Задания, решения и критерии отборочного этапа олимпиады школьников Ugra CTF Quals 2021

Критерии

Формат проведения этапа

Каждый участник олимпиады получал персональный вариант каждой задачи. Варианты были сгенерированы непосредственно при получении задания. Генераторы вариантов и исходные коды задач расположены в репозитории олимпиады: https://github.com/teamteamdev/ugractf-2021-quals.

Для генерации собственного варианта запустите в директории соответствующего задания скрипт: `python3 generate.py uuid ..`

Критерии оценивания

Каждое задание оценивается в полный балл, если участник смог получить и сдать соответствующий ответ, и в ноль баллов во всех остальных случаях.

В заключительный этап были приглашены участники команд, набравших 650 и более баллов.

Задачи и решения

Антивирус

Бинарная эксплуатация, 250 баллов.

> Югорские разработчики создали новый облачный антивирус, с большой точностью определяющий вредоносные приложения. Но хорошо ли протестирован сам антивирус?
>
> https://antivirus.q.2021.ugractf.ru/token/

Решение

Таск — вариация на тему Local File Inclusion.

Заходим на сайт с антивирусом. Здесь мы видим возможность загрузить произвольный файл на проверку, а также выбор расширения файла. Загрузив произвольный файл, мы получаем отчёт.

Уязвимость заключалась в недостаточной проверке поля `ext` (расширение файла). Сделав запрос с неверным расширением, можно получить такое сообщение об ошибке:

```
Не удалось проверить файл /tmp/uploads/example.txt плагин /app/plugins/test.py не найден
```

Такие запросы можно выполнять любым удобным способом; автор пользовался функцией “Edit and Resend” в Firefox, которая позволяет отредактировать и отправить произвольный запрос из вкладки инспектора “Network”.

Таким образом мы получили сразу два пути: путь к загруженному файлу и путь к несуществующему плагину. Становится понятно, что антивирус динамически загружает модули, отвечающие за проверку файлов того или иного типа.

Осталось лишь попробовать загрузить наш собственный файл в качестве модуля. Сделав запрос с `ext=/tmp/uploads/test`, и приложив файл `test.py` с кодом `print("Hello, world!")`, получаем:

```
Hello, world! Не удалось проверить файл /tmp/uploads/test.py
```

На этом этапе мы можем выполнять произвольный код на стороне сервера. Осталось изучить файловую систему, пользуясь стандартными функциями Python. Искомый флаг находится, по традиции, в файле /etc/passwd. Получить его можно, например, однострочником на Python:

```
print(open("/etc/passwd").read())
```

Флаг: **ugra_who_checks_the_checker_49e6dac45e8f**

Арбатско-Покровская линия?

Криптография, 150 баллов.

> Вот сеанс работы в некой среде:
>
> ```
> ⎕IO ← 0
> A ← 'abcdefghijklmnopqrstuvwxy_1234567890'
> I ← (6 6 ⍴ 1 + ⍳6)+⍉(10 × 6 6 ⍴ 1 + ⍳6)
> S ← 6 6 ⍴ A
> E ← {⌈/⌈/ (~⍵ ∊ S) × I}
> E¨flag
> ```
>
> ```
> 43 21 36 11 52 42 22 15 52 32 15 46 42 52 41 42 11 42 23 33 32 52 23 41 52 15 41 33 42 15 36
> 23 13 52 34 36 33 21 36 11 31 31 23 32 21 52 13 16 65 66 54 13 54 16 53 12 53 56 54 13 15 64
> 61 55 16 65 12 61 12 14 62 62 14 14 61 66 63 14 54 55 61 53 15 63 16 61 62 15 55 16 64 62 53
> 65 61 15 63 12 65 53 65 12 55 15
> ```
>
> Мы так и не поняли, что это за среда, но очень рады результату, ведь теперь вы тоже, скорее всего, ничего не поймёте.

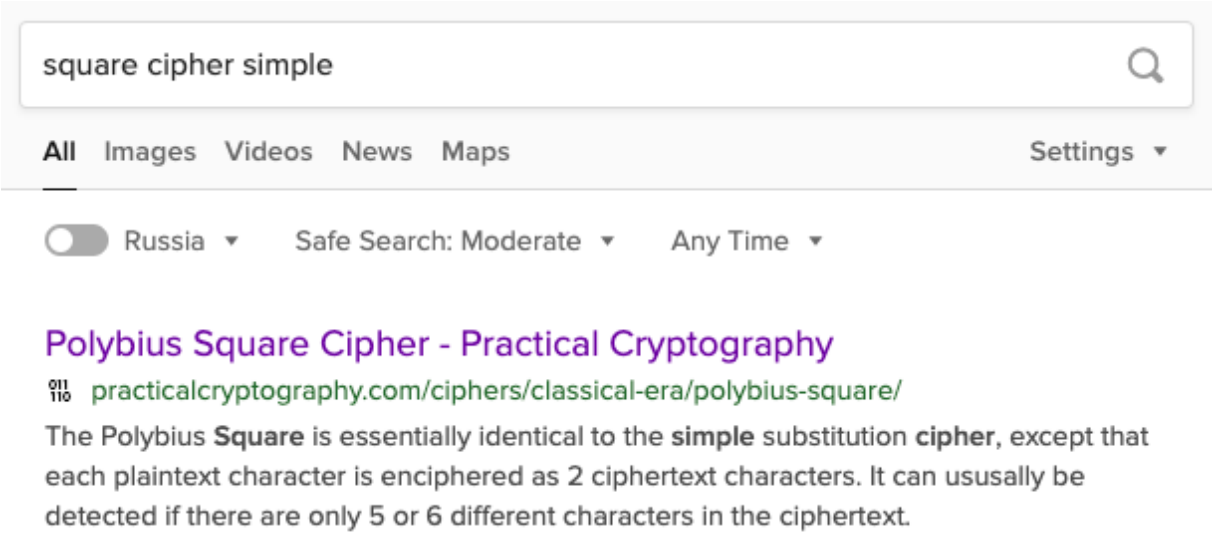
Решение

У этого таска есть два пути решения.

