	–	Ь

В результате получается слово – УЯЗВИМОСТЬ.

Ответ: УЯЗВИМОСТЬ.

Вариант 2

Аналитику удалось обнаружить папку с графическими изображениями и текстовым файлом. Известно, что в изображениях скрыто кодовое слово. Помогите определить кодовое слово, если известно, что для его сокрытия изменили содержимое всех файлов.

К задаче прилагается:
1) 6 файлов-изображений (.png),*
2) текстовый файл bytes.txt.

Решение

Файлы с изображениями флагов городов Европы (рисунок 1.2-1):

1 – Берлин
2 – Запорожье
3 – Париж
4 – Хельсинки
5 – Осло
6 – Стокгольм

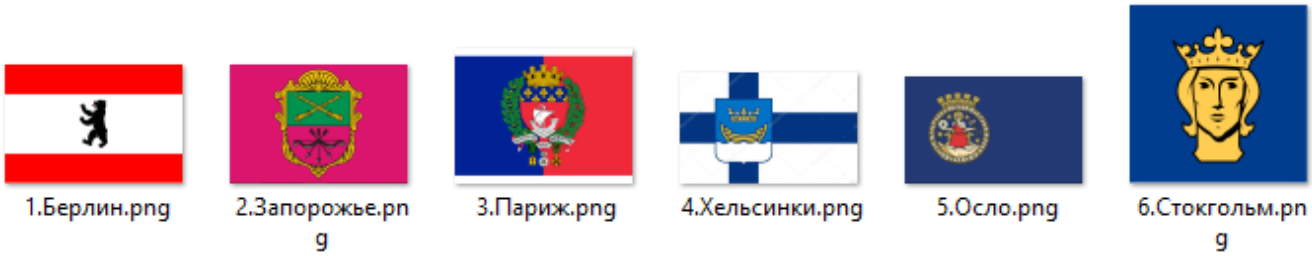

Рисунок 1.2-1 – Изображения гербов городов России

В представленном текстовом файле содержится следующая информация (рисунок 1.2-2):

1 - 66AD, 66BC
2 - 28B75, 28BA8
3 - 12D, 21F
4 - 29AB7, 453EA
5 - 85BD7, 85BF2
6 - 2B4C8, 2B504

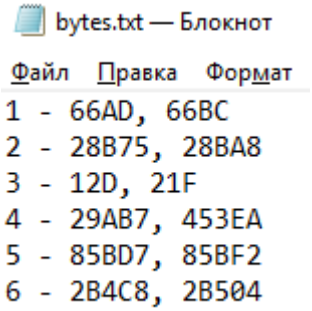

Рисунок 1.2-2 – Содержимое файла bytes.txt

Можно сделать предположение, что поскольку имеется 6 файлов с изображениями и в текстовом файле представлено 6 строчек, то каждая строчка в файле bytes.txt относится к соответствующему изображению.

Второе предположение – числа, записанные в строчках файла bytes.txt могут являться номерами (адресами) байт в соответствующих файлах. Проверим это предположение на примере первого файла – *1.Берлин.png*. В строке 1 указаны 2 числа: *66AD, 66BC*. С учетом остального содержимого файла можно предположить, что эти числа в 16-ой системе счисления. Посмотрим содержимое байтов с адресами `0x000066AD` и `0x000066BC` (рисунок 1.2-3). Воспользуемся для этого приложением HexEditor.

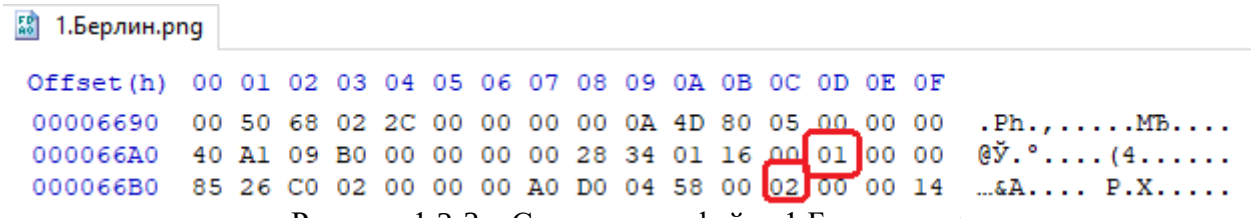

Рисунок 1.2-3 – Содержимое файла 1.Берлин.png

Байту с номером 66AD соответствует значение «01».
Байту с номером 66BC соответствует значение «02».

Аналогичным образом рассмотрим содержимое байтов и для остальных файлов.
Для файла 1.Берлин.png – значение «01» и «02».
Для файла 2.Запорожье.png – значение «01» и «04».
Для файла 3.Париж.png – значение «01» и «02».
Для файла 4.Хельсинки.png – значение «05» и «07».
Для файла 5.Осло.png – значение «01» и «02».
Для файла 6.Стокгольм.png – значение «02» и «08».

Из полученных значений можно сделать вывод: числа в разных файлах повторяются, но их значения не превышают 8. На коды символов ASCII-таблицы это непохоже. На номера букв в алфавите тоже – зачем тогда разделение на файлы.

Можно сделать предположение, что каждое значение – это номер буквы в имени файла, а конкретно, в названии города. Получается:

Берлин	–	01,02	–	БЕ
Запорожье	–	01,04	–	ЗО
Париж	–	01,02	–	ПА
Хельсинки	–	05,07	–	СН
Осло	–	01	–	О
Стокгольм	–	02,08	–	ТЬ

В результате получается слово – БЕЗОПАСНОСТЬ.

Ответ: БЕЗОПАСНОСТЬ.

Задача 2. Аутентификация

Вариант 1

Система аутентификации шифрует пароли особым образом, показанном в виде функции `scrambler` на языке C++. Зная алгоритм шифрования, вычислите пароль. Фрагмент кода указан ниже.

C++

```
int swap(int value, int start, int len)
{
  len = len % (sizeof(value) * 8);
  start = start % (sizeof(value) * 8);

  int bits = (INT_MAX >> ((sizeof(value) * 8) - len - 1));
  bits = bits << start;

  int buf = (value >> len) & bits;
  int res = value & bits;
  res = res << len;
  res |= buf;

  int rest = value & ~bits;
  rest = rest & ~(bits << len);

  return rest | res;
}
```

```
//////////////////////////////////////////////////
void scrambler(int keyword)
{
  int res = keyword;
  for (int i = 1; i < 16; i = i * 2)
  {
    for (int j = 0; j < 32 - i; j = j + i * 2)
      res = swap(res, j, i);
  }

  if (res == 1268560121)
    std::cout << "Password is correct\n";
  else
    std::cout << "Password is wrong\n";
}
```

Решение

Разберем представленный код и перепишем его построчно.

1.	`int swap(int value, int start, int len)`
2.	`{`
3.	`  len = len % (sizeof(value) * 8);`
4.	`  start = start % (sizeof(value) * 8);`
5.	`  int bits = (INT_MAX >> ((sizeof(value) * 8) - len - 1));`
6.	`  bits = bits << start;`
7.	`  int buf = (value >> len) & bits;`
8.	`  int res = value & bits;`
9.	`  res = res << len;`
10.	`  res |= buf;`
11.	`  int rest = value & ~bits;`
12.	`  rest = rest & ~(bits << len);`
13.	`  return rest | res;`
14.	`}`
15.	`void scrambler(int keyword)`
16.	`{`
17.	`  int res = keyword;`
18.	`  for (int i = 1; i < 16; i = i * 2)`
19.	`  {`
20.	`    for (int j = 0; j < 32 - i; j = j + i * 2)`
21.	`      res = swap(res, j, i);`
22.	`  }`
23.	`  if (res == 1268560121)`
24.	`    std::cout << "Password is correct\n";`
25.	`  else`
26.	`    std::cout << "Password is wrong\n";`
27.	`}`

Функция `scrambler()` проверяет число (`keyword`). Если число правильное (строка 23), то выводится на экран сообщение *«Password is correct»* (строка 24), иначе – выводится на экран сообщение «*Password is wrong*» (строка 25). Проверка правильности числа осуществляется с

помощью функции swap() (строки1-14) и значение этой функции сравнивается с константой 1268560121 (строка 23).

Существует 2 способа решения задачи:

1) Перебор.
2) Аналитический.

Способ 1.

Перебору будет подвергаться параметр функции `scrambler()`: от 0 до максимального значения типа данных `int` (`MAX_INT`).

Для автоматизации перебора необходимо модифицировать саму функцию `scrambler()`:

1) Сделать так, чтобы функция возвращала результат, например, логического типа (`bool`). Для этого необходимо заменить объявление функции (строка 15) на:

```
bool scrambler(int keyword)
```

2) Заменить вывод на экран оператором возвращения результата:
   а) строку 24 заменить на `return true;`
   б) строку 26 заменить на `return false;`

Пример реализации самого цикла перебора представлен в листинге 2.1-1.

Листинг 2.1-1 – Реализация цикла перебора параметра функции scrambler() на языке программирования C++

```
int main()
{
  // начальное время (в мс)
  int startTime = GetTickCount();

  // основной цикл перебора
  for (int i = 0; i < INT_MAX; i++)
    if (scrambler(i) == true)
    {
      // вывод результата на экран
      std::cout << "Result: " << i << std::endl;
    }

  // время завершения цикла (в мс)
  int finishTime = GetTickCount();

  // вывод статистики по времени работы на экран
  std::cout << "Worked: " << (finishTime - startTime) / 1000.0 << " sec" <<
std::endl;
}
```

В результате выполнения программа выдает следующее:

```
Result: 970104589
Worked: 2578.33 sec
```

Проверяем полученный результат:

`scrambler(970104589)` → «Password is correct»

Ответ найден, он равен 970104589.

Способ 2.

Можно предположить, что функция `swap()`, выполняемая в цикле (строки 18-22), является симметричной. То есть, если выполнить этот цикл 2 раза: первый раз – над числом, второй раз – над полученным в первом шаге результатом, то в этом случае можно получить то же самое исходное число. Это легко проверить на любом случайном числе. Пример приведен в

листинге 2.1-2.

Листинг 2.1-2 – Проверка симметричности функции swap()

```
/// Проверка работы только цикла
/// из функции scrambler()
/// PARAMS
///    keyword - преобразуемое число
/// RETURN
///    преобразованное число (int)
///
int scrambler_test(int keyword)
{
  int res = keyword;
  // цикл из оригинальной функции scrambler()
  for (int i = 1; i < 16; i = i * 2)
  {
    for (int j = 0; j < 32 - i; j = j + i * 2)
      res = swap(res, j, i);
  }
  // возвращаем результат
  return res;
}

int main()
{
  int keyword1, keyword2, keyword3;
  // исходное число
  keyword1 = 1234567;
  // первое преобразование
  keyword2 = scrambler_test(keyword1);
  // вотрое преобразование
  keyword3 = scrambler_test(keyword2);
  // вывод на экран
  std::cout << "keyword1: " << keyword1 << std::endl;
  std::cout << "keyword2: " << keyword2 << std::endl;
  std::cout << "keyword3: " << keyword3 << std::endl;
}
```

Программа вернула следующий результат:

```
keyword1: 1234567
keyword2: 1208017259
keyword3: 1234567
```

Если цикл с функцией swap() симметричный (функция `scrambler_test()` в листинге 2.1-2), то на вход функции `scrambler()` необходимо подать результат выполнения функции `scrambler_test()` с числом 1268560121 (строка 23).

Проверим результат на практике.

`scrambler_test(1268560121)` → `970104589`
`scrambler(970104589)` → «Password is correct»

Ответ найден, он равен 970104589.

Ответ: 970104589.

Вариант 2

Система аутентификации шифрует пароли особым образом, показанном в виде функции `scrambler` на языке C++. Зная алгоритм шифрования, вычислите пароль. Фрагмент кода указан ниже.

C++

```
int swap(int value, int start, int len)
{
  len = len % (sizeof(value) * 8);
  start = start % (sizeof(value) * 8);

  int bits = (INT_MAX >> ((sizeof(value) * 8) - len - 1));
  bits = bits << start;

  int buf = (value >> len) & bits;
  int res = value & bits;
  res = res << len;
  res |= buf;

  int rest = value & ~bits;
  rest = rest & ~(bits << len);

  return rest | res;
}

//////////////////////////////////////////////////
void scrambler(int keyword)
{
  int res = keyword;
  for (int i = 1; i < 16; i = i * 2)
  {
    for (int j = 0; j < 32 - i; j = j + i * 2)
      res = swap(res, j, i);
  }

  if (res == -1268496634)
    std::cout << "Password is correct\n";
  else
    std::cout << "Password is wrong\n";
}
```

Решение

Разберем представленный код и перепишем его построчно.

1.	`int swap(int value, int start, int len)`
2.	`{`
3.	`  len = len % (sizeof(value) * 8);`
4.	`  start = start % (sizeof(value) * 8);`
5.	`  int bits = (INT_MAX >> ((sizeof(value) * 8) - len - 1));`
6.	`  bits = bits << start;`
7.	`  int buf = (value >> len) & bits;`
8.	`  int res = value & bits;`
9.	`  res = res << len;`
10.	`  res |= buf;`
11.	`  int rest = value & ~bits;`
12.	`  rest = rest & ~(bits << len);`

```
13.   return rest | res;
14. }
15. void scrambler(int keyword)
16. {
17.   int res = keyword;
18.   for (int i = 1; i < 16; i = i * 2)
19.   {
20.     for (int j = 0; j < 32 - i; j = j + i * 2)
21.       res = swap(res, j, i);
22.   }
23.   if (res == -1268496634)
24.     std::cout << "Password is correct\n";
25.   else
26.     std::cout << "Password is wrong\n";
27. }
```

Функция `scrambler()` проверяет число (`keyword`). Если число правильное (строка 23), то выводится на экран сообщение *«Password is correct»* (строка 24), иначе – выводится на экран сообщение «*Password is wrong*» (строка 25). Проверка правильности числа осуществляется с помощью функции swap() (строки1-14) и значение этой функции сравнивается с константой -1268496634 (строка 23).

Существует 2 способа решения задачи:

1) Перебор.
2) Аналитический.

Способ 1.

Перебору будет подвергаться параметр функции `scrambler()`: от 0 до максимального значения типа данных `int` (`MAX_INT`).

Для автоматизации перебора необходимо модифицировать саму функцию `scrambler()`:

1) Сделать так, чтобы функция возвращала результат, например, логического типа (`bool`). Для этого необходимо заменить объявление функции (строка 15) на:

```
bool scrambler(int keyword)
```

2) Заменить вывод на экран оператором возвращения результата:
   а) строку 24 заменить на `return true;`
   б) строку 26 заменить на `return false;`

Пример реализации самого цикла перебора представлен в листинге 2.2-1.

Листинг 2.2-1 – Реализация цикла перебора параметра функции scrambler() на языке программирования C++

```
int main()
{
  // начальное время (в мс)
  int startTime = GetTickCount();

  // основной цикл перебора
  for (int i = 0; i < INT_MAX; i++)
    if (scrambler(i) == true)
    {
      // вывод результата на экран
      std::cout << "Result: " << i << std::endl;
    }
```

```
  // время завершения цикла (в мс)
  int finishTime = GetTickCount();

  // вывод статистики по времени работы на экран
  std::cout << "Worked: " << (finishTime - startTime) / 1000.0 << " sec" <<
std::endl;
}
```

В результате выполнения программа выдает следующее:

```
Result: 640508130
Worked: 2647.28 sec
```

Проверяем полученный результат:

`scrambler(640508130)` → «Password is correct»

Ответ найден, он равен 640508130.

Способ 2.

Можно предположить, что функция `swap()`, выполняемая в цикле (строки 18-22), является симметричной. То есть, если выполнить этот цикл 2 раза: первый раз – над числом, второй раз – над полученным в первом шаге результатом, то в этом случае можно получить то же самое исходное число. Это легко проверить на любом случайном числе. Пример приведен в листинге 2.1-2.

Листинг 2.1-2 – Проверка симметричности функции swap()

```
/// Проверка работы только цикла
/// из функции scrambler()
/// PARAMS
///    keyword - преобразуемое число
/// RETURN
///    преобразованное число (int)
///
int scrambler_test(int keyword)
{
  int res = keyword;
  // цикл из оригинальной функции scrambler()
  for (int i = 1; i < 16; i = i * 2)
  {
    for (int j = 0; j < 32 - i; j = j + i * 2)
      res = swap(res, j, i);
  }
  // возвращаем результат
  return res;
}

int main()
{
  int keyword1, keyword2, keyword3;
  // исходное число
  keyword1 = 1234567;
  // первое преобразование
  keyword2 = scrambler_test(keyword1);
  // вотрое преобразование
  keyword3 = scrambler_test(keyword2);
  // вывод на экран
  std::cout << "keyword1: " << keyword1 << std::endl;
  std::cout << "keyword2: " << keyword2 << std::endl;
  std::cout << "keyword3: " << keyword3 << std::endl;
}
```

Программа вернула следующий результат:

```
keyword1: 1234567
keyword2: 1208017259
keyword3: 1234567
```

Если цикл с функцией swap() симметричный (функция `scrambler_test()` в листинге 2.1-2), то на вход функции `scrambler()` необходимо подать результат выполнения функции `scrambler_test()` с числом -1268496634 (строка 23).