Путь первый. Простой Можно было угадать криптографический алгоритм по ряду намёков:

- поскольку в условии задачи дан и шифртекст, и алгоритм, которым он получен, явно применялся симметричный шифр;
- в коде часто фигурирует число 6;
- в шифртексте много чисел, но состоят они только из чисел от 1 до 6;
- в алфавите отсутствует буква z, зато его длина — ровно 36 (6^2) символов.

Ключевые слова: *квадрат, простой, шифр*:



И действительно. Это так называемый квадрат Полибия: алфавит последовательно выписывается в квадрат, а каждый символ открытого текста записывается как два числа: номер столбца и номер строки соответствующего символа в этом квадрате. В интернете есть онлайн-дешифраторы, но давайте напишем свой на Питоне. Алгоритм несложный:

```
alphabet = "abcdefghijklmnopqrstuvwxy_1234567890"
ciphertext = "43 21 [...] 15".split()


def decode_one(character):
   row, col = map(int, character)
   index = (row -1) * 6 + (col - 1) # вычитаем по единице — индексация с нуля
   return alphabet[index]


print(''.join(decode_one(x) for x in ciphertext))
```

Путь второй. Интересный Можно было ничего не угадывать, а понять, что это за среда. Тогда становится понятной цепочка преобразований открытого текста — и, следовательно сам алгоритм. Среда называется APL и представляет собой язык программирования из шестидесятых, ориентированный на работу с числами и массивами. Это определяется, во-первых, по странным значкам в коде, во-вторых, по названию таска (и, в особенности, по его URL).

> Любопытно, что этот язык появился до массового распространения компьютерных мониторов — для работы с ним люди использовали печатающие терминалы, причём той модели, которая способна воспроизводить большой набор нестандартных символов, и которая уже знакома преданным фанатам наших соревнований, — IBM Selectric.
>
> Кстати, специальные символы языка чуть ли не целиком занимают отдельный специализированный блок Юникода.

Язык обладает нехитрым синтаксисом, который лишь кажется сложным из-за закорючек, стрелочек и прочих греческих букв. Каждое выражение выполняется справа налево (скобки в приоритете). Записанные подряд числа формируют массивы: `1 2 3`. Между числом или массивом можно поставить значок — функцию. У неё может быть либо один аргумент: `(⍳10)`, либо два: `1 + 3`. Вот перечень некоторых функций, используемых в таске:

Символ	Один аргумент	Два аргумента
`⍳`	Последовательность от 0 до аргумента как `range()`	Индекс элемента (правый аргумент) в массиве (левый аргумент)
`~`	Логическое отрицание	Массив (левый) без элемента (правый)
`⍴`	Длина правого аргумента как `len()`	Изменяет размерность правого аргумента согласно левому аргументу как `np.reshape()`
`⍉`	Транспонированный аргумент как `np.ndarray.T`	—
`×`	Знак (плюс или минус) правого аргумента	Умножение
`⌈`	Округление вверх	Выбирает наибольший аргумент
`¨`	—	Применить левый аргумент к каждому элементу правого как `map()`
`/`	—	Как `functools.reduce()`

Можно совершать преобразования на листочке или в уме, а можно найти какой-нибудь интерпретатор APL. Давайте смотреть. Сперва язык получает два не особо интересных указания:

1. индексировать массивы не с единицы, а с ноля: `⎕IO ← 0`;
2. запомнить алфавит как переменную `A`.

Затем начинается интересно. Складывается матрица 6×6 (`6 6 ⍴ 1`) чисел от 1 до 6 (`1 + ⍳6`) и такая же матрица, домноженная на 10 и повёрнутая набок. Получается матрица индексов `I` для квадрата Полибия:

```
11 12 13 14 15 16
21 22 23 24 25 26
31 32 33 34 35 36
41 42 43 44 45 46
51 52 53 54 55 56
```

```
61 62 63 64 65 66
```

Затем строится сам квадрат Полибия `S` из символов алфавита `A`:

```
S ← 6 6 ⍴ A
```

Теперь определяется функция `E`, которая имеет один аргумент `⍵`. Она делает вот что (на примере буквы `u`).

1. Находит, в какой позиции находится `⍵` в квадрате `S`: `(~⍵ ⍳ S)`:

```
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
```

2. Находит произведение матрицы выше с квадратом индексов `I`, выступая в роли маски. Получается квадрат индексов, в котором все нули, кроме индекса, соответствующего `⍵`:

```
0 0  0 0 0 0
0 0  0 0 0 0
0 0  0 0 0 0
0 0 43 0 0 0
0 0  0 0 0 0
0 0  0 0 0 0
```

3. Дважды находит максимальный элемент в таком квадрате — сперва по строкам, затем по столбцам: `⌈/ ⌈/ [...]`.

Эта функция применяется к каждому символу флага: `E¨flag`. Совсем не трудно придумать обратное преобразование, и мы это успели сделать, когда пошли первым путём.

На Питоне этот код мог выглядеть бы вот так, если перевести его «дословно»:

```
import numpy as np
flag = "ugra_the_next_station_is_esoteric_programming_..."

A = "abcdefghijklmnopqrstuvwxy_1234567890"
I = np.broadcast_to(1 + np.arange(6), (6, 6)) + np.broadcast_to(10 * (1 + np.arange(6)), (6, 6)).T
S = np.reshape(list(A), (6,6))
E = lambda x: np.maximum.reduce(np.maximum.reduce((x == S) * I))

print(list(map(E, flag)))
```

Флаг: **ugra_the_next_station_is_esoteric_programming_cf902c207d2351e7f**

Морской бой

Программирование, 300 баллов.

> Мы разработали новую игру на тему бессмертной классики. Теперь вражеский корабль всего один, зато он передвигается! Сможете попасть?
>
> https://battleship.q.2021.ugractf.ru/token/

Решение

Задача на программирование и простую геометрию.

Нам предлагают поиграть в «Морской бой». На деле правила сильно отличаются — корабль у врага всего один, и ответным огнём он не отвечает. Зато активно передвигается — немного поиграв, это сразу можно заметить. При этом время и количество выстрелов ограничено. Сервер выдаёт несколько вариантов ответа, различаемых по полю `"status"` в возвращаемом JSON:

- `"game_not_started"`: необходимо начать игру;
- `"game_over"`: игра окончена (закончилось время);
- `"rockets_ran_out"`: игра окончена (не осталось снарядов);

- `"miss"`: непопадание;
- `"hit"`: корабль уничтожен.

В случае непопадания сервер возвращает ещё и расстояние до корабля. Таким образом и можно вычислить его положение. Однако сделать это вручную не удастся из-за движения судна, и придётся писать программу для решения. Местонахождение корабля мы будем вычислять с помощью *триангуляции*.

Триангуляция Истоки метода восходят к древнему Египту, и в разных модификациях он применяется для множества задач. Суть метода в нахождении местоположения объекта путём измерений расстояния до него с нескольких отдалённых точек.

Предположим, что точное положение нашей цели неизвестно, но мы можем получить расстояние до неё из произвольной точки. Измерим первое расстояние до цели — r_1 (от точки A). Нарисуем окружность с центром в A и радиусом r_1. Искомый корабль будет находиться на этой окружности.

Теперь выберем точку B произвольно так, чтобы она находилась за пределами окружности, и получим расстояние до цели r_2. Нарисуем вторую окружность. Здесь возможны три варианта:

1. Окружности пересеклись в одной точке и находятся рядом: такое может быть, если мы случайно выбрали точку B так, что цель находится на отрезке между A и B. В этом случае $AB = r_1 + r_2$;
2. Окружности пересеклись в одной точке и вторая окружность включает в себя первую. Цель находится на прямой между A и B, но не между этих двух точек. В этом случае $AB = r_2 - r_1$;
3. Окружности пересеклись в двух точках (как на рисунке): цель находится в одной из этих точек.