Проверим результат на практике.

`scrambler_test(-1268496634)` → `640508130`

`scrambler(640508130)` → «`Password is correct`»

Ответ найден, он равен 640508130.

Ответ: 640508130.

Задача 3. Сеть LOR

Вариант 1

Один студент решил создать свою анонимную сеть с шифрованием и виртуальными тоннелями и назвал её LOR.

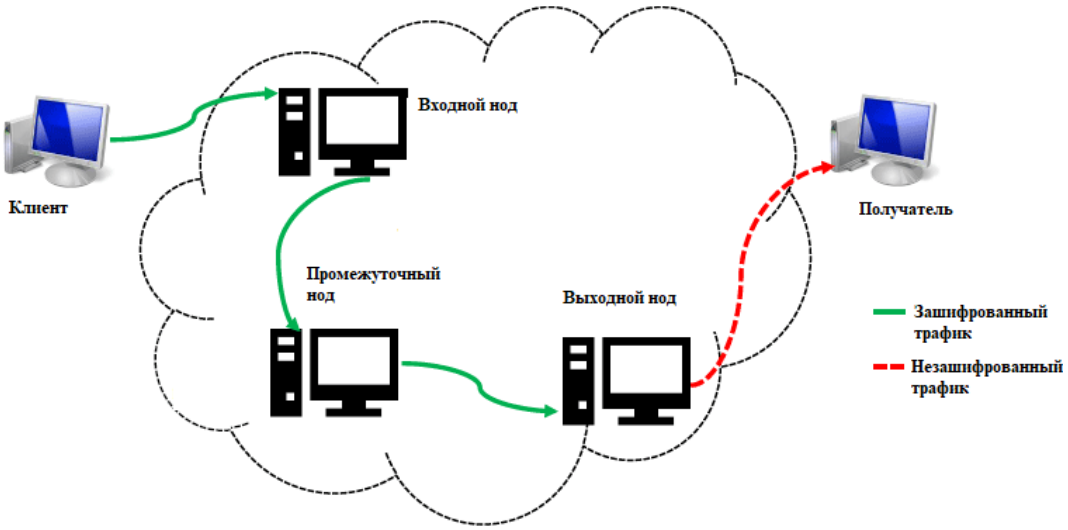

Рисунок. Схема сети LOR

Нод – узел сети LOR, способный принимать данные, расшифровывать и передавать их.

Чтобы отправить данные, клиент три раза шифрует их особым методом. Далее зашифрованная информация передается входному ноду, который расшифровывает её один раз. После этого данные отправляются на промежуточный нод, который так же расшифровывает их один раз. Далее промежуточный нод отправляет данные выходному ноду, который расшифровывает их третий раз, получая данные уже в открытом виде. После этого данные в открытом виде отправляются получателю.

Используемая функция шифрования:

$$E(x) = (ax + b) \bmod m, \quad \text{где}$$

x – номер шифруемого символа (см. таблицу),

a и **b** – ключи, при этом **a** и **m** должны быть взаимно простыми (НОД(a, m) = 1, a < m),

m – количество символов в алфавите (m = 30).

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25
.	**,**	**(пробел)**	**–**									
26	27	28	29									

При первом шифровании ключ **a** выбирается так, чтобы **a** и **m** были взаимно простыми (НОД(a, m) = 1, a < m).

При втором и третьем шифровании ключ **a** равен номеру первого зашифрованного символа сообщения, полученного после применения шифрования. Если номер первого зашифрованного символа не является взаимно простым к **m**, то в качестве ключа **a** берется ближайшее большее число, удовлетворяющее правилу. Если такого числа нет (например, номер символа равен 30), то в качестве ключа используется значение 1.

Для расшифрования используется другая функция:

$$\mathbf{D(x) = a^{-1}(E(x) - b) \bmod m},$$ где

$\mathbf{a^{-1}}$ – число, обратное **a** по модулю **m** ($a * a^{-1} = 1 \bmod m$). При этом, число $\mathbf{a^{-1}}$ так же удовлетворяет условию: НОД(a^{-1}, m) = 1, $a^{-1} < m$.

Расшифруйте отправленное клиентом сообщение, если известно, что b = 5 на всех нодах, а исходное сообщение заканчивается символом “**.**”. В ответе укажите исходное сообщение, а также ключи шифрования входного, промежуточного и выходного нода.

Перехваченное сообщение от клиента:

```
YMXNDXNDYMJDJS L
```

Решение

Для начала необходимо определиться, какие ключи могут быть использованы для шифрования сообщения. Для этого необходимо найти все числа **a**, которые будут взаимно простыми с **m=30**, то есть НОД(a, 30) = 1, a < 30. Напишем функцию, которая возвращает все подходящие числа (листинг 3.1-1). Воспользуемся алгоритмом Евклида по вычислению НОД.

Листинг 3.1-1 – Функция получения массива чисел, взаимнопростых с m, на языке программирования C++

```
/// Вычисление НОД по алгоритму Евклида
/// (Берем остаток от деления большего на меньшее, пока не будет ноль)
/// PARAMS
///    числа a,b (int)
/// RETURN
///    значение НОД
///
int NOD(int a, int b)
{
  // цикл пока все числа не нулевые
  while (a > 0 && b > 0)
  {
    if (a > b)
      a %= b;
    else
      b %= a;
  }
  return a + b;
}
```

Для получения массива ключей при m = 30 необходимо воспользоваться функцией `getKeys()` (листинг 3.1-2).

Листинг 3.1-2 – Получение массива ключей для m = 30 на языке программирования C++

```
/// Получение массива ключей
/// PARAMS
///    число m (int)
///    ключи должны быть взаимно простым с ним
/// RETURN
///    массив ключей (vector<int>)
///
vector<int> getKeys(int m)
{
  vector<int> keys;
  // цикл от 1 до m
  for (int i = 1; i < m; i++)
  {
    // если НОД(очередное число, m) == 1,
    // то добавляем очередное число в массив ключей
    if (NOD(i, m) == 1)
      keys.push_back(i);
  }
  // возвращаем результат
  return keys;
}

int main()
{
  // получение массива ключей для m = 30
  vector<int> keys = getKeys(30);

  // вывод на экран результата
  for (int i = 0; i < keys.size(); i++)
    cout << keys[i] << " ";
  cout << endl;
}
```

Результат выполнения программы:

```
1 7 11 13 17 19 23 29
```

Всего таких ключей – 8.

Дальше возможно 2 варианта:

1) Перебирать комбинации из 3-х ключей на примере шифрования символа точки ‘.’

2) Перебирать комбинации из 3-х обратных ключей ($\mathbf{a^{-1}}$), которые будут из этого же множества ключей (листинг 3.1-2) и перебирать их комбинации для расшифрования последнего символа сообщения, пока не получим символ точки ‘.’. После этого искать (подбирать или вычислить) используемые ключи шифрования.

Первый вариант проще, поскольку сразу дает комбинацию ключей шифрования. Однако для него все равно потребуется реализация функции расшифрования, чтобы получить исходное сообщение.

Для удобства и наглядности ниже приведен пример реализации второго варианта. Для него необходимо написать программу, которая перебирает все возможные комбинации ключей и расшифровывает последний символ сообщения (‘**L**’) три раза, пока не получится символ точки ‘**.**’. Учитывая, что множество ключей шифрования и расшифрования одинаковое (их всего 8), такой перебор можно реализовать достаточно быстро. Всего возможно 8*8*8 = 512 комбинаций. Интересуют только те комбинации ключей, в результате использования которых при расшифровании получится символ точки ‘**.**’. Дополнительно необходимо реализовать функцию расшифрования одного символа и всего текстового сообщения.

Пример реализации такой программы на языке программирования C++ приведен в

листинге 3.1-3.

Листинг 3.1-3 – Пример реализации программы перебора ключей для расшифрования символа ‘L’ до получения символа точки ‘.’ на языке программирования C++

```
// размер алфавита
const int m = 30;
// АЛФАВИТ
char ALPHA[m] = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
                  'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
                  'U', 'V', 'W', 'X', 'Y', 'Z', '.', ',', ' ', '-' };

/// Получение индекса символа алфавита ALPHA
/// PARAMS
///    c - символ (char)
/// RETURN
///    индекс символа (int)
///    ИЛИ
///    -1 - если символ не найден
///
int getIndex(char c)
{
  for (int i = 0; i < m; i++)
    if (ALPHA[i] == c)
      return i;
  // если ошибка
  return -1;
}

/// Расшифрование символа с использованием ключей a,b,m
/// PARAMS
///    index - индекс зашифрованного символа (int)
///    a - ключ расшифрования 1-я часть (int)
///    b - ключ расшифрования 2-я часть (int)
///    m - модуль для расшифрования (int)
/// RETURN
///    индекс расшифрованного символа (int)
///
int decipher(int index, int a, int b, int m)
{
  int res = (index * a - b) % m;
  return res;
}

/// Расшифрование сообщения (string) с использованием ключей a,b,m
/// PARAMS
///    msg - зашифрованное сообщение (string)
///    a - ключ расшифрования 1-я часть (int)
///    b - ключ расшифрования 2-я часть (int)
///    m - модуль для шифрования (int)
/// RETURN
///    зашифрованное сообщение (string)
///
string decipherText(string msg, int a, int b, int m)
{
  string res = "";
  int cindex, index;
  // цикл по каждому символу зашифрованной строки
  for (int i = 0; i < msg.length(); i++)
  {
    // получение индекса очередного символа
```

```
      cindex = getIndex(msg[i]);
      // если индекс символа найден - расшифровываем его
      if (cindex != -1)
      {
        // расшифрование символа
        index = decipher(cindex, a, b, m);
        // добавление расшифрованного символа к строке результата
        res = res + ALPHA[index];
      }
    }
    return res;
}

int main()
{
    const int b = 5; // ключ 2-я часть
    // получение массива ключей для m = 30
    vector<int> keys = getKeys(30);

    char csymbol = 'L';                  // зашифрованный символ 'L'
    int csymbolIndex = getIndex(csymbol);// индекс зашифрованного символа(11)
    char symbol = '.';                   // исходный символ '.'
    int symbolIndex = getIndex(symbol); // индекс исходного символа (26)

    for (int i1 = 0; i1 < keys.size(); i1++)
      for (int i2 = 0; i2 < keys.size(); i2++)
        for (int i3 = 0; i3 < keys.size(); i3++)
        {
          csymbolIndex = getIndex(csymbol);
          // расшифрование на 3-м ключе (keys[i3])
          csymbolIndex = decipher(csymbolIndex, keys[i3], b, m);
          // расшифрование на 2-м ключе (keys[i2])
          csymbolIndex = decipher(csymbolIndex, keys[i2], b, m);
          // расшифрование на 1-м ключе (keys[i1])
          csymbolIndex = decipher(csymbolIndex, keys[i1], b, m);

          // проверка результата
          if (csymbolIndex == symbolIndex)
          {
            // вывод на экран ключей расшифрования
            cout << "keys: " << keys[i3] << "," << keys[i2] << "," <<
              keys[i1] << endl;
          }
        }
```

Результатом выполнения данной программы будет следующее:

```
keys: 13,1,7
keys: 19,7,7
keys: 13,11,7
keys: 1,13,7
keys: 19,17,7
keys: 7,19,7
keys: 1,23,7
keys: 7,29,7
keys: 11,11,11
keys: 23,17,11
keys: 17,23,11
keys: 29,29,11
keys: 7,1,13
```

```
keys: 1,7,13
keys: 7,11,13
keys: 19,13,13
keys: 1,17,13
keys: 13,19,13
keys: 19,23,13
keys: 13,29,13
keys: 23,11,17
keys: 29,17,17
keys: 11,23,17
keys: 17,29,17
keys: 19,1,19
keys: 7,7,19
keys: 19,11,19
keys: 13,13,19
keys: 7,17,19
keys: 1,19,19
keys: 13,23,19
keys: 1,29,19
keys: 17,11,23
keys: 11,17,23
keys: 29,23,23
keys: 23,29,23
keys: 29,11,29
keys: 17,17,29
keys: 23,23,29
keys: 11,29,29
```

Для поиска подходящей комбинации из найденных 40-ка вариантов необходимо расшифровать всё сообщение и посмотреть результат. Для этого необходимо дополнить программу исходным кодом, пример которого приведен на листинге 3.1-4.

Листинг 3.1-4 – Пример реализации программы перебора ключей для расшифрования всего сообщения на языке программирования C++

```
  char csymbol = 'L';                  // зашифрованный символ 'L'
  int csymbolIndex = getIndex(csymbol);// индекс зашифрованного символа(11)
  char symbol = '.';                   // исходный символ '.'
  int symbolIndex = getIndex(symbol); // индекс исходного символа (26)
  string ctext = "YMXNDXNDYMJDJS L"; // зашифрованное сообщение
  string text = "";                    // исходное сообщение
  for (int i1 = 0; i1 < keys.size(); i1++)
    for (int i2 = 0; i2 < keys.size(); i2++)
      for (int i3 = 0; i3 < keys.size(); i3++)
      {
        csymbolIndex = getIndex(csymbol);
        // расшифрование на 3-м ключе (keys[i3])
        csymbolIndex = decipher(csymbolIndex, keys[i3], b, m);
        text = decipherText(ctext, keys[i3], b, m);
        // расшифрование на 2-м ключе (keys[i2])
        csymbolIndex = decipher(csymbolIndex, keys[i2], b, m);
        text = decipherText(text, keys[i2], b, m);
        // расшифрование на 1-м ключе (keys[i1])
        csymbolIndex = decipher(csymbolIndex, keys[i1], b, m);
        text = decipherText(text, keys[i1], b, m);

        // проверка результата
        if (csymbolIndex == symbolIndex)
        {
          // вывод на экран ключей расшифрования и самого сообщения
```

```
            cout << "keys: " << keys[i3] << "," << keys[i2] << "," <<
              keys[i1] << " - ";
            cout << text << endl;
        }
    }
```

Результатом выполнения данной программы будет следующее:

```
keys: 13,1,7 - JI I JYYDN.
keys: 19,7,7 - J,I SI SJ,YSYDN.
keys: 13,11,7 - THIS IS THE END.
keys: 1,13,7 - J,I I J,YYDN.
keys: 19,17,7 - THIS IS THE END.
keys: 7,19,7 - J,I SI SJ,YSYDN.
keys: 1,23,7 - THISISTHEEND.
keys: 7,29,7 - THIS IS THE END.
keys: 11,11,11 - THIS IS THE END.
keys: 23,17,11 - THIS IS THE END.
keys: 17,23,11 - THIS IS THE END.
keys: 29,29,11 - THIS IS THE END.
keys: 7,1,13 - J,I SI SJ,YSYN.
keys: 1,7,13 - J,I I J,YYDN.
keys: 7,11,13 - THIS IS THE END.
keys: 19,13,13 - J,I SI SJ,YSYDN.
keys: 1,17,13 - THISISTHEEND.
keys: 13,19,13 - J,I SI SJ,YSYDN.
keys: 19,23,13 - THIS IS THE END.
keys: 13,29,13 - THIS IS THE END.
keys: 23,11,17 - THIS IS THE END.
keys: 29,17,17 - THIS IS THE END.
keys: 11,23,17 - THIS IS THE END.
keys: 17,29,17 - THIS IS THE END.
keys: 19,1,19 - ,ISIS,YSYDN.
keys: 7,7,19 - J,I SI SJ,YSYDN.
keys: 19,11,19 - THIS IS THE END.
keys: 13,13,19 - J,I SI SJ,YSYDN.
keys: 7,17,19 - THIS IS THE END.
keys: 1,19,19 - J,I I J,YYDN.
keys: 13,23,19 - THIS IS THE END.
keys: 1,29,19 - THISISTHEEND.
keys: 17,11,23 - THIS IS THE END.
keys: 11,17,23 - THIS IS THE END.
keys: 29,23,23 - THIS IS THE END.
keys: 23,29,23 - THIS IS THE END.
keys: 29,11,29 - THIS IS THE END.
keys: 17,17,29 - THIS IS THE END.
keys: 23,23,29 - THIS IS THE END.
keys: 11,29,29 - THIS IS THE END.
```

В результате получается 25 комбинаций, дающий фразу «THIS IS THE END.». Эта фраза и есть исходное сообщение. Осталось подобрать исходные ключи, которые использовались для шифрования с учетом правила их выбора. Это возможно решить двумя способами:

1) Подбирать ключи шифрования для фразы «THIS IS THE END.».

2) Вычислить ключи шифрования, зная ключи расшифрования.