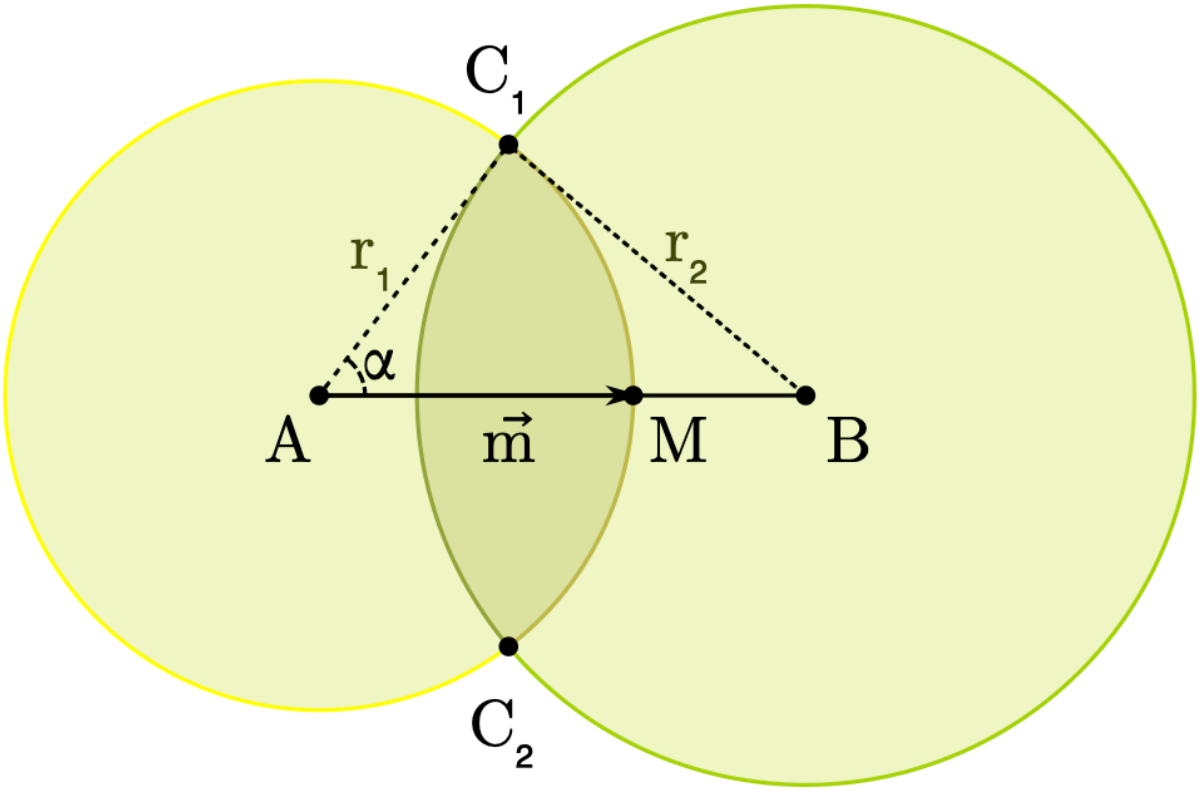


Рис. 1: Триангуляция

Для начала найдём расстояние между A и B:

$$AB = \sqrt{(x_B - x_A)^2 + (y_B - x_A)^2}$$

В первых двух случаях решение уже почти найдено. Если окружности находятся рядом, нужно найти точку их пересечения T:

$$\overrightarrow{T} = \overrightarrow{A} + (\overrightarrow{B} - \overrightarrow{A})\frac{r_1}{AB}$$

Если же окружности вложены друг в друга, можно найти точку T:

$$\overrightarrow{T} = \overrightarrow{A} - \overrightarrow{B - A}\frac{r_1}{AB}$$

Точка пересечения здесь находится на той же прямой, что и в первом случае, но с другой стороны относительно A.

Наконец, в случае (3) нам неясно, в какой из двух точек (C_1 или C_2) находится цель. Для того, чтобы найти эти точки, нужно воспользоваться тем, что все стороны треугольника ABC_1 нам известны. Воспользуемся теоремой косинусов:

$$r_2^2 = AB^2 + r_1^2 - 2r_1 AB\cos\alpha$$

$$\cos\alpha = \frac{AB^2 + r_1^2 - r_2^2}{2r_1 AB}$$

$$\sin\alpha = \sqrt{1 - \cos^2\alpha}$$

Теперь найдём вектор AM $\overrightarrow{m}$:

$$\overrightarrow{m} = (\overrightarrow{B} - \overrightarrow{A})\frac{r_1}{AB}$$

И повернём его на углы α и $-\alpha$:

$$\overrightarrow{c_1} = (\cos\alpha \cdot x_m - \sin\alpha \cdot y_m, \quad \sin\alpha \cdot x_m + \cos\alpha \cdot y_m)$$

$$\overrightarrow{c_2} = (\cos(-\alpha)\cdot x_m - \sin(-\alpha)\cdot y_m, \quad \sin(-\alpha)\cdot x_m + \cos(-\alpha)\cdot y_m) = (\cos\alpha\cdot x_m + \sin\alpha\cdot y_m, \quad -\sin\alpha\cdot x_m + \cos\alpha\cdot y_m)$$

Соответственно, точка $\overrightarrow{C_1} = \overrightarrow{A} + \overrightarrow{c_1}$; $\overrightarrow{C_2} = \overrightarrow{A} + \overrightarrow{c_2}$.

Выстрелим по C_1. Если мы попали, победа в наших руках. Если же нет, выстрелим сразу же по C_2.

Готовый код на Python, использующий библиотеки NumPy и requests:

```
import requests
import numpy as np
import sys
import math

BATTLESHIP_URL = "https://battleship.q.2021.ugractf.ru/token"

with requests.Session() as s:
  # Start the game.
  s.post(f"{BATTLESHIP_URL}/reset", cookies=jar).raise_for_status()

  def fire(p):
    """
    Fire at given NumPy point. Return distance on a miss, otherwise print result and exit.
    """

    ret = s.post(f"{BATTLESHIP_URL}/fire", cookies=jar, data={"x": p[0], "y": p[1]})
    ret.raise_for_status()
    robj = ret.json()
    if robj["status"] == "miss":
```

```
    return robj["distance"]
  else:
    print(robj)
    sys.exit(0)

A = np.array([0, 0])
r1 = fire(A)
B = np.array([2 * r1, 0])
r2 = fire(B)
AB = B[0]

if np.isclose(AB, r1 + r2):
  dist = fire(A + (B - A) * (r1 / AB))
  raise RuntimeError(f"Unexpected miss in case (1) with distance {dist}")
elif np.isclose(AB< r2 - r1):
  dist = fire(A - (B - A) * (r1 / AB))
  raise RuntimeError(f"Unexpected miss in case (2) with distance {dist}")
else:
  cos_a = (AB * AB + r1 * r1 - r2 * r2) / (2  * r1 * AB)
  sin_a = math.sqrt(1 - cos_a * cos_a)
  m = (B - A) * (r1 / AB)
  c1 = np.array([cos_a * m[0] - sin_a * m[1], sin_a * m[0] + cos_a * m[1]])
  c2 = np.array([cos_a * m[0] + sin_a * m[1], -sin_a * m[0] + cos_a * m[1]])
  fire(A + c1)
  fire(A + c2)
  raise RuntimeError("Unexpected miss in case (3)")
```

Флаг: **ugra_gotta_hit_fast_3e8ef2d0fd48**

Сертификация соответствия

Веб-технологии, 150 баллов.