Первый способ проще, поскольку известна начальная фраза и известна зашифрованная фраза. Осталось подобрать комбинацию из трех ключей шифрования так, чтобы каждый ключ удовлетворял условию:

*«ключ **a** равен номеру первого зашифрованного символа сообщения, полученного после*

применения шифрования. Если номер первого зашифрованного символа не является взаимно простым к ***m****, то в качестве ключа* ***a*** *берется ближайшее большее число, удовлетворяющее правилу. Если такого числа нет (например, номер символа равен 30), то в качестве ключа используется значение 1».*

Для этого можно реализовать цикл, аналогичный циклу из листинга 3.1-4, но с использованием функции шифрования и дополнительной проверкой ключей на соответствие условию. Пример такой программы на языке программирования C++ приведен на листинге 3.1-5.

Листинг 3.1-5 – Пример реализации программы перебора ключей для шифрования фразы «THIS IS THE END.» на языке программирования C++

```
string text_orig = "THIS IS THE END.";       // исходное сообщение
string ctext_origin = "YMXNDXNDYMJDJS L";    // полученное шифрованное
                                             // сообщение
string ctext1, ctext2, ctext3;
// перебор 1-го ключа
for (int i1 = 0; i1 < keys.size(); i1++)
{
  // шифрование на 1-м ключе (keys[i1])
  ctext1 = cipherText(text_orig, keys[i1], b, m);
  // поиск 2-го ключа
  for (int i2 = 0; i2 < keys.size(); i2++)
  {
  // пропускаем ключ, если он меньше номера первого зашифрованного символа
    if (keys[i2] < getIndex(ctext1[0]))
      continue;
    // шифрование на 2-м ключе (keys[i2])
    ctext2 = cipherText(ctext1, keys[i2], b, m);
    // перебор 3-го ключа
    for (int i3 = 0; i3 < keys.size(); i3++)
    {
  // пропускаем ключ, если он меньше номера первого зашифрованного символа
      if (keys[i3] < getIndex(ctext2[0]))
        continue;
      // шифрование на 3-м ключе (keys[i3])
      ctext3 = cipherText(ctext2, keys[i3], b, m);
      // сравнение с результатом
      if (ctext3 == ctext_origin)
      {
        // вывод на экран
        cout << "key1: " << keys[i1] << ", 1st letter - " << ctext1[0] << "
(" << getIndex(ctext1[0]) << ")" << endl;
        cout << "key2: " << keys[i2] << ", 1st letter - " << ctext2[0] << "
(" << getIndex(ctext2[0]) << "), " << endl;
        cout << "key3: " << keys[i3] << ", 1st letter - " << ctext3[0] << "
(" << getIndex(ctext3[0]) << "), " << endl;
        cout << "Result: " << ctext3 << endl;
          cout << "--------" << endl;
      }
    } // for (i3)
  } // for (i2)
} // for(i1)
```

Результат выполнения программы:

```
key1: 7, 1st letter - S (18)
key2: 19, 1st letter - R (17),
key3: 17, 1st letter - Y (24),
Result: YMXNDXNDYMJDJS L
```

```
--------
key1: 13, 1st letter - M (12)
key2: 13, 1st letter - L (11),
key3: 29, 1st letter - Y (24),
Result: YMXNDXNDYMJDJS L
--------
key1: 13, 1st letter - M (12)
key2: 19, 1st letter - X (23),
key3: 23, 1st letter - Y (24),
Result: YMXNDXNDYMJDJS L
--------
key1: 19, 1st letter - G (6)
key2: 7, 1st letter - R (17),
key3: 17, 1st letter - Y (24),
Result: YMXNDXNDYMJDJS L
--------
key1: 19, 1st letter - G (6)
key2: 13, 1st letter - X (23),
key3: 23, 1st letter - Y (24),
Result: YMXNDXNDYMJDJS L
--------
```

Вариант (7,19,17) и (19,7,17) подходят. Остальные – не подходят, поскольку в тройке (13,13,29) третьим ключом должен был быть ключ 13 (первое простое число после 12). Аналогично в тройке (19,13,23) – вторым ключом должен был быть ключ 7 (первое простое число после 6).

Ответ: THIS IS THE END. Ключи: a1 = 7, a2 = 19, a3 = 17 ИЛИ
a1 = 19, a2 = 7, a3 = 17.

Вариант 2

Один студент решил создать свою анонимную сеть с шифрованием и виртуальными тоннелями и назвал её LOR.

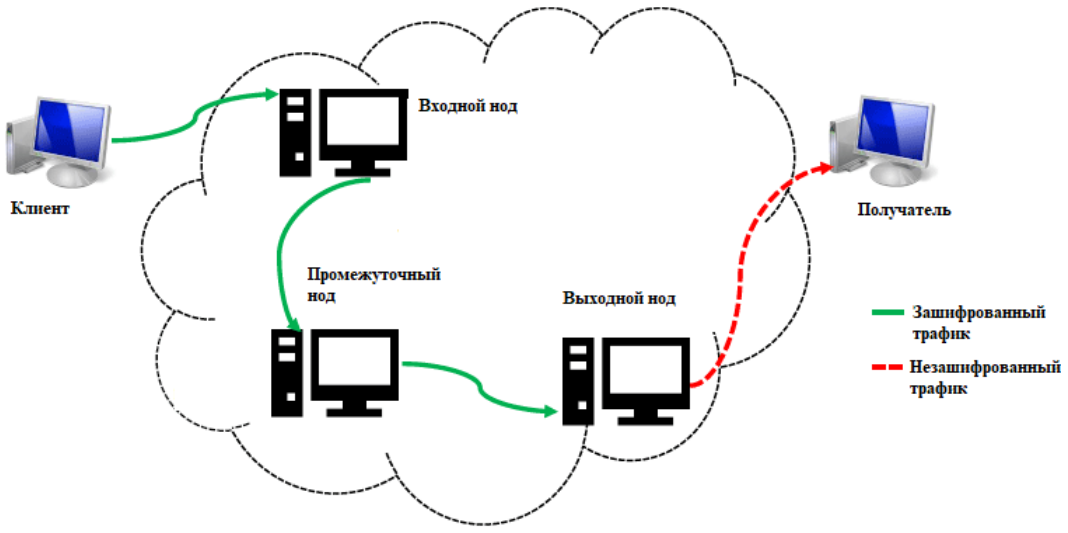

Рисунок. Схема сети LOR

Нод – узел сети LOR, способный принимать данные, расшифровывать и передавать их.

Чтобы отправить данные, клиент три раза шифрует их особым методом. Далее зашифрованная информация передается входному ноду, который расшифровывает её один раз. После этого данные отправляются на промежуточный нод, который так же расшифровывает их

один раз. Далее промежуточный нод отправляет данные выходному ноду, который расшифровывает их третий раз, получая данные уже в открытом виде. После этого данные в открытом виде отправляются получателю.

Используемая функция шифрования:

$$\mathbf{E(x) = (ax + b) \bmod m},$$ где

x – номер шифруемого символа (см. таблицу),

a и **b** – ключи, при этом **a** и **m** должны быть взаимно простыми (НОД(a, m) = 1, a < m),

m – количество символов в алфавите (m = 30).

A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**
0	1	2	3	4	5	6	7	8	9	10	11	12
N	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
13	14	15	16	17	18	19	20	21	22	23	24	25
.	**,**	**(пробел)**	**–**									
26	27	28	29									

При первом шифровании ключ **a** выбирается так, чтобы **a** и **m** были взаимно простыми (НОД(a, m) = 1, a < m).

При втором и третьем шифровании ключ **a** равен номеру первого зашифрованного символа сообщения, полученного после применения шифрования. Если номер первого зашифрованного символа не является взаимно простым к **m**, то в качестве ключа **a** берется ближайшее большее число, удовлетворяющее правилу. Если такого числа нет (например, номер символа равен 30), то в качестве ключа используется значение 1.

Для расшифрования используется другая функция:

$$\mathbf{D(x) = a^{-1}(E(x) - b) \bmod m},$$ где

$\mathbf{a^{-1}}$ – число, обратное **a** по модулю **m** ($a * a^{-1} = 1 \bmod m$). При этом, число $\mathbf{a^{-1}}$ так же удовлетворяет условию: НОД(a^{-1}, m) = 1, a^{-1} < m.

Расшифруйте отправленное клиентом сообщение, если известно, что b = 5 на всех нодах, а исходное сообщение заканчивается символом “**.**”. В ответе укажите исходное сообщение, а также ключи шифрования входного, промежуточного и выходного нода.

Перехваченное сообщение от клиента:

```
ZSS-V,BR-C,-ROED
```

Решение

Для начала необходимо определиться, какие ключи могут быть использованы для шифрования сообщения. Для этого необходимо найти все числа **a**, которые будут взаимно простыми с **m=30**, то есть НОД(a, 30) = 1, a < 30. Напишем функцию, которая возвращает все подходящие числа (листинг 3.1-1). Воспользуемся алгоритмом Евклида по вычислению НОД.

Листинг 3.2-1 – Функция получения массива чисел, взаимнопростых с m, на языке программирования Python

```
##
# поиск НОД для 2-х чисел
#    a - первое число
#    b - второе число
# RETURN
#    вычисленный НОД
##
def NOD(a: int, b: int):
  while a != 0 and b != 0:
```

```
    if a > b:
      a = a % b
    else:
      b = b % a
  return a + b
```

Для получения массива ключей при m = 30 необходимо воспользоваться функцией `getKeys()` (листинг 3.2-2).

Листинг 3.2-2 – Получение массива ключей для m = 30 на языке программирования Python.

```
## Получение массива ключей
# PARAMS
#          число m (int)
#          ключи должны быть взаимно простым с ним
# RETURN
#          массив ключей (vector<int>)
##
def getKeys(m: int):
  keys = []
  # цикл от 1 до m
  for i in range(1,m):
    # если НОД(очередное число, m) == 1,
    # то добавляем очередное число в массив ключей
    if NOD(i, m) == 1:
      keys.append(i)
  # возвращаем результат
  return keys;


m = 30
keys = getKeys(m)
print(keys)
```

Результат выполнения программы:

```
[1, 7, 11, 13, 17, 19, 23, 29]
```

Всего таких ключей – 8.

Дальше возможно 2 варианта:

1) Перебирать комбинации из 3-х ключей на примере шифрования символа точки ‘.’

2) Перебирать комбинации из 3-х обратных ключей ($\mathbf{a^{-1}}$), которые будут из этого же множества ключей (листинг 3.1-2) и перебирать их комбинации для расшифрования последнего символа сообщения, пока не получим символ точки ‘.’. После этого искать (подбирать или вычислить) используемые ключи шифрования.

Воспользуемся вторым вариантом, поскольку для расшифрования используются ключи из того же множества, что и ключи шифрования. В программе необходимо перебрать все возможные тройки ключей, использование которых при расшифровании перехваченного сообщения дает текст, заканчивающийся на символ точки “**.**”. Всего возможно 8*8*8 = 512 таких комбинаций, перебор которых выполнится достаточно быстро. Пример реализации такой программы на языке программирования Python приведен в листинге 3.2-3.

Листинг 3.2-3 – Пример реализации программы перебора ключей для расшифрования сообщения до получения символа точки ‘.’ в конце на языке программирования Python

```
# размер алфавита
m = 30
# АЛФАВИТ
ALPHA = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
```

```
          'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
          'U', 'V', 'W', 'X', 'Y', 'Z', '.', ',', ' ', '-']

## Получение индекса символа алфавита ALPHA
# PARAMS
#   c - символ (char)
# RETURN
#   индекс символа (int)
#   ИЛИ
#   -1 - если символ не найден
##
def getIndex(c: str):
  for i in range(len(ALPHA)):
    if ALPHA[i] == c[0] :
      return i
  # если ошибка
  return -1

## Расшифрование символа с использованием ключей a,b,m
# PARAMS
#   index - индекс зашифрованного символа (int)
#   a - ключ расшифрования 1-я часть (int)
#   b - ключ расшифрования 2-я часть (int)
#   m - модуль для расшифрования (int)
# RETURN
#   индекс расшифрованного символа (int)
##
def decipher(index: int, a: int, b: int, m: int):
  res = (a * (index - b)) % m
  return res

## Расшифрование сообщения (string) с использованием ключей a,b,m
# PARAMS
#   msg - зашифрованное сообщение (string)
#   a - ключ расшифрования 1-я часть (int)
#   b - ключ расшифрования 2-я часть (int)
#   m - модуль для шифрования (int)
# RETURN
#   зашифрованное сообщение (string)
##
def decipherText(msg: str, a: int, b: int, m: int):
  res = ""
  cindex = 0
  index = 0
  # цикл по каждому символу зашифрованной строки
  for i in range(len(msg)):
    # получение индекса очередного символа
    cindex = getIndex(msg[i])
    # если индекс символа найден - расшифровываем его
    if cindex != -1:
      # расшифрование символа
      index = decipher(cindex, a, b, m)
      # добавление расшифрованного символа к строке результата
      res = res + ALPHA[index]
  return res


## main()
b = 5  # ключ 2-я часть
```

```
# получение массива ключей для m = 30
keys = getKeys(m)
print(keys)

ctext = 'ZSS-V,BR-C,-ROED' # зашифрованное сообщение
symbol = '.'        # искомый символ после расшифрования ('.')
symbolIndex = getIndex(symbol)   # индекс исходного символа (26)
ctext1 = ''
ctext2 = ''

# перебор 1-го ключа
for key1 in keys:
  # перебор 2-го ключа
  for key2 in keys:
    # перебор 3-го ключа
    for key3 in keys:
      # расшифрование на 3-м ключе (key3)
      ctext2 = decipherText(ctext, key3, b, m)
      # расшифрование на 2-м ключе (key2)
      ctext1 = decipherText(ctext2, key2, b, m)
      # расшифрование на 1-м ключе (key1)
      text = decipherText(ctext1, key1, b, m)
      # проверка последнего символа
      if text[-1] == symbol:
        # вывод на экран ключей и результат расшифрования
        print("keys: (" + str(key1) + "," + str(key2) + "," + str(key3) +
              "): " + text)
```

Результатом выполнения данной программы будет следующее:

```
keys: (11,1,7): KLLSCOWYSJOSYDN.
keys: (11,1,17): ALL COME TO END.
keys: (11,7,1): KLLSCOWYSJOSYDN.
keys: (11,7,11): ALL COME TO END.
keys: (11,13,19): KLLSCOWYSJOSYDN.
keys: (11,13,29): ALL COME TO END.
keys: (11,19,13): KLLSCOWYSJOSYDN.
keys: (11,19,23): ALL COME TO END.
keys: (17,1,1): KLLSCOWYSJOSYDN.
keys: (17,1,11): ALL COME TO END.
keys: (17,7,13): KLLSCOWYSJOSYDN.
keys: (17,7,23): ALL COME TO END.
keys: (17,13,7): KLLSCOWYSJOSYDN.
keys: (17,13,17): ALL COME TO END.
keys: (17,19,19): KLLSCOWYSJOSYDN.
keys: (17,19,29): ALL COME TO END.
keys: (23,1,19): KLLSCOWYSJOSYDN.
keys: (23,1,29): ALL COME TO END.
keys: (23,7,7): KLLSCOWYSJOSYDN.
keys: (23,7,17): ALL COME TO END.
keys: (23,13,13): KLLSCOWYSJOSYDN.
keys: (23,13,23): ALL COME TO END.
keys: (23,19,1): KLLSCOWYSJOSYDN.
keys: (23,19,11): ALL COME TO END.
keys: (29,1,13): KLLSCOWYSJOSYDN.
keys: (29,1,23): ALL COME TO END.
keys: (29,7,19): KLLSCOWYSJOSYDN.
keys: (29,7,29): ALL COME TO END.
keys: (29,13,1): KLLSCOWYSJOSYDN.
keys: (29,13,11): ALL COME TO END.
```

```
keys: (29,19,7): KLLSCOWYSJOSYDN.
keys: (29,19,17): ALL COME TO END.
```

В результате получается 32 комбинации, из которых 16 дает фразу «ALL COME TO END.». Эта фраза и есть исходное сообщение. Осталось подобрать исходные ключи, которые использовались для шифрования с учетом правила их выбора. Это возможно решить двумя способами:

1) Подбирать ключи шифрования для фразы «ALL COME TO END.».

2) Вычислить ключи шифрования, зная ключи расшифрования.

Воспользуемся вторым способом. Для вычисления ключа шифрования (**a**) из ключа расшифрования ($\mathbf{a^{-1}}$) необходимо найти число **a**, что

$$\mathbf{a * a^{-1} = 1 \bmod m, НОД(a, m) = 1.}$$

Очевидно, что множество ключей шифрования и расшифрования совпадает. Найти ключи шифрования, зная ключи расшифрования можно с использованием программы. Пример программы поиска обратных чисел приведен в листинге 3.2-4.

Листинг 3.2-4 – Пример реализации программы поиска обратных ключей на языке программирования Python.

```
## Получение обратных ключей
# PARAMS
#    key - ключ, для которого ищется обратный (int)
#    keys - массив ключей ([])
#    m - модуль для шифрования (int)
# RETURN
#    массив обратных ключей (того же размера, что и keys)
##
def getBackKey(key: int, keys: [], m: int):
  # для ключа key перебор ключей (i) и проверка на обратность
  for i in keys:
      if (key * i) % m == 1:
        return i

## main()
b = 5  # ключ 2-я часть
# получение массива ключей для m = 30
keys = getKeys(m)
# вывод на экран ключей
for i in keys:
  print(str(i) + " - " + str(getBackKey(i,keys,m)))
```

В результате работы программа выдает следующее:

```
1 - 1
7 - 13
11 - 11
13 - 7
17 - 23
19 - 19
23 - 17
29 - 29
```

В первом столбце содержатся ключи шифрования/расшифрования, во втором – ключи расшифрования/шифрования.

Осталось выполнить следующее:

1) Для каждой комбинации ключей расшифрования, дающих фразу «ALL COME TO END.», найти ключи шифрования.

2) Для найденных ключей шифрования проверить условие:

*«ключ **a** равен номеру первого зашифрованного символа сообщения, полученного после*

применения шифрования. Если номер первого зашифрованного символа не является взаимно простым к ***m****, то в качестве ключа* ***a*** *берется ближайшее большее число, удовлетворяющее правилу. Если такого числа нет (например, номер символа равен 30), то в качестве ключа используется значение 1».*

Для этого можно усовершенствовать программу из листинга 3.2-3, добавив необходимый код в цикл поиска ключей расшифрования. Пример модифицированного цикла представлен в листинге 3.2-5.

Листинг 3.2-5 – Пример реализации программы перебора ключей для расшифрования с проверкой условия на обратные ключи шифрования на языке программирования Python

```
# перебор 1-го ключа
for key1 in keys:
  # перебор 2-го ключа
  for key2 in keys:
    # перебор 3-го ключа
    for key3 in keys:
      # расшифрование на 3-м ключе (key3)
      ctext2 = decipherText(ctext, key3, b, m)
      # расшифрование на 2-м ключе (key2)
      ctext1 = decipherText(ctext2, key2, b, m)
      # расшифрование на 1-м ключе (key1)
      text = decipherText(ctext1, key1, b, m)
      # проверка последнего символа
      if text[-1] == symbol:
        #получение обратного ключа
        bkey3 = getBackKey(key3,keys,m)
        bkey2 = getBackKey(key2,keys,m)
        bkey1 = getBackKey(key1,keys,m)
        # проверка условия на ключи
        if bkey2 >= getIndex(ctext1[0]) \  # 2-й ключ больше зашифрованного
                                           # первым ключом 1-го символа
        and bkey3 >= getIndex(ctext2[0]):  # 3-й ключ больше зашифрованного
                                           # вторым ключом 1-го символа
          # вывод на экран ключей и результат расшифрования
          print("CipherKey1: " + str(bkey1) + ", ctext1[0]: " +
str(getIndex(ctext1[0])))
          print("CipherKey2: " + str(bkey2) + ", ctext2[0]: " +
str(getIndex(ctext2[0])))
          print("CipherKey3: " + str(bkey3) + ", ctext3[0]: " +
str(getIndex(ctext[0])))
          print("--------")
```

Результат выполнения программы:

```
ALL COME TO END.
CipherKey1: 11, ctext1[0]: 5
CipherKey2: 7, ctext2[0]: 10
CipherKey3: 29, ctext3[0]: 25
--------
…
ALL COME TO END.
CipherKey1: 23, ctext1[0]: 5
CipherKey2: 7, ctext2[0]: 10
CipherKey3: 23, ctext3[0]: 25
--------
…
ALL COME TO END.
CipherKey1: 23, ctext1[0]: 5
```

```
CipherKey2: 19, ctext2[0]: 10
CipherKey3: 29, ctext3[0]: 25
--------
ALL COME TO END.
CipherKey1: 29, ctext1[0]: 5
CipherKey2: 13, ctext2[0]: 10
CipherKey3: 29, ctext3[0]: 25
--------
ALL COME TO END.
CipherKey1: 29, ctext1[0]: 5
CipherKey2: 7, ctext2[0]: 10
CipherKey3: 11, ctext3[0]: 25
--------
ALL COME TO END.
CipherKey1: 29, ctext1[0]: 5
CipherKey2: 19, ctext2[0]: 10
CipherKey3: 23, ctext3[0]: 25
--------
```

Из полученных вариантов подходит только (29,7,1), поскольку второй ключ 7 – первое простое число после 5 (первый зашифрованный символ), а 11 – первое простое число после 10. Остальные варианты не подходят: в тройке (11,7,29): 29 – не первое простое число после 10. аналогично в тройке (29,19,23): 19 – не первое простое число после 5.

Ответ: ALL COME TO END. Ключи: a1 = 29, a2 = 7, a3 = 11.

Задача 4. DLL Hijacking

Вариант 1

В разделе импорта заголовка исполняемого файла содержится информация о подключаемых библиотеках (DLL) и импортируемых из них функциях. Для подмены одной из DLL необходимо, чтобы имя библиотеки и набор функций совпадали с именем библиотеки и набором функций, описанными в разделе импорта исполняемого файла. Для упрощения разработки подменяемой библиотеки DLL, из всех библиотек выбирают ту, из которой импортируется наименьшее количество функций.

Из предоставленного образа раздела импорта определите имя библиотеки, из которой импортируется наименьшее количество функций.

В ответе укажите имя библиотеки DLL, имена импортируемых из нее функций и количество аргументов каждой из таких функций.

Структура раздела импорта показана на рисунке.

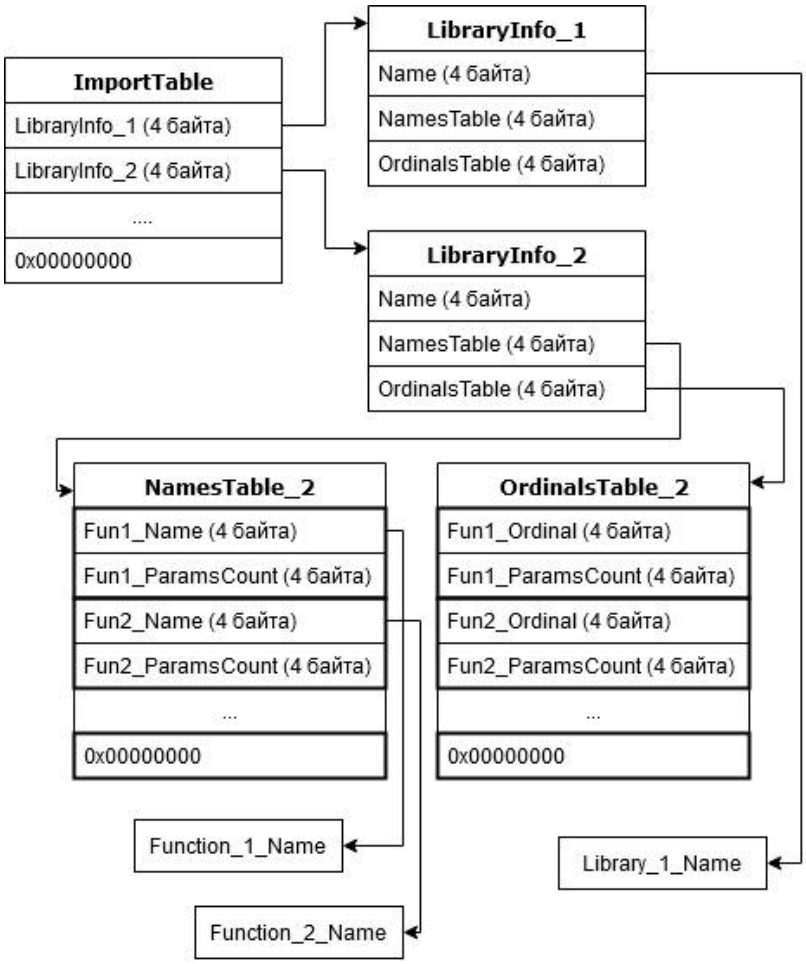

Рисунок. Структура раздела импорта

ImportTable – таблица импорта. Хранится в начале раздела импорта. Содержит адреса (4-х байтовые) на структуры *LibraryInfo*. В конце таблицы записывается 4 нулевых байта (0x00000000).

LibraryInfo – структура, содержащая информацию о библиотеке. Содержит поля:

Name – адрес строки (4 байта), содержащей имя библиотеки. Строка заканчивается нулевым байтом ('\0');

NamesTable – адрес таблицы имен импортируемых функций (4 байта);

OrdinalsTable – адрес таблицы идентификаторов импортируемых функций (4 байта).

NamesTable – таблица имен импортируемых функций. Каждая запись содержит следующие параметры:

Fun_Name – адрес строки (4 байта), содержащей имя функции. Строка заканчивается нулевым байтом ('\0');

Fun_ParamsCount – количество аргументов функции (4 байта).

В конце таблицы записывается 4 нулевых байта (0x00000000).

OrdinalsTable – таблица идентификаторов импортируемых функций. Каждая запись содержит следующие параметры:

Fun_Ordinal – идентификатор функции (4 байта);

Fun_ParamsCount – количество аргументов функции (4 байта).

В конце таблицы записывается 4 нулевых байта (0x00000000).

Все адреса и числовые значения хранятся в формате *Little-Endian*. Адреса указываются относительно начала раздела импорта, адрес которого считается равным 0x00000000.

К задаче прилагается:

файл образа раздела импорта dump_v1.bin.

Решение

Для решения задачи необходимо отыскать все импортируемые библиотеки. Далее для каждой из найденных определить количество импортируемых функций. После этого уже возможно определить, какая именно библиотека является наиболее подходящей, и для нее необходимо получить названия всех функций.

Элементами структур являются смещения относительно начала дампа, соответственно для получения необходимых данных нужно корректно определить адреса всех структур **LibraryInfo**, **NamesTable**, **OrdinalsTable**, а также адреса строк. Можно предложить 2 способа решения:

– ручной поиск (наиболее удобно использовать HEX-редактор);

– автоматизированный с использованием программы.

Способ 1.

Откроем предоставленный дамп в HEX-редакторе. Зная формат структуры **ImportTable** несложно выделить (см. рисунок 4.1-1) смещения трех структур **LibraryInfo**: *0x000003E8*, 0x000004a9 и *0x00000539* (байты расположены в обратном порядке согласно *little-endian*).

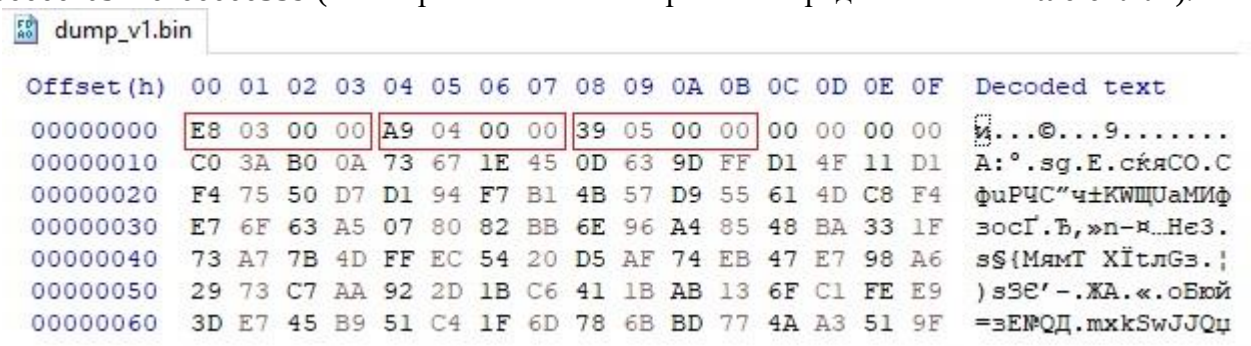

Рисунок 4.1-1 – Структура ImportTable

Используя инструмент перехода по смещению (*Search – Go to* или горячее сочетание *Alt+G*) перейдем по адресу первой структуры **LibraryInfo.**

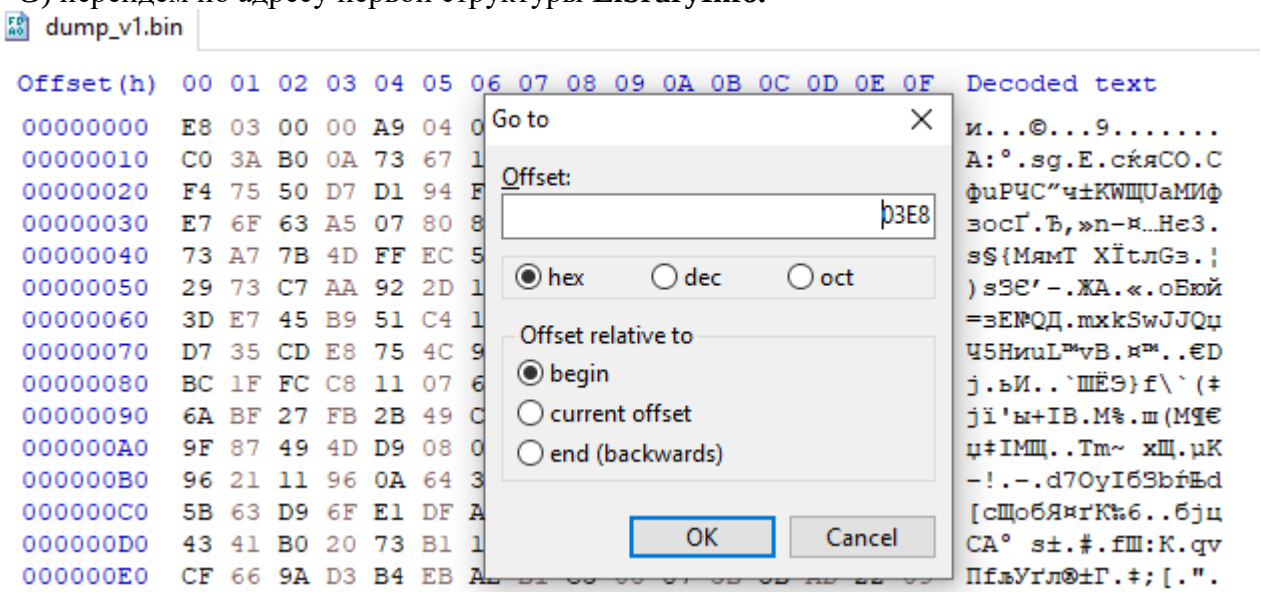

Рисунок 4.1-2 – Переход по смещению

Зная формат структуры **LibraryInfo** можно выделить адрес названия библиотеки, адреса таблицы имен и ординалов импортируемых функций.

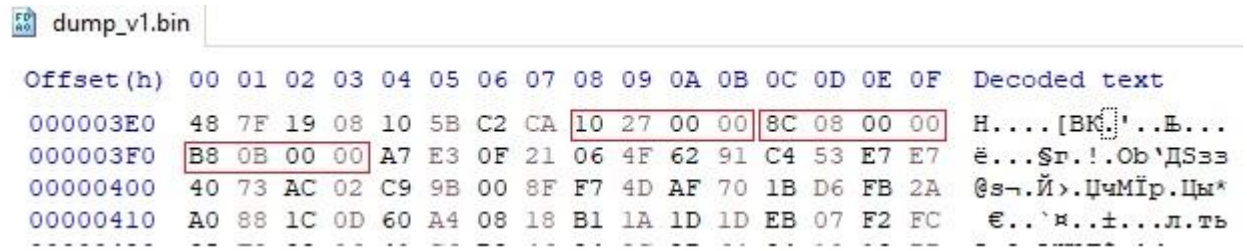

Рисунок 4.1-3 – Структура LibraryInfo первой библиотеки

На текущем этапе решения нет необходимости получать названия библиотеки, пока достаточно определить только количество импортируемых функций. Для этого нужно перейти либо к таблице имен, либо к таблице ординалов. Перейдем по адресу *0x0000088C* и, зная формат **NamesTable**, посчитаем количество импортируемых из первой библиотеки функций.

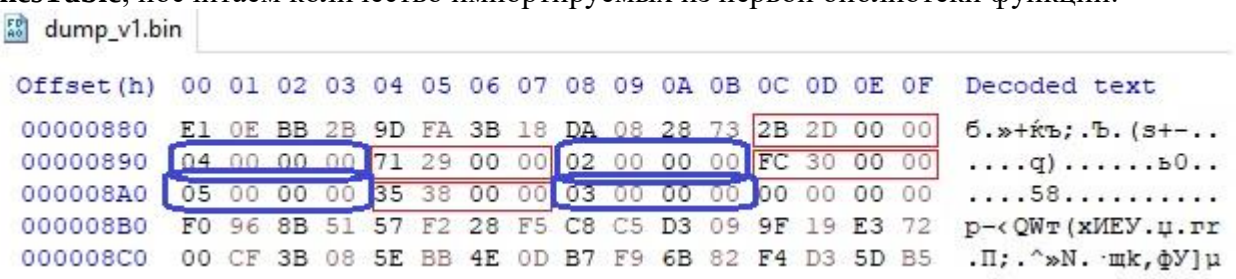

Рисунок 4.1-4 – Таблица имен функций первой библиотеки

На рисунке 4.1-4 отмечены смещения имен импортируемых функций, количество которых равно **четырем**. Синим отмечено число параметров для каждой функции: 4, 2, 5 и 3.