> Для того, чтобы дорогой вам бизнес мог работать с новыми партнёрами, они проводят проверку на соответствие их строгим требованиям, для чего собирают важную для них информацию. Вот только вам неохота делиться всем сразу...
>
> https://compliance.q.2021.ugractf.ru/token/

Решение

Нас встречает форма сертификации соответствия ЗАО «Намбо-М» с полями, запрашивающими личную информацию, и требованиями, абсурдность которых прогрессирует по мере заполнения формы. Всё лишь для того, чтобы отсеять тех, кто недостоин сотрудничать с компанией (кстати, некоторые названия юридических лиц иногда бывает полезно прочитать задом наперёд).

По мере попыток заполнить форму выясняем, что иногда возникает ошибка 405 Not Allowed, и запрос приходится повторять.

После попытки заполнить форму в адресе страницы появляется index.php. Неплохо бы выяснить, что за версия PHP используется — возможно, это поможет нам преодолевать валидацию. И почему возникают эти странные 405, ведь мы делаем POST-запрос к форме, и большую часть времени всё работает нормально.

Откроем вкладку Network инспектора, чтобы совершаемые нами запросы записывались для подробного изучения. Видим в заголовках ответа (Response Headers) заголовок X-Powered-By, но вот что удивительно: там написано то `PHP/7.4.15`, то `PHP/8.0.2`, а у страниц ошибки 405 заголовок отсутствует вовсе.

Похоже, мы имеем дело с балансировкой нагрузки между серверами, настроенными по-разному; попадать на тот или иной сервер можно повторением запросов. Раз серверу, отдающему ошибку 405, не нравится метод POST, попробуем поделать обычные GET-запросы, пока не попадём на него. А когда всё-таки попадём, мы получим — скорее всего, в виде скачанного файла, но возможно, отображённый в виде страницы — некий файл index.php. Попавшийся сервер просто не обрабатывает PHP и отдаёт эти файлы как статические — так что теперь у нас есть исходный код этой формы.

Получение флага и даже его расчёт упрятаны в недоступный нам файл flag.py, который вызывается только на последнем шаге формы, поэтому нам придётся всё-таки понять, как её заполнить. Бросается в глаза огромная функция `validate_fields` — найдём там упоминания полей, которые не получается преодолеть сходу.

Прежде всего сталкиваемся с полем «Количество транспортных средств, находящихся в пользовании по иным основаниям» (`vehcount2`) на втором шаге.

```
if (@$form["vehcount2"] != 0) {
    $errors["vehcount2"] = "Вы не можете пользоваться иными ТС во избежание конфликта интересов.";
}
if (@$form["vehcount2"] == "0") {
    $errors["vehcount2"] = "Нельзя указывать значение 0 в данном поле!";
}
if (empty(@$form["vehcount2"])) {
    $errors["vehcount2"] = "Поле не может быть пустым!";
}
```

Требования кажутся взаимоисключающими: ноль нельзя, не ноль нельзя, и зачем-то в явном виде пустое значение тоже нельзя. Однако сравнения с нулём различаются: в первом случае строка из формы сравнивается с числом `0`, во втором — со строкой `"0"`. Итак, нам нужна строка, при сравнении с числом дающая true, а при сравнении со строкой — false. Смотрим таблицу сравнений в документации и обнаруживаем, что произвольная нечисловая строка вроде `"php"` обладает нужными свойствами. Вводим её в поле.

На третьем шаге нас поджидают поля «Количество детей» (`children`) и «Возраст детей» (`cage`).

```
if (@$form["children"] <= 1) {
    $errors["children"] = "Вы <u>ДОЛЖНЫ</u> обладать хотя бы двумя детьми для работы с нашей компанией!!";
}
if (@$form["children"] >= 2) {
    $errors["children"] = "Форма не поддерживает указание более чем одного ребёнка. Укажите одного ребёнка.";
}
```

Требуют указать число, большее 1, но меньшее 2. Для целых неотрицательных чисел, таких как количество детей, это взаимоисключающие требования, но тип данных никто не проверяет, а значит, можно указать, например, `1.5`.

```
if (@$form["cage"] == 0) {
    $errors["cage"] = "Укажите возраст Вашего ребёнка.";
}
if (@$form["cage"] == "") {
    $errors["cage"] = "Поле не может быть пустым!";
}
if ((int)(@$form["cage"]) != "11") {
    $errors["cage"] = "Указанное значение не отражает возраст Ваших детей.";
}
if (@$form["cage"] == 11) {
    $errors["cage"] = "Вы не можете указывать какое попало значение в это поле. Отнеситесь к заполнению формы серьёзно.";
}
```

11 — это «какое попало» значение, а не 11 указывать нельзя, потому что это число, по мнению формы, не отражает возраст детей. Опять видим странности в сравнении: на этот раз явно полученное из значения число сравнивается со строкой, а строка — наоборот, с числом. Смотрим в документации про числовые строки, как работает приведение типов. Узнаём про понятие *префиксной* числовой строки (вроде `"11abcd"`), которое рассматривается или не рассматривается как число, в зависимости от контекста.