Аналогичным образом отыщем количество импортируемых функций для второй и третьей библиотек.

На рисунке 4.1-5 приведена структура **LibraryInfo** (смещение *0x000004A9*), а на рисунке 4.1-6 – таблица имен второй импортируемой библиотеки (смещение *0x000007D0*), из которой импортируется **три** функций. Синим на рисунке отмечены количества параметров функций: 6, 2, 1.

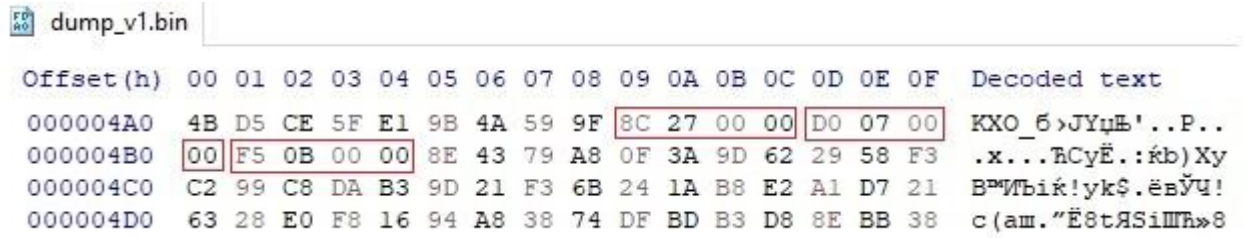

Рисунок 4.1-5 – Структура LibraryInfo второй библиотеки

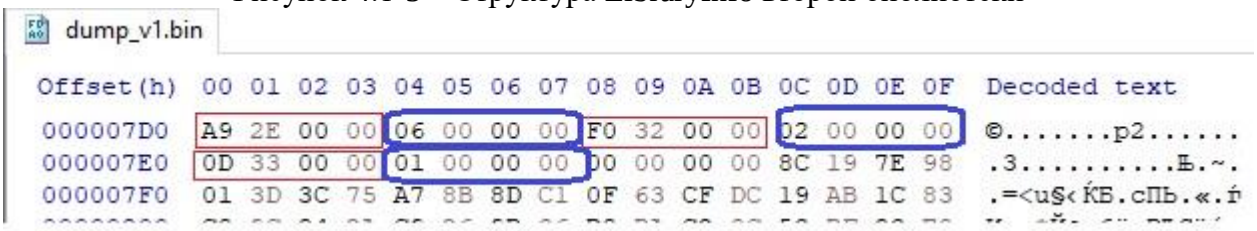

Рисунок 4.1-6 – Таблица имен функций второй библиотеки

Осталось проделать те же действия для третьей библиотеки. На рисунке 4.1-7 приведена структура **LibraryInfo** (смещение *0x00000539*), а на рисунке 4.1-8 – таблица имен второй импортируемой библиотеки (смещение *0x000008E4*), из которой импортируется **пять** функций. Синим на рисунке отмечены количества параметров функций: 6, 2, 3, 4 и 2.

dump_v1.bin

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000530	55	64	C6	E5	36	A9	81	42	F3	A3	28	00	00	E4	08	00	UdЖе6©ЃByJ(..д..
00000540	00	2D	0C	00	00	4E	87	12	A5	6E	C1	D1	21	B3	5B	DF	.-...N‡.ҐnБС!і[Я
00000550	DF	E6	FD	70	F6	8D	16	8C	E8	53	DF	80	21	D6	58	86	ЯжэрцЌ.ЊиSЯЂ!ЦX†

Рисунок 4.1-7 – Структура LibraryInfo третьей библиотеки

dump_v1.bin

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
000008E0	5D	11	5E	57	C2	2D	00	00	06	00	00	00	DF	34	00	00	].^WB-.....Я4..
000008F0	02	00	00	00	D7	35	00	00	03	00	00	00	7B	37	00	00	....Ч5.....{7..
00000900	04	00	00	00	D0	2B	00	00	02	00	00	00	00	00	00	00	....Р+..........
00000910	8D	DC	AC	CD	76	7A	C3	3F	CF	D1	6C	4C	2B	5E	19	15	ЌЬ¬HvzГ?ПСlL+^..

Рисунок 4.1-8 – Таблица имен функций третьей библиотеки

Таким образом, наименьшее количество функций (три) импортируется из **второй** библиотеки. Теперь необходимо найти ее название, а также названия и количество аргументов всех функций. На рисунке 4.1-5 в первом прямоугольнике выделено смещение до строки с названием библиотеки. Перейдя по смещению *0x0000278C* получим нужное значение (см. рисунок 4.1-9) – *cShv.dll.*

dump_v1.bin

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00002780	15	82	6C	77	EC	24	44	4A	EF	33	22	F1	53	68	76	2E	.,lwм$DJп3"сShv.
00002790	64	6C	6C	00	6E	50	60	FC	AD	A9	97	CB	AD	9F	26	DA	dll.nP`ь.©—Л.ц&Ъ
000027A0	BB	92	15	29	70	C7	94	E9	99	48	7A	18	4D	08	E0	9F	»’.)pЗ”й™Hz.M.аџ

Рисунок 4.1-9 – Название второй библиотеки

Далее необходимо получить названия импортируемых функций. Соответствующие смещения представлены на рисунке 4.6: *0x00002EA9, 0x000032F0* и *0x0000330D*. По указанным смещениям расположены искомые строки: (см. рисунок 4.1-10, рисунок 4.1-11 и рисунок 4.1-12) – **swap, GetLastError, GetEvent**

dump_v1.bin

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00002EA0	31	C7	2B	5F	54	33	C4	07	C3	73	77	61	70	00	D5	B9	1З+_Т3Д.Гswap.Х№
00002EB0	8D	0F	8D	38	AF	A5	13	22	3A	3C	7A	2E	F2	14	BE	23	Ќ.Ќ8ЇҐ.":<z.т.s#

Рисунок 4.1-10 – Имя первой функции

dump_v1.bin

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
000032F0	47	65	74	4C	61	73	74	45	72	72	6F	72	00	E7	0C	75	GetLastError.з.u
00003300	C6	75	37	C0	85	97	66	D3	F7	1A	67	39	ED	47	65	74	Жu7А…—fУч.g9нGet

Рисунок 4.1-11 – Имя второй функции

dump_v1.bin

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00003300	C6	75	37	C0	85	97	66	D3	F7	1A	67	39	ED	47	65	74	Жu7А…—fУч.g9нGet
00003310	45	76	65	6E	74	00	66	EC	1C	2E	EA	CE	83	1C	BE	ED	Event.fм..кОѓ.sн
00003320	D6	8F	21	87	BB	D1	70	E9	BF	53	AE	01	CA	B9	DF	11	ЦЏ!‡»СpйїS®.К№Я.

Рисунок 4.1-12 – Имя третьей функции

Осталось получить ординалы этих функций. Таблица ординалов расположена по смещению *0x000BF5*(третий прямоугольник на рисунке 4.1-5). Зная формат таблицы **OrdinalsTable** получаем ординалы функций: **2**, **7**, **16** (см. рисунок 4.1-13).

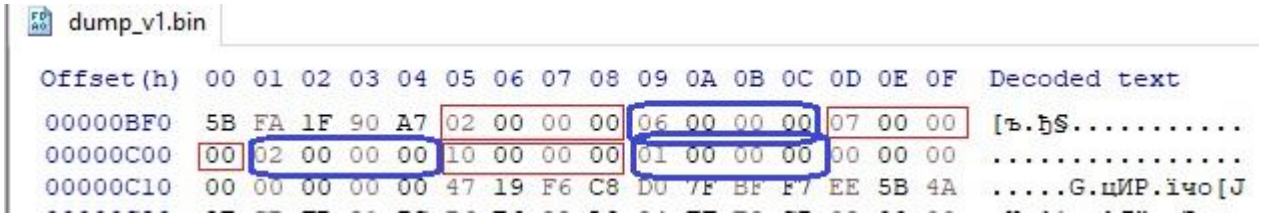

Рисунок 4.1-13 – Ординалы импортируемых функций

Количество параметров каждой функции можно получить либо из таблицы имен (рисунок 4.1-6), либо из таблицы ординалов (рисунок 4.1-13) (на рисунках отмечены синим овалом): **6**, **2**, **1**. Дублирование количества параметров также позволяет легко проверить правильность решения – значения должны совпадать.

Ответ: Shv.dll – swap(0x02): 6, GetLastError(0x07): 2, GetEvent(0x10): 1.

Способ 2

Для решения задачи также можно написать несложную программу: объявить соответствующие структуры и приведением типов получить нужные значения. Код такой программы на языке программирования C++ приведен в листинге 4.1-1.

Листинг 4.1-1. Листинг программы на языке программирования C++

```
struct NameInfo
{
  int NameOffset;
  int ParamsCount;
};
struct OrdinalInfo
{
  int Ordinal;
  int ParamsCount;
};
struct LibraryInfo
{
  int NameOffset;
  int NamesTableOffset;
  int OrdinalsInfoOffset;
};
struct Import
{
  int LibInfoOffset;
};

int main()
{
    char buffer[30000];
  // Считывание всего файла
  auto dumpFile = fopen("dump_v1.bin", "rb");
  fread(buffer, 1, 30000, dumpFile);
  fclose(dumpFile);

  auto libraries = (Import*)buffer;

  // Цикл по всем структурам LibraryInfo
  while(libraries->LibInfoOffset != 0)
```

```
  {
    auto library = (LibraryInfo*)&buffer[libraries->LibInfoOffset];

    // Вывод названия библиотеки
    std::cout << "Library: ";
    std::cout << &buffer[library->NameOffset] << std::endl;
    std::cout << "Functions:" << std::endl;

    // Получение таблиц имен и ординалов функций
    auto names = (NameInfo*)&buffer[library->NamesTableOffset];
    auto ordinals = (OrdinalInfo*)&buffer[library->OrdinalsInfoOffset];

    // Цикл по всем импортируемым функциям
    while(names->NameOffset != 0)
    {
      // Вывод имени, количества параметров и ординала
      std::cout << "Name: " << &buffer[names->NameOffset] << " ";
      std::cout << "Params: " << names->ParamsCount << " ";
      std::cout << "Ordinal: " << ordinals->Ordinal << std::endl;
      ++names;
      ++ordinals;
    }
    ++libraries;
  }
}
```

В результате выполнения программа выводит на экран следующее.

```
Library: Dstp.dll
Functions:
Name: OpenFile Params: 4 Ordinal: 6
Name: CreatePipe Params: 2 Ordinal: 7
Name: CreateProcess Params: 5 Ordinal: 22
Name: GetErrorMessage Params: 3 Ordinal: 32
Library: Shv.dll
Functions:
Name: swap Params: 6 Ordinal: 2
Name: GetLastError Params: 2 Ordinal: 7
Name: GetEvent Params: 1 Ordinal: 16
Library: Ppte.dll
Functions:
Name: Search Params: 6 Ordinal: 2
Name: random Params: 2 Ordinal: 8
Name: Print Params: 3 Ordinal: 26
Name: CopyFile Params: 4 Ordinal: 37
Name: CreateThread Params: 2 Ordinal: 41
```

Однако поскольку количество импортируемых библиотек в представленном дампе файла небольшое, автоматизировать поиск нецелесообразно.

Ответ: Shv.dll – swap(0x02): 6, GetLastError(0x07): 2, GetEvent(0x10): 1.

Вариант 2

В разделе импорта заголовка исполняемого файла содержится информация о подключаемых библиотеках (DLL) и импортируемых из них функциях. Для подмены одной из DLL необходимо, чтобы имя библиотеки и набор функций совпадали с именем библиотеки и набором функций, описанными в разделе импорта исполняемого файла. Для упрощения разработки подменяемой библиотеки DLL, из всех библиотек выбирают ту, из которой

импортируется наименьшее количество функций.

Из предоставленного образа раздела импорта определите имя библиотеки, из которой импортируется наименьшее количество функций.

В ответе укажите имя библиотеки DLL, имена импортируемых из нее функций и количество аргументов каждой из таких функций.

Структура раздела импорта показана на рисунке.

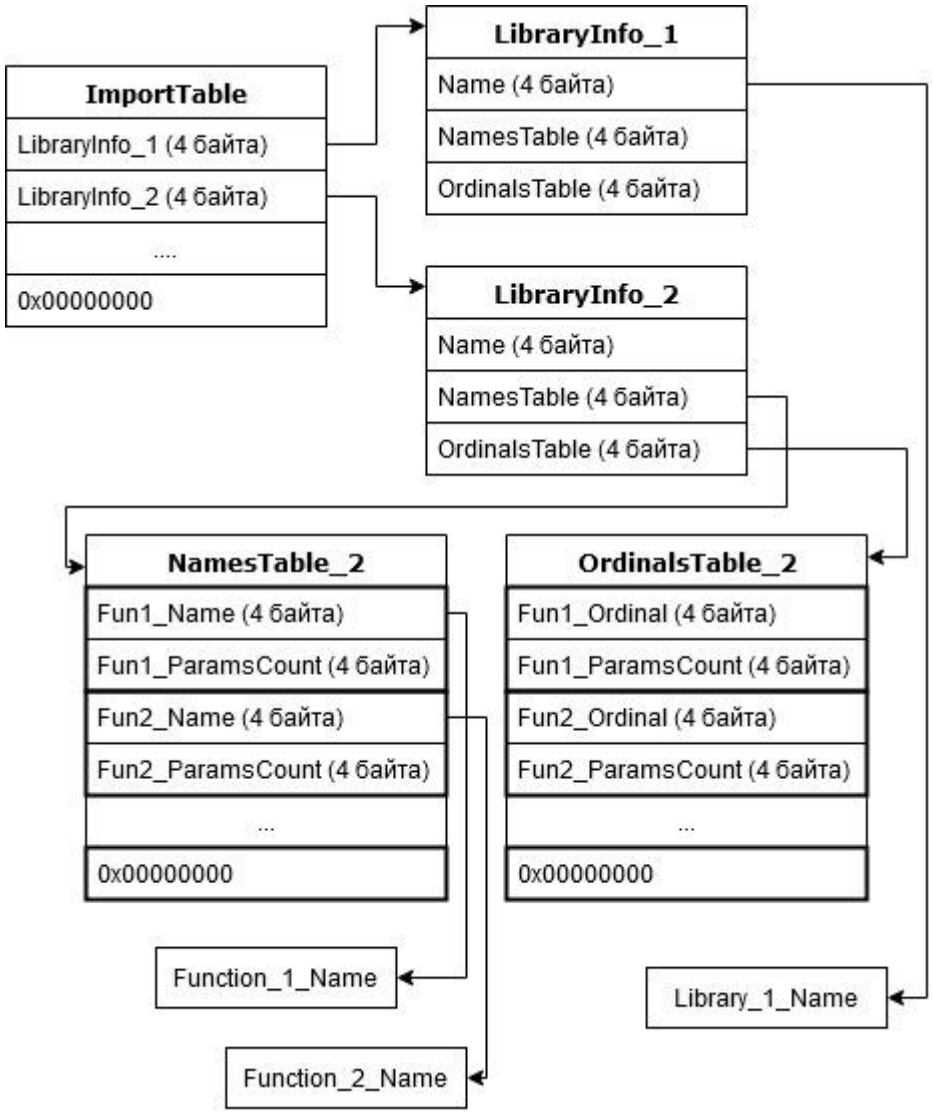

Рисунок. Структура раздела импорта

ImportTable – таблица импорта. Хранится в начале раздела импорта. Содержит адреса (4-х байтовые) на структуры *LibraryInfo*. В конце таблицы записывается 4 нулевых байта (0x00000000).

LibraryInfo – структура, содержащая информацию о библиотеке. Содержит поля:

Name – адрес строки (4 байта), содержащей имя библиотеки. Строка заканчивается нулевым байтом ('\0');

NamesTable – адрес таблицы имен импортируемых функций (4 байта);

OrdinalsTable – адрес таблицы идентификаторов импортируемых функций (4 байта).

NamesTable – таблица имен импортируемых функций. Каждая запись содержит следующие параметры:

Fun_Name – адрес строки (4 байта), содержащей имя функции. Строка заканчивается нулевым байтом ('\0');

Fun_ParamsCount – количество аргументов функции (4 байта).

В конце таблицы записывается 4 нулевых байта (0x00000000).

OrdinalsTable – таблица идентификаторов импортируемых функций. Каждая запись содержит следующие параметры:

Fun_Ordinal – идентификатор функции (4 байта);

Fun_ParamsCount – количество аргументов функции (4 байта).

В конце таблицы записывается 4 нулевых байта (0x00000000).

Все адреса и числовые значения хранятся в формате *Little-Endian*. Адреса указываются относительно начала раздела импорта, адрес которого считается равным 0x00000000.

К задаче прилагается:

файл образа раздела импорта dump_v2.bin.

Решение

Для решения задачи необходимо отыскать все импортируемые библиотеки, для каждой из них определить количество импортируемых функций. После этого уже возможно определить, какая именно библиотека является наиболее подходящей, и для нее необходимо получить названия всех функций.

Элементами структур являются смещения относительно начала дампа, соответственно для получения необходимых данных нужно корректно определить адреса всех структур **LibraryInfo**, **NamesTable**, **OrdinalsTable**, а также адреса строк. Можно предложить 2 способа решения:

– ручной поиск (наиболее удобно использовать HEX-редактор);

– автоматизированный с использованием программы.

Способ 1.

Откроем предоставленный дамп в HEX-редакторе. Зная формат структуры **ImportTable** несложно выделить (см. рисунок 4.2-1) смещения трех структур **LibraryInfo**: *0x000003E8*, *0x0000044A* и *0x000004EA* (байты расположены в обратном порядке согласно *little-endian*).

dump_v2.bin

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000   E8 03 00 00 4A 04 00 00 EA 04 00 00 00 00 00 00  и...J...к.......
00000010   98 8C BB 79 F5 70 25 30 1B EF 5D 20 24 38 78 22  .Ћ»yхp%0.п] $8x"
00000020   26 47 D7 9E 0C CC 09 59 EC B8 59 06 5A 6E 45 80  &GЧћ.М.YмёY.ZnEЂ
00000030   E6 2A D7 0B E6 EA 5F 0F 83 A3 76 72 F1 6B 5A F0  ж*Ч.жк_.ѓЈvrckZр
00000040   BD 0F 6C D0 F2 48 5A 4F FE 33 A6 A2 30 0C C0 09  S.lРтHZOю3¦ў0.А.
00000050   52 1C ED F5 7F 89 6F 10 A4 72 95 35 A2 A9 BD A5  R.нх.‰o.¤r•5ў©SЃ
00000060   74 C1 E8 34 D5 69 90 9F 42 C7 F3 76 97 98 40 41  tБи4Xiђџ‚Зуv—.@A
```

Рисунок 4.2-1 – Структура ImportTable

Используя инструмент перехода по смещению (*Search – Go to* или горячее сочетание *Alt+G*) перейдем по адресу первой структуры **LibraryInfo.**

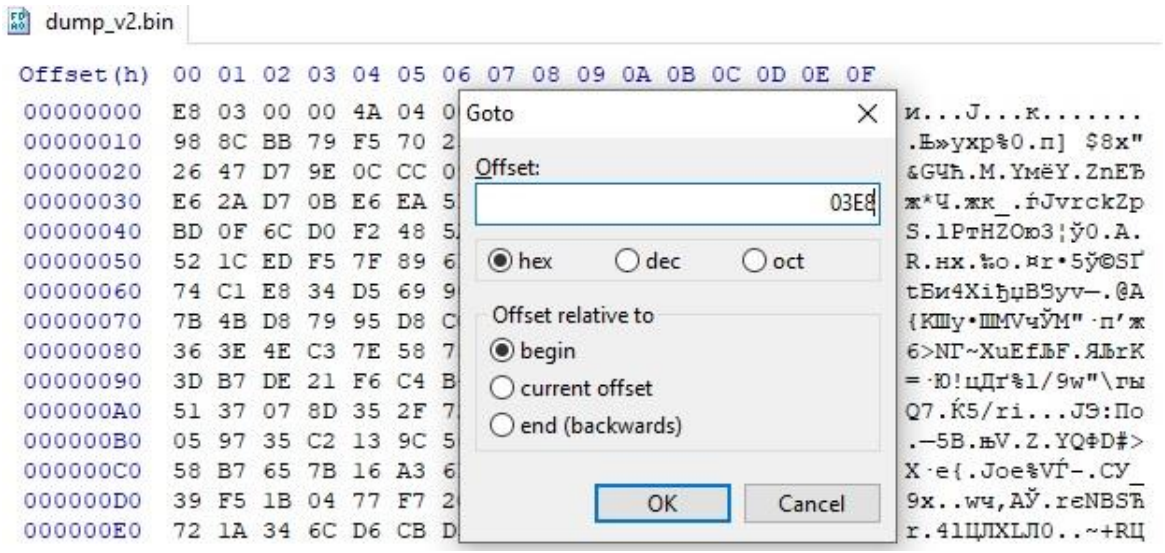

Рисунок 4.2-2 – Переход по смещению

Зная формат структуры **LibraryInfo** можно выделить адрес названия библиотеки, адреса таблицы имен и ординалов импортируемых функций.

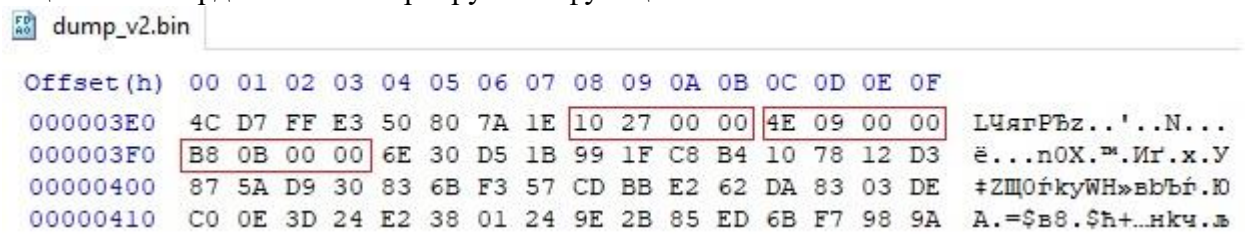

Рисунок 4.2-3 – Структура LibraryInfo первой библиотеки

На текущем этапе решения нет необходимости получать названия библиотеки, пока достаточно определить только количество импортируемых функций. Для этого нужно перейти либо к таблице имен, либо к таблице ординалов. Перейдем по адресу *0x0000094E* и, зная формат **NamesTable**, посчитаем количество импортируемых из первой библиотеки функций.

На рисунке 4.2-4 отмечены смещения имен импортируемых функций, количество которых равно **пяти**. Синим на рисунке отмечены количества параметров функций: 3, 3, 3, 4 и 2.

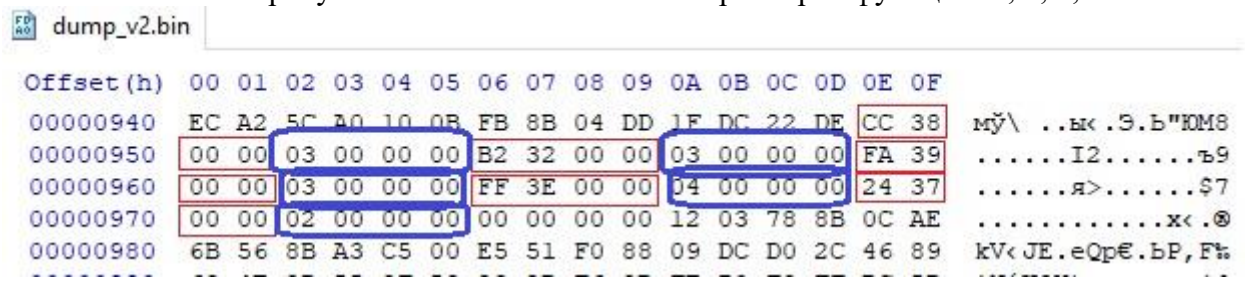

Рисунок 4.2-4 – Таблица имен функций первой библиотеки

Аналогичным образом отыщем количество импортируемых функций для второй и третьей библиотек.

На рисунке 4.2-5 приведена структура **LibraryInfo** (смещение *0x0000044A*), а на рисунке 4.2-6 – таблица имен второй импортируемой библиотеки (смещение *0x00000870*), из которой импортируется **четыре** функций. Синим на рисунке отмечены количества параметров функций: 3, 6, 4, 2.

dump_v2.bin

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000440   0D EE 29 85 F9 00 87 4B C4 97 DB 28 00 00 70 08  .о)…щ.‡КД—Ы(..p.
00000450   00 00 1A 0C 00 00 BC 8E 4E C0 81 E3 12 86 0E 06  ......jЋNAЃг.†..
00000460   83 7E 4F A2 63 90 43 DA 59 1C B4 F8 D8 1C 68 F1  ѓ~OўcђCЪY.ґшШ.hc
00000470   BB B9 F6 3E FB D1 DD 73 6E 8F 8F CD 29 C2 2D 14  »№ц>ыСЭsnЏЏН)В-.
```

Рисунок 4.2-5 – Структура LibraryInfo второй библиотеки

dump_v2.bin

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000870   41 3E 00 00 03 00 00 00 E5 2D 00 00 06 00 00 00  A>......е-......
00000880   8E 2C 00 00 04 00 00 00 38 36 00 00 02 00 00 00  Ћ,......86......
00000890   00 00 00 00 60 0E E3 26 5C 51 B9 02 9D E2 A9 72  ....`.г&\Q№.ќв©r
```

Рисунок 4.2-6 – Таблица имен функций второй библиотеки

Осталось проделать те же действия для третьей библиотеки. На рисунке 4.2-7 приведена структура **LibraryInfo** (смещение *0x000004EA*), а на рисунке 4.2-8 – таблица имен второй импортируемой библиотеки (смещение *0x000007D0*), из которой импортируется **три** функции. Синим на рисунке отмечены количества параметров функций: 6, 1, 6.

dump_v2.bin

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
000004E0   DE 2A 32 39 31 81 5B CD E1 5A 1C 29 00 00 D0 07  Ю*291Ѓ[НбZ.)..Р.
000004F0   00 00 99 0C 00 00 BB BE 17 E3 5B 84 17 FA 1C 0E  ..™...»ѕ.г[„.ъ..
00000500   F7 5D 6C F2 47 CC A8 B0 6F 74 5D 04 61 14 F5 3D  ч]lтGМЁ°ot].a.x=
```

Рисунок 4.2-7 – Структура LibraryInfo третьей библиотеки

dump_v2.bin

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
000007D0   C2 34 00 00 06 00 00 00 74 33 00 00 01 00 00 00  В4......t3......
000007E0   27 2B 00 00 06 00 00 00 00 00 00 00 81 69 EF 23  '+..........Ѓiп#
000007F0   F7 AC CF 72 25 C9 E7 82 1E 2B D8 C4 42 E1 61 FA  ч¬Пr%Йз‚.+ШДВбаъ
00000800   EE A8 52 78 47 F4 21 C2 C1 7C F1 1F A9 68 AD 46  оЁRxGф!ВБ|с.©h.F
```

Рисунок 4.2-8 – Таблица имен функций третьей библиотеки

Таким образом, наименьшее количество функций (три) импортируется из **третьей** библиотеки. Теперь необходимо найти ее название, а также названия и количество аргументов всех функций.

На рисунке 4.2-7 в первом прямоугольнике выделено смещение до строки с названием библиотеки. Перейдя по смещению *0x0000291C* получим нужное значение (см. рисунок 4.2-9) – *Ppte.dll.*

dump_v2.bin

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00002910   05 97 3B 04 6E 39 94 9B DA A8 5F 9D 50 70 74 65  .—;.n9”›ЪЁ_ќPpte
00002920   2E 64 6C 6C 00 D6 98 3C 7C A0 C3 DF 2D EF 0B D5  .dll.Ц.<| ГЯ-п.Х
00002930   7D FC 13 B8 19 AA 48 FC 54 2F D6 8D 16 BF D5 BD  }ь.ё.ЄНьТ/ЦЌ.їXS
```

Рисунок 4.2-9 – Название второй библиотеки

Далее необходимо получить названия импортируемых функций. Соответствующие смещения представлены на рисунке 4.2-8: *0x000034C2, 0x00003374* и *0x00002B27*. По указанным смещениям расположены искомые строки: (см. рисунок 4.2-10, рисунок 4.2-11 и рисунок 4.2-12) – **Connect, CreateMutex, Terminate.**

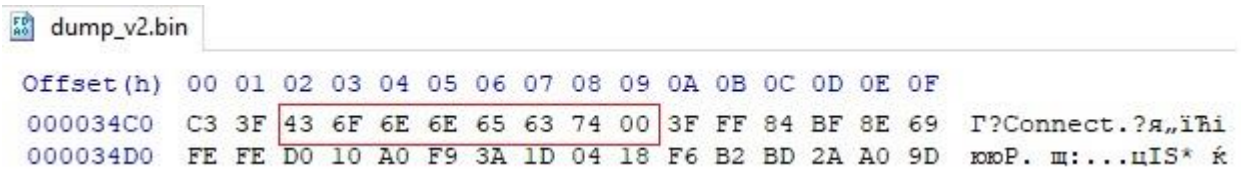

Рисунок 4.2-10 – Имя первой функции

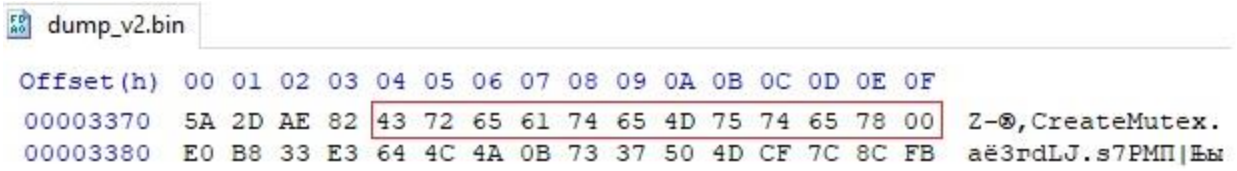

Рисунок 4.2-11 – Имя второй функции

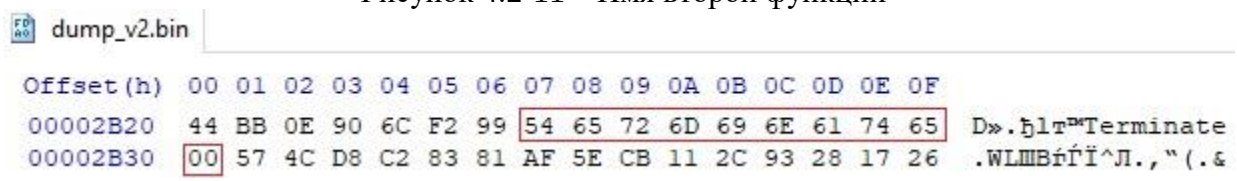

Рисунок 4.2-12 – Имя третьей функции

Осталось получить ординалы этих функций. Таблица ординалов расположена по смещению *0x000C99* (третий прямоугольник на рисунке 4.2-7). Зная формат таблицы **OrdinalsTable** получаем ординалы функций: **4**, **6**, **9** (см. рисунок 4.2-13). Синим на рисунке отмечены количества параметров функций: 6, 1, 6.

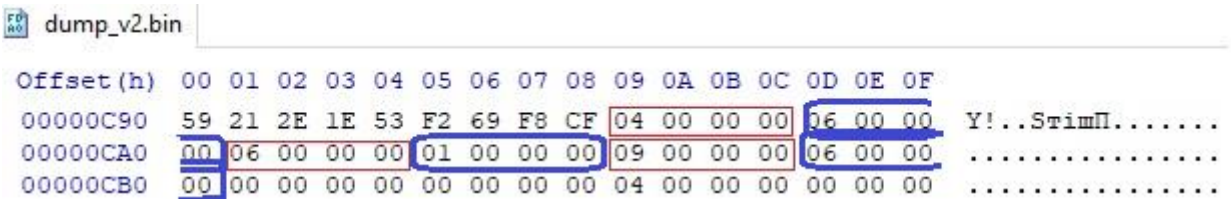

Рисунок 4.2-13 – Ординалы импортируемых функций

Количество параметров каждой функции можно получить либо из таблицы имен (рисунок 4.2-8), либо из таблицы ординалов (рисунок 4.2-13): **6**, **1**, **6**. Дублирование количества параметров также позволяет легко проверить правильность решения – значения должны совпадать.

Ответ: Ppte.dll – Connect(0x04): 6, CreateMutex(0x06): 1, Terminate(0x09): 6.

Способ 2

Для решения задачи также можно написать несложную программу: объявить соответствующие структуры и приведением типов получить нужные значения. Код такой программы на языке программирования C++ приведен в листинге 4.2-1.