С одной стороны, числовая строка должна обязательно соответствовать числу 11, чтобы пройти третье условие, строка `"11abcd"` здесь подходит. С другой стороны, сравнение строки с числом 11 должно быть ложным, и вот тут поведение PHP 7 и PHP 8 различается: PHP 7 приводит префиксную числовую строку к числу, так что проскочить условие никак не получается. А вот PHP 8 префиксную числовую строку в этом контексте за число не считает, сравнение даёт результат false, и форма проходит валидацию.

Повторяем запрос несколько раз, чтобы он точно был обработан версией 8; можно это сделать, несколько раз нажав на кнопку, потому что в версии 7 валидация не проходит.

На четвёртом шаге, к нашему удивлению, не встречаем никаких препятствий, а на пятом и вовсе обнаруживаем «код подтверждения», то есть флаг.

Флаг: **ugra_said_secret_service_secretly_succumbs_c9061152e8a7**

Конгресс

Программирование, 200 баллов.

> В 2022 году в России пройдёт Международный конгресс математиков — крупнейшая конференция в области фундаментальной и прикладной математики.
>
> Участникам конгресса уже доступны материалы докладов. Мы оформили вам персональное приглашение, для участия нужно всего лишь зарегистрироваться...
>
> https://congress.q.2021.ugractf.ru/?ref=token

Решение

Переходя по ссылке, мы попадали на сайт Международного математического конгресса — 2022. Выглядел он не очень-то и современно, но это не так важно. Понимаем, что наша цель — во что бы то ни стало попасть на этот Конгресс, и идём регистрироваться.

Заполняем фамилию, имя, логин, а с пин-кодом проблемы: нам сообщают, что он слишком популярен.

Первое желание, возникающее при виде такого задания — просто перебрать все четырёхзначные числа. Однако, уже после пятого запроса сайт отправляет нас подождать некоторое время. Таймаут в несколько минут заставляет нас отказаться от этой идеи.

Что ж, пойдём искать эти самые последовательности. Для большей их части мы наткнёмся на сайт OEIS, который гордо именует себя «Онлайн-энциклопедией числовых последовательностей»[1].

Обнаруживаем, что нередко ответы из OEIS по тому или иному числу полностью дублируют выдачу нашего сайта. Очевидно, что организаторы Конгресса вряд ли стали бы придумывать сотни и тысячи числовых последовательностей лишь с целью усложнить математикам жизнь: у нас возникают подозрения, что сайт конгресса как раз и ищет пин-коды в OEIS.

В документации узнаём, что искать последовательности, содержащие нужное число, нужно с помощью специального оператора: `seq:1234`.

Далее есть несколько путей.

Перебор Мы можем просто для каждого числа от 1000 до 9999 сделать запрос в OEIS и распарсить ответ, чтобы узнать количество совпадений. Можно даже не парсить: OEIS поддерживает `&fmt=text` для получения текстового ответа и `&fmt=json` для получения ответа в формате JSON.

Однако, если перестараться — то OEIS нас заблокирует за слишком быстрый скрейпинг. Так произошло, в том числе, с эксплойтом от автора.

База данных На самом деле, у OEIS есть встроенная Вики, на которой расположена различная документация. В частности, там есть ссылки на дампы всей базы, обновляемые ежедневно. Нам нужен первый — sequences and their A-numbers. Ищем в последовательностях каждое число и находим единственное, которое есть всего в пяти последовательностях. Это — 9935.

Мы не писали отдельный скрипт для решения, но вы можете посмотреть, как наполнялась база, из которой и выдавал последовательности наш сервер.

Бонус: OSINT Помимо всего этого, на той же Вики нашлась статья про частоту встречаемости чисел в базе OEIS. Она уже потеряла свою актуальность — исследование было проведено 13 лет назад, в 2008 году, а база пополняется новыми последовательностями чуть ли не каждую неделю.

[1] В оригинале «integer sequences» — «последовательностей из натуральных чисел»

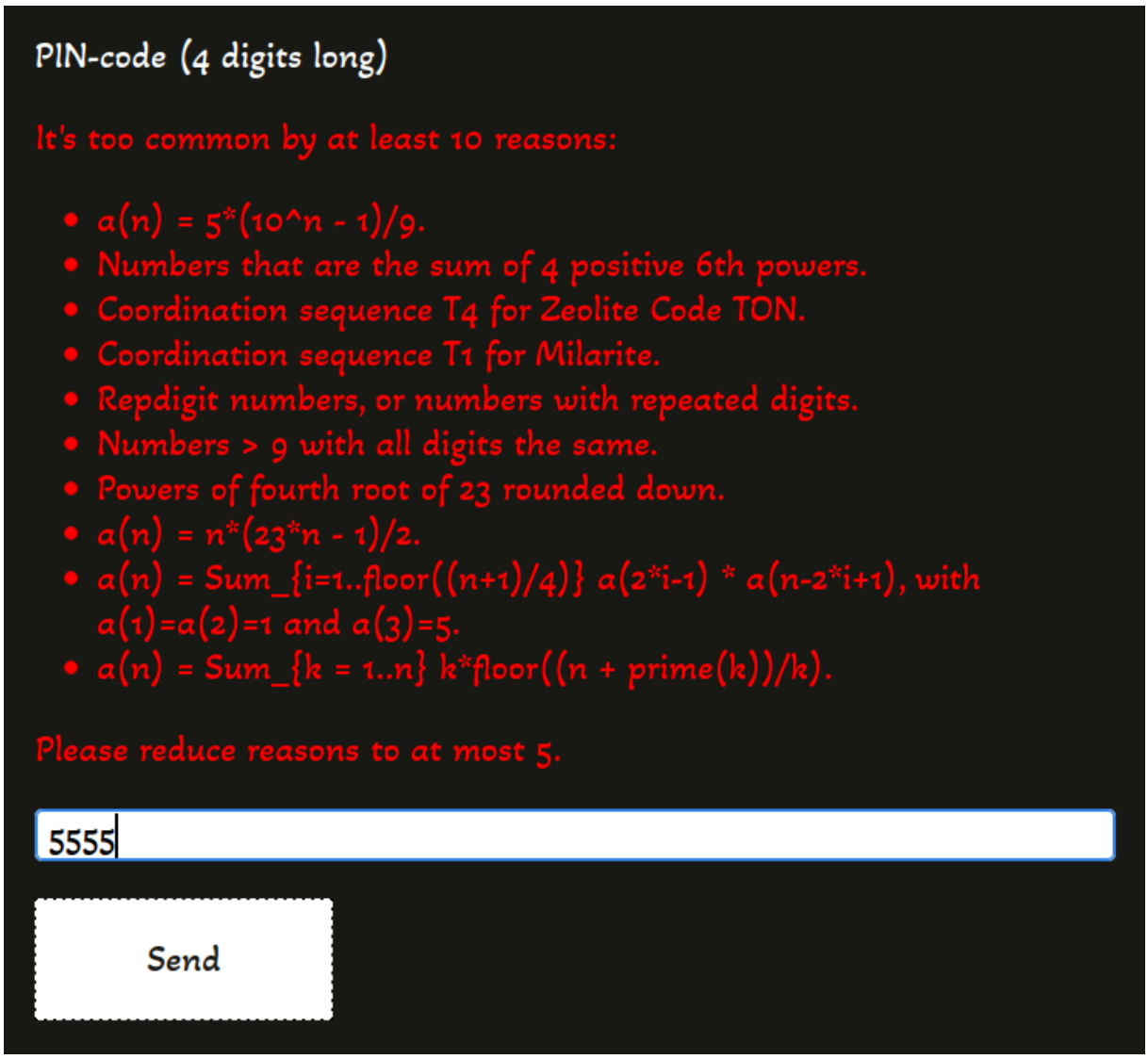


Рис. 2: Популярный PIN-код

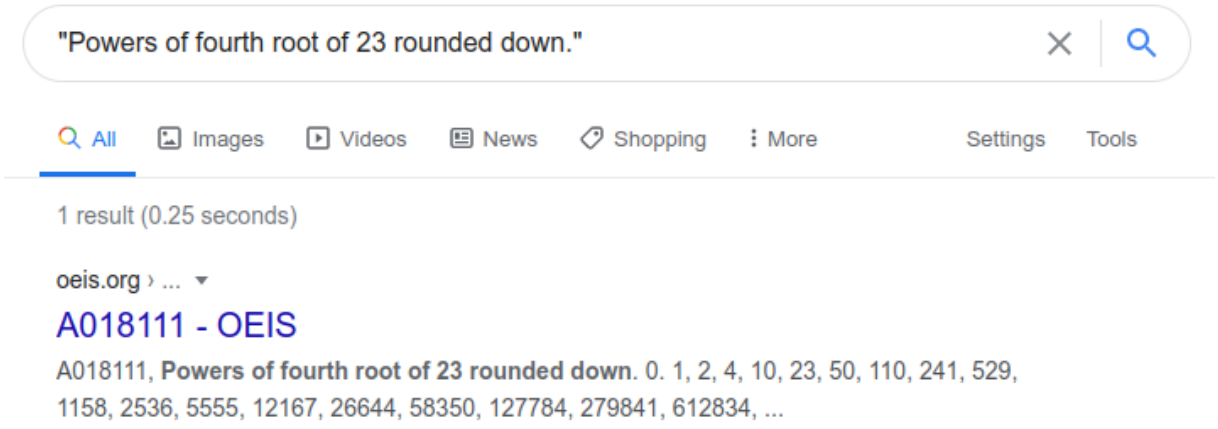


Рис. 3: Пример OEIS

Тем не менее, за это время «интересность» чисел не сильно изменилась, и одно из них — 9935 — всё ещё остаётся самым редко встречаемым среди первых десяти тысяч чисел.

> Среди пятизначных чисел уже довольно много таких (чуть более 10 000), что они встречаются в OEIS всего один раз.

Флаг: **ugra_what_a_beautiful_number_85829a3e3026**

Книга пути и достоинства

Разное, 150 баллов.

> Китайская мудрость утверждает, что, чтобы не сбиться с трудного пути, необходимо употреблять достаточное количество одного витамина. Не собьётесь?
>
> *daodejing.txt*

Решение

Книга пути и достоинства — по-китайски Дао Дэ цзин (Дао — путь, Дэ — достоинство). В задании упомянуты витамины, один из них — витамин D.

Всё это должно было намекнуть на сущность содержимого файла: это данные **пути** в формате SVG-атрибута **d**.

Создадим минимальный документ с одним объектом *path*: можно открыть графический редактор и нарисовать в нём какой-нибудь путь, а можно просто взять образец из спецификации и убрать необязательные атрибуты. У объекта *path* укажем *d* из файла:

```
<svg xmlns="http://www.w3.org/2000/svg">
    <path d="m 176.06939,6.1349363 c 0.59631,0 0.98918,-0.3648043 1.05934,-0.8418562 h -0.41392 ..." />
</svg>
```

Откроем этот файл в редакторе или браузере, перепишем флаг.