Листинг 4.2-1. Листинг программы на языке программирования C++

```
struct NameInfo
{
  int NameOffset;
  int ParamsCount;
};
struct OrdinalInfo
{
  int Ordinal;
  int ParamsCount;
};
struct LibraryInfo
{
  int NameOffset;
```

```
  int NamesTableOffset;
  int OrdinalsInfoOffset;
};
struct Import
{
  int LibInfoOffset;
};

int main()
{
    char buffer[30000];
  // Считывание всего файла
  auto dumpFile = fopen("dump_v2.bin", "rb");
  fread(buffer, 1, 30000, dumpFile);
  fclose(dumpFile);

  auto libraries = (Import*)buffer;

  // Цикл по всем структурам LibraryInfo
  while(libraries->LibInfoOffset != 0)
  {
    auto library = (LibraryInfo*)&buffer[libraries->LibInfoOffset];

    // Вывод названия библиотеки
    std::cout << "Library: ";
    std::cout << &buffer[library->NameOffset] << std::endl;
    std::cout << "Functions:" << std::endl;

    // Получение таблиц имен и ординалов функций
    auto names = (NameInfo*)&buffer[library->NamesTableOffset];
    auto ordinals = (OrdinalInfo*)&buffer[library->OrdinalsInfoOffset];

    // Цикл по всем импортируемым функциям
    while(names->NameOffset != 0)
    {
      // Вывод имени, количества параметров и ординала
      std::cout << "Name: " << &buffer[names->NameOffset] << " ";
      std::cout << "Params: " << names->ParamsCount << " ";
      std::cout << "Ordinal: " << ordinals->Ordinal << std::endl;
      ++names;
      ++ordinals;
    }
    ++libraries;
  }
}
```

В результате выполнения программа выводит на экран следующее.

```
Library: Vmxops.dll
Functions:
Name: LoadLibrary Params: 3 Ordinal: 8
Name: OpenFile Params: 3 Ordinal: 9
Name: SubscribeForEvent Params: 3 Ordinal: 10
Name: strcpy Params: 4 Ordinal: 12
Name: GetErrorMessage Params: 2 Ordinal: 20
Library: Trsx.dll
Functions:
Name: swap Params: 3 Ordinal: 3
Name: CreateProcess Params: 6 Ordinal: 7
Name: random Params: 4 Ordinal: 9
```

```
Name: CopyFile Params: 2 Ordinal: 26
Library: Ppte.dll
Functions:
Name: Connect Params: 6 Ordinal: 4
Name: CreateMutex Params: 1 Ordinal: 6
Name: Terminate Params: 6 Ordinal: 9
```

Однако поскольку количество импортируемых библиотек в представленном дампе файла небольшое, автоматизировать поиск нецелесообразно.

Ответ: Ppte.dll – Connect(0x04): 6, CreateMutex(0x06): 1, Terminate(0x09): 6.

Задача 5. Web-сайт

Вариант 1

Пользователь хранит на сервере секретное слово, доступ к которому можно получить, авторизовавшись через web-сайт. Сервер выдаст секретное слово только в том случае, если ему будет отправлена верная зашифрованная последовательность, сформированная из логина и пароля. Чтобы не забыть логин и пароль, пользователь оставил себе подсказки на сайте.

Определите секретное слово.

К задаче прилагается:
папка с содержимым web-страницы.

Решение

Задача предполагает 2 способа решения:
1) аналитический;
2) анализ исходного кода (reverse-engineering).

Способ 1.

На открывшейся web-странице есть следующие активные поля (рисунок 5.1-1):
–логин (1);
–пароль (2);
– ссылка «Помнишь меня?» (3);
–ссылка «Забыли код?» (4).

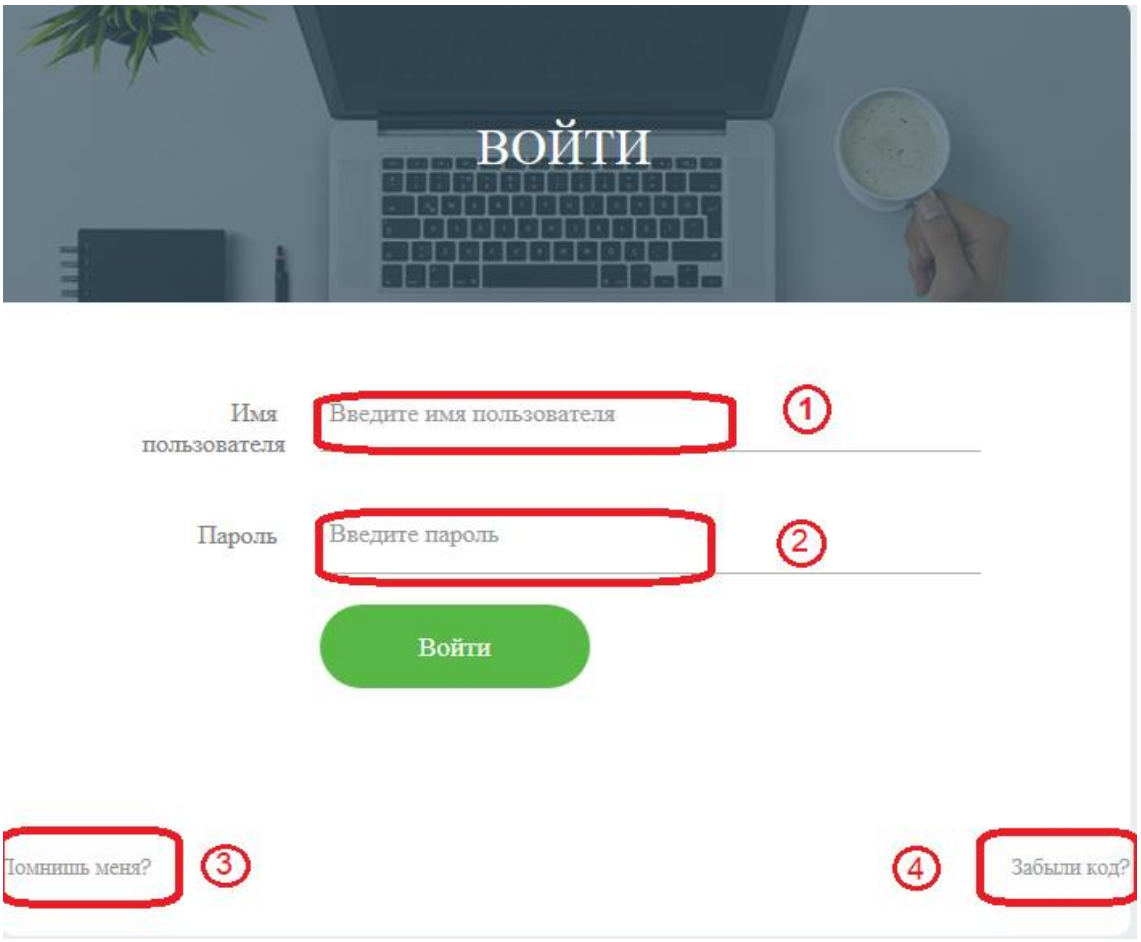

Рисунок 5.1-1 Внешний вид web-страницы

Известно, что логин состоит только из букв, пароль состоит только из цифр. Длина логина и пароля должны совпадать.

Нажав на ссылку «Помнишь меня?», страница выдает сообщение:

`z6m7n3w5`

Нажав на ссылку «Забыли код?», страница выдает сообщение:

`alfredojustmybingvxchwqkpz`
`01234567890123456789012345`

Можно заметить, что в первой строке содержатся все символы английского алфавита без повторений (26 символов). Каждому символу соответствует свой номер из нижней строки: `a`–0, `l`–1, `f`–2, … `z`–5.

Можно предположить, что это не случайность и не просто так. Поэтому можно попробовать взять сообщение `z6m7n3w5`, использовать его как логин, а в качестве пароля подобрать цифры и буквы из второй подсказки «Забыли код?»:

z–5, 6–o, m–1, 7–j, n–5, 3–r, w–1, 5–d.

Вводим следующие данные:

логин – `z6m7n3w5`,

пароль – `5o1j5r1d`

В результате на странице отобразится код: *MANAGEMENT* (рисунок 5.1-2).

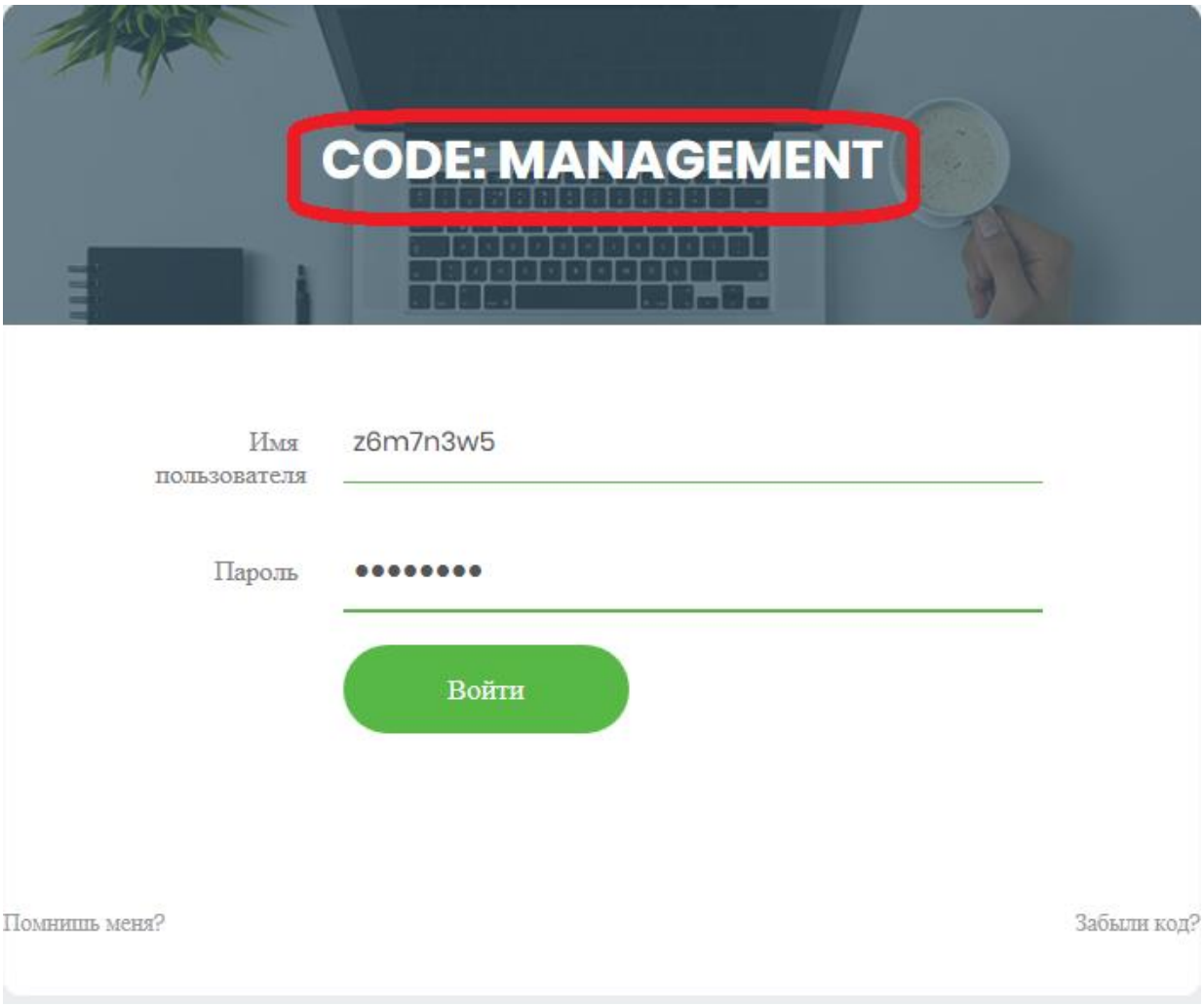

Рисунок 5.1-2 – Содержимое web-страницы после правильно введенных логина и пароля

Способ 2.

Проанализировав код страницы можно увидеть, что к странице подключены следующие JavaScript-файлы:

```
<script src="script/proof.js"></script>
<script src="script/script.js"></script>
<script src="js/script.js"></script>
```

Остальные файлы относятся к платформе для корректного отображения объектов.

Рассмотрим содержимое файла `js/script.js`. В этом файле устанавливаются обработчики событий click на объекты web-страницы:

- при нажатии на ссылку «Помнишь меня?» выводится сообщение с результатом выполнения функции `forgetCode();`
- при нажатии на ссылку «Забыли код?» выводится сообщение с выражением `getBaseString()+'\n012345678901234567890123456789012345';`
- при нажатии на кнопку «Войти» вызывается функция `myfunc()`, которая описана в этом же файле.

Интерес вызывает именно функция `myfunc()`. Рассмотрим её подробнее (листинг 5.1-1).

Листинг 5.1-1 – Содержимое функции myfunc()

```
1    function myfunc() {
2        let code = '';
3        if (username_form.value && passwd_form.value)
4            code = getCode(username_form.value, passwd_form.value);
5        if (code && code.length > 0)
6            code_label.innerText = code;
```

```
7       else
8           code_label.innerText = 'ВОЙТИ';
9   }
```

В строке 3 проверяется на непустые значения полей логин и пароль, после чего вызывается функция `getCode()`. Результат функции отображается на web-странице.

Проанализируем функцию `getCode()`. Она реализована в файле `script/script.js` (листинг 5.1-2).

Листинг 5.1-2 – Содержимое функции getCode()

```
1.  function getCode(username, password) {
2.      if (username.length !== password.length && username.length > 0 &&
    password.length > 0) {
3.          alert('Длина имени пользователя и пароля не совпадают!');
4.          return String('');
5.      }
6.      const str = getBaseString();
7.      const nums = '0123456789';
8.      let res = Bar();
9.      let check = Foo();
10.     let code = '';
11.     let pos1 = -1;
12.     let pos2 = -1;
13.     for (i = 0; i < username.length; i++) {
14.         if (i % 2) {
15.             pos1 = nums.indexOf(username[i]);
16.             pos2 = str.indexOf(password[i].toLowerCase());
17.         } else {
18.             pos1 = str.indexOf(username[i].toLowerCase());
19.             pos2 = nums.indexOf(password[i]);
20.         }
21.         if (pos1 === -1 || pos2 === -1) {
22.             code += '-';
23.         }
24.         else {
25.             code += str[(pos1 + pos2) % str.length];
26.         }
27.     }
28.     if (code === check) {
29.         return res;
30.     }
31.     else {
32.         return code;
33.     }
34. }
```

В самой функции интерес представляет лишь последняя конструкция if (строки 28-33). В этих строках и формируется результат. Если условие в строке 28 верное, то результатом выполнения функции является переменная `res`, иначе переменная `code`. Значение переменной `res` получается из функции `Bar()` (строка 8).

Проанализируем функцию `Bar()`, которая реализована в файле `script/proof.js` (листинг 5.1-3).

Листинг 5.1-3 – Содержимое функции Bar()

```
1.  function Bar() {
2.      const arr = [67, 110, 98, 98, 54, 27, 71, 90, 102, 88, 93, 90,
    97, 88, 96, 101];
```

```
3.      let res = '';
4.      for (i = 0; i < arr.length; i++) {
5.          res = res + String.fromCharCode(arr[i] + i);
6.      }
7.      return res;
8.  };
```

Функция преобразовывает массив чисел в символы в соответствии с ASCII-таблицей. Можно запустить в отладчике эту функцию и посмотреть результат ее выполнения (ответом будет строка “*CODE: MANAGEMENT*”), а можно в функции `getCode()` в строке 32 вместо строки

`return code;`

записать

`return Bar();` ИЛИ `return res;`

Тогда при любых значениях логина и пароля с одинаковой длиной получится следующий результат (рисунок 5.1-3).

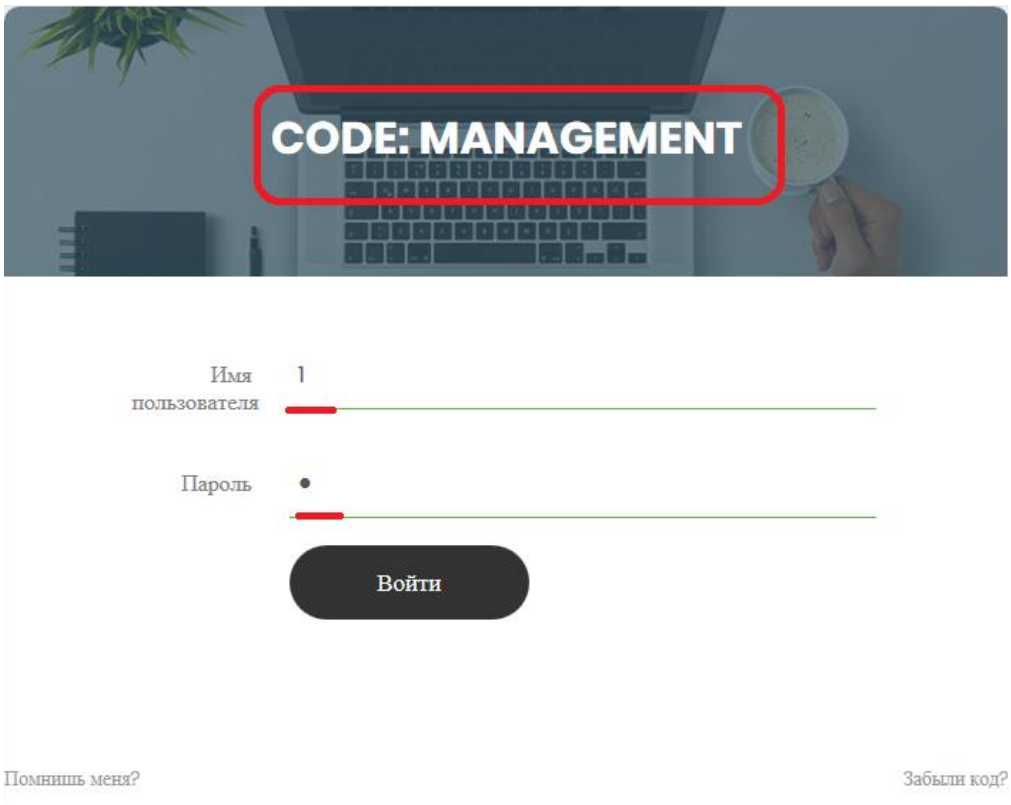

Рисунок 5.1-3 – Содержимое web-страницы с измененной функцией getCode()

Ответ: MANAGEMENT.

Вариант 2

Пользователь хранит на сервере секретное слово, доступ к которому можно получить, авторизовавшись через web-сайт. Сервер выдаст секретное слово только в том случае, если ему будет отправлена верная зашифрованная последовательность, сформированная из логина и пароля. Чтобы не забыть логин и пароль, пользователь оставил себе подсказки на сайте.

Определите секретное слово.

К задаче прилагается:
папка с содержимым web-страницы.

Решение

Задача предполагает 2 способа решения:

1) аналитический;

2) анализ исходного кода (reverse-engineering).

Способ 1.

На открывшейся web-странице есть следующие активные поля (рисунок 5.2-1):

–логин (1);

–пароль (2);

– ссылка «Помнишь меня?» (3);

–ссылка «Забыли код?» (4).

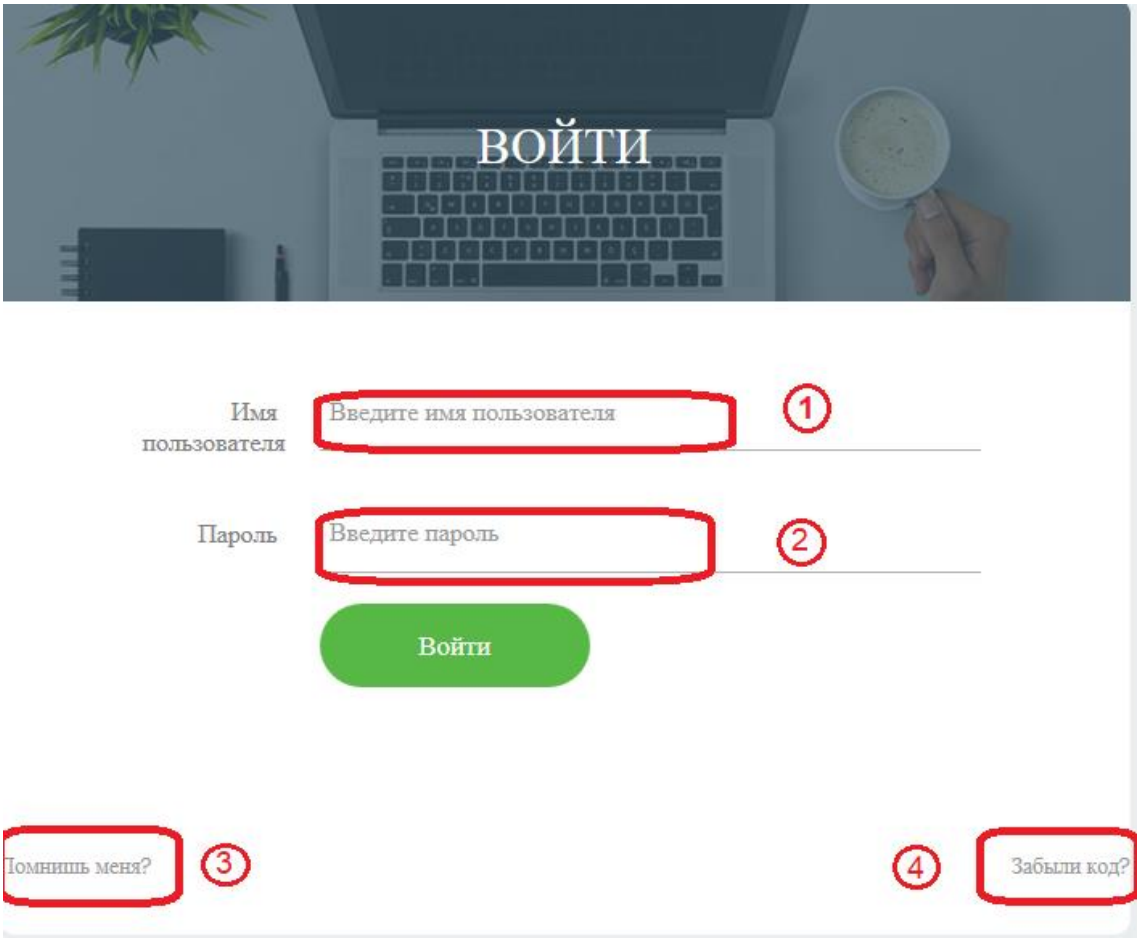

Рисунок 5.2-1 Внешний вид web-страницы

Известно, что логин состоит только из букв, пароль состоит только из цифр. Длина логина и пароля должны совпадать.

Нажав на ссылку «Помнишь меня?», страница выдает сообщение:

x7m3d4q9

Нажав на ссылку «Забыли код?», страница выдает сообщение:

thequickbrownfxjmpsvlazydg
01234567890123456789012345

Можно заметить, что в первой строке содержатся все символы английского алфавита без повторений (26 символов). Каждому символу соответствует свой номер из нижней строки: t–0, h–1, e–2, … q–5.

Можно предположить, что это не случайность и не просто так. Поэтому можно попробовать взять сообщение *x7m3d4q9*, использовать его как логин, а в качестве пароля подобрать цифры и буквы из второй подсказки «Забыли код?»:

x–4, 7–k, m–6, 3–q, d–4, 4–u, q–3, 9–r.

Вводим следующие данные:
логин – *x7m3d4q9*,
пароль – *4k6q4u3r*
В результате на странице отобразится код: *ROCKET* (рисунок 5.2-2).

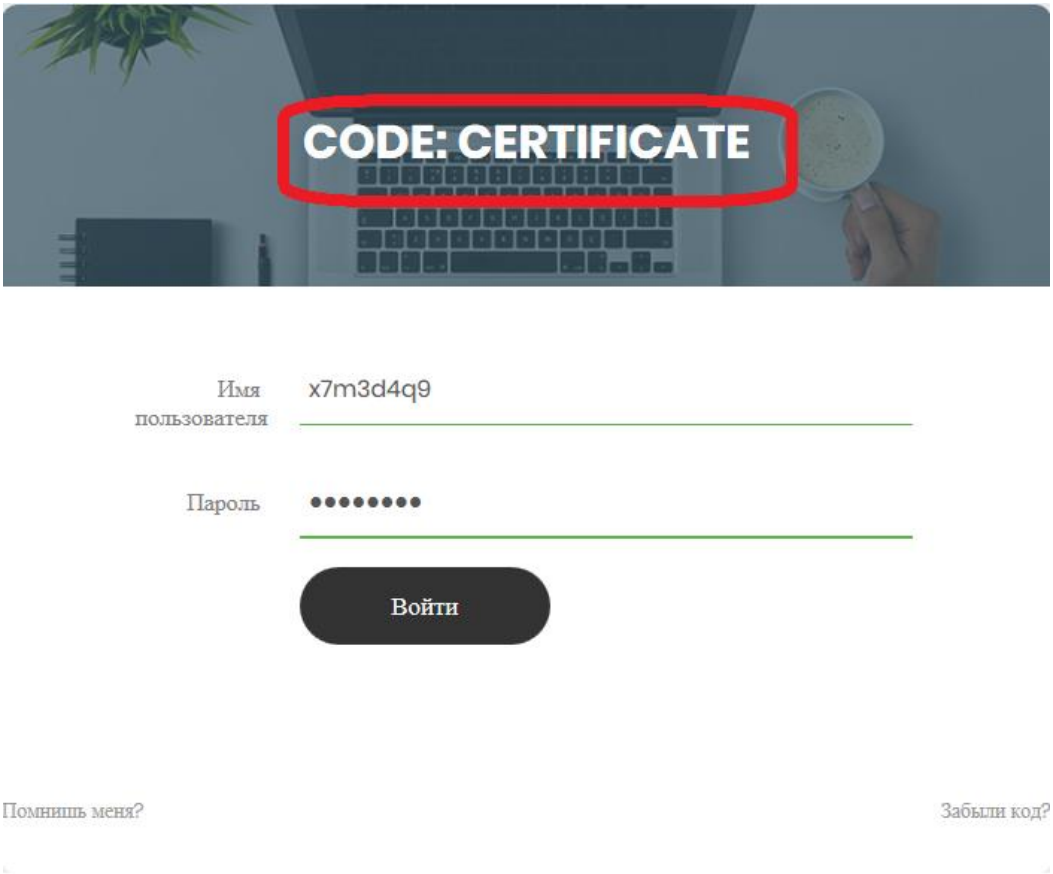

Рисунок 5.2-2 – Содержимое web-страницы после правильно введенных логина и пароля

Способ 2.

Проанализировав код страницы можно увидеть, что к странице подключены следующие JavaScript-файлы:

```
<script src="script/proof.js"></script>
<script src="script/script.js"></script>
<script src="js/script.js"></script>
```

Остальные файлы относятся к платформе для корректного отображения объектов.

Рассмотрим содержимое файла `js/script.js`. В этом файле устанавливаются обработчики событий click на объекты web-страницы:

- при нажатии на ссылку «Помнишь меня?» выводится сообщение с результатом выполнения функции `forgetCode()`;
- при нажатии на ссылку «Забыли код?» выводится сообщение с выражением `getBaseString()+'\n012345678901234567890123456789012345'`;
- при нажатии на кнопку «Войти» вызывается функция `myfunc()`, которая описана в этом же файле.

Интерес вызывает именно функция `myfunc()`. Рассмотрим её подробнее (листинг 5.2-1).

Листинг 5.2-1 – Содержимое функции myfunc()

```
1    function myfunc() {
2        let code = '';
3        if (username_form.value && passwd_form.value)
4            code = getCode(username_form.value, passwd_form.value);
5        if (code && code.length > 0)
```

```
6           code_label.innerText = code;
7       else
8           code_label.innerText = 'ВОЙТИ';
9   }
```

В строке 3 проверяется на непустые значения полей логин и пароль, после чего вызывается функция `getCode()`. Результат функции отображается на web-странице.

Проанализируем функцию `getCode()`. Она реализована в файле `script/script.js` (листинг 5.2-2).

Листинг 5.2-2 – Содержимое функции getCode()

```
1.  function getCode(username, password) {
2.      if (username.length !== password.length && username.length > 0 &&
    password.length > 0) {
3.          alert('Длина имени пользователя и пароля не совпадают!');
4.          return String('');
5.      }
6.      const str = getBaseString();
7.      const nums = '0123456789';
8.      let res = Bar();
9.      let check = Foo();
10.     let code = '';
11.     let pos1 = -1;
12.     let pos2 = -1;
13.     for (i = 0; i < username.length; i++) {
14.         if (i % 2) {
15.             pos1 = nums.indexOf(username[i]);
16.             pos2 = str.indexOf(password[i].toLowerCase());
17.         } else {
18.             pos1 = str.indexOf(username[i].toLowerCase());
19.             pos2 = nums.indexOf(password[i]);
20.         }
21.         if (pos1 === -1 || pos2 === -1) {
22.             code += '-';
23.         }
24.         else {
25.             code += str[(pos1 + pos2) % str.length];
26.         }
27.     }
28.     if (code === check) {
29.         return res;
30.     }
31.     else {
32.         return code;
33.     }
34. }
```

В самой функции интерес представляет лишь последняя конструкция if (строки 28-33). В этих строках и формируется результат. Если условие в строке 28 верное, то результатом выполнения функции является переменная `res`, иначе переменная `code`. Значение переменной `res` получается из функции `Bar()` (строка 8).

Проанализируем функцию `Bar()`, которая реализована в файле `script/proof.js` (листинг 5.2-3).

Листинг 5.2-3 – Содержимое функции Bar()

```
1.  function Bar() {
```

```
2.      const arr = [67, 110, 98, 98, 54, 27,  61, 94, 106, 107, 95, 91,
    93, 86, 83, 101, 85];
3.      let res = '';
4.      for (i = 0; i < arr.length; i++) {
5.          res = res + String.fromCharCode(arr[i] + i);
6.      }
7.      return res;
8.  };
```

Функция преобразовывает массив чисел в символы в соответствии с ASCII-таблицей. Можно запустить в отладчике эту функцию и посмотреть результат ее выполнения (ответом будет строка “*CODE: CERTIFICATE*”), а можно в функции `getCode()` в строке 32 вместо строки

`return code;`

записать

`return Bar();` ИЛИ `return res;`

Тогда при любых значениях логина и пароля с одинаковой длиной получится следующий результат (рисунок 5.2-3).

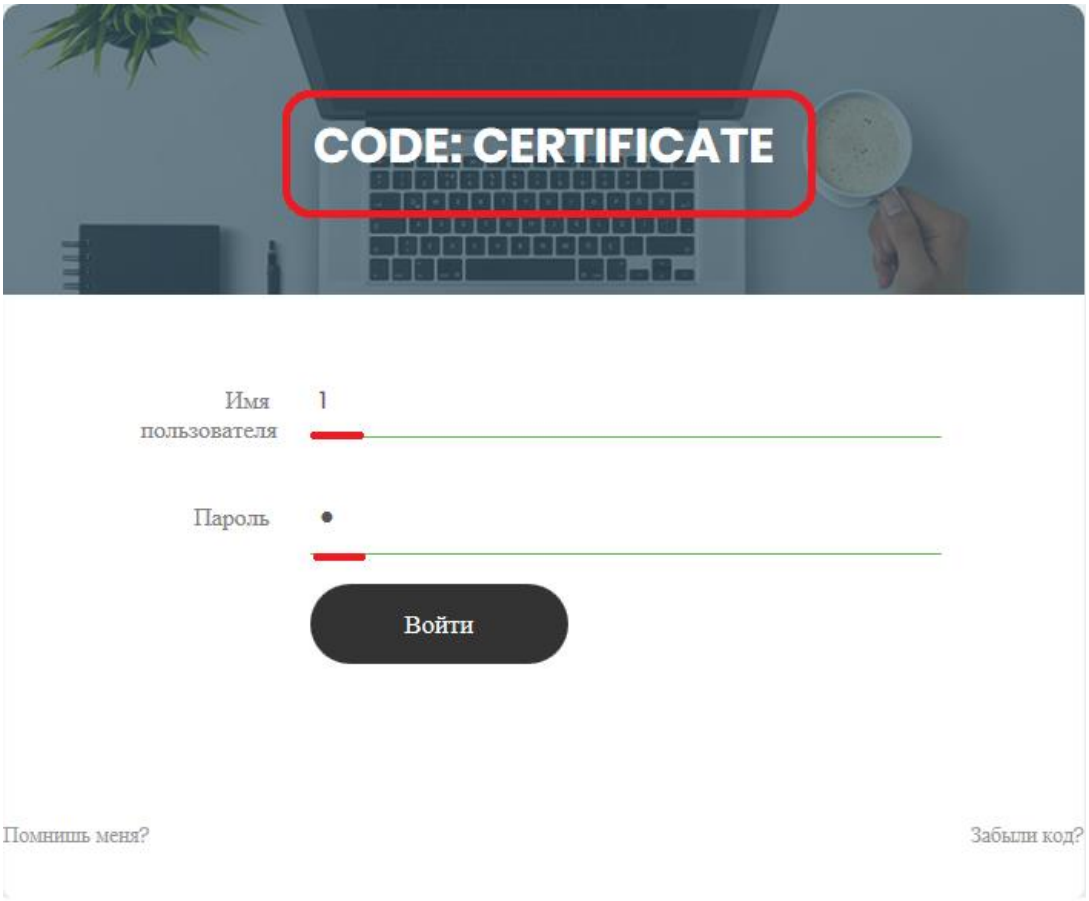

Рисунок 5.2-3 – Содержимое web-страницы с измененной функцией getCode()

Ответ: CERTIFICATE.