Флаг: **ugra_grandpa_take_your_pills_or_we_will_beat_up_your_bfc187edca9a70c8**

Новейшая разработка

Форензика, 200 баллов.

> Это исходный код программного обеспечения, разработанного очень крутыми разработчиками. Но иногда и они допускают ошибки. Ну так, по невнимательности.
>
> *kafkacat.zip*

Решение

Архив содержит исходный код программы kafkacat и директорию .git со всей историей разработки. Её изучение не даёт ничего интересного — она в точности совпадает с публичной историей на Гитхабе. Проверим, нет ли в репозитории каких-нибудь неучтённых объектов, выполнив команду `git fsck`:

```
$ git fsck
Checking object directories: 100% (256/256), done.
Checking objects: 100% (1347/1347), done.
dangling commit 4693d471cb035704931d4d28f70b999dd158f4fb
```

Посмотрим содержимое коммита:

```
$ git show 4693d471cb035704931d4d28f70b999dd158f4fb
commit 4693d471cb035704931d4d28f70b999dd158f4fb
Author: Validian <validian@validian.name>
Date:   Wed Dec 11 06:00:30 2019 +0845

    added info.txt.

diff --git a/info.txt b/info.txt
new file mode 100644
```

```
index 0000000..0a70bb9
--- /dev/null
+++ b/info.txt
@@ -0,0 +1,7 @@
+If you are reading this, I am definitely dead by now.
+
+I also know that by reaching this file, you have demonstrated
+the best of your ability and courage. Here is the key to my
+lifetime secret: ugra_the_yellow_purse_bfc187edca9a70c8
+
+You will understand what to do next. Good luck!
```

Взгрустнув о тяжёлой судьбе никогда не существовавшего разработчика, сдаём флаг.

Флаг: **ugra_the_yellow_purse_bfc187edca9a70c8**

Команда мечты

Сетевая разведка, 100 баллов.

> За их игрой так интересно наблюдать, что можно потерять всякий контроль над временем. А его терять никак нельзя. Придётся вспомнить, во сколько был сделан вот этот снимок.
>
> *Примечание 1.* У вас есть восемь попыток. Проверяющая система беспощадна и принимает лишь точный ответ — прочитайте правила ввода и не тратьте попытки зря.
>
> *Примечание 2.* Проверяющая система не содержит известных авторам уязвимостей. Настойчивые попытки получить флаг способом, отличным от ввода верного ответа в рамках предоставленного количества попыток, считаются атаками на инфраструктуру с соответствующими последствиями для команды.
>
> gGeQ3Et32Jw.jpg
> https://dreamteam.q.2021.ugractf.ru/token/

Решение

В задании видим фотографию с футбольного матча. Рефлекторно отправляем её в поиск по картинкам Гугла. Карточка говорит про вратаря Акинфеева, а первый результат, в котором встречается изображение — подборка твитов про матч Россия — Испания, случившийся 1 июля 2018. Немного пролистав вниз, обнаружим твит, содержащий то самое фото.

Перейдя к поиску такого же изображения в другом размере, находим более крупные версии, например, вот эту. У этого файла есть EXIF-метаданные; можно просмотреть их командой `exiv2 -pa GettyImages-988874128.jpg`. Согласно ним, фотограф использовал камеру Canon EOS 1D X Mark II; указано и время съёмки — 2018:07:01 17:44:38. Несмотря на то, что 1D X Mark II оборудована встроенным GPS-приёмником и теоретически может сверять по нему время, эта функциональность может быть отключена (например, по умолчанию) или не обеспечивать достаточной точности синхронизации. На отключенность GPS на камере намекает и отсутствие в метаданных координат съёмки (при том, что остальные метаданные, включая серийный номер камеры, сохранились, то есть вряд ли оттуда что-то удалялось специально). Поэтому не будем доверять времени из метаданных и перепроверим его по другим источникам.

Так или иначе, мы с уверенностью знаем матч. Он удостоен даже отдельной статьи в Википедии. Там перечислены составы. Выясняем, что игрок сборной Испании на переднем плане — Диего Коста (на его красной форме с трудом, но можно узнать номер 19; игроков номер 14 или, скажем, 74 — как иначе можно было бы прочитать этот номер — на поле в тот день не было). В синей форме под номером 1 — вратарь Акинфеев.

Откроем одну из текстовых трансляций по ссылкам из конца статьи Википедии и найдём там упоминания взаимодействий Косты и Акинфеева. Что-то похожее на происходящее на картинке *(Коста под углом выскочил один на один, но Акинфеев вышел из ворот)* произошло в первую дополнительную минуту первого тайма. Поищем эти же две фамилии в Твиттере и найдём твит об этом же моменте, отправленный в 17:46:32. (Секунды твита можно посмотреть в инспекторе, найдя во вкладке Network запрос к адресу, совпадающему с идентификатором твита, и прочитав JSON-ответ.)

Можем уточнить игровое время по какой-нибудь нарезке моментов — 45:10 или 45:11. Осталось поточнее сопоставить игровое время с астрономическим. В том же видео видим, что в 40:54 игрового времени забили второй и последний в основном игровом времени гол, и сделал это Дзюба. Понятно, что первые твиты про это появятся спустя несколько секунд. Ищем все твиты про Дзюбу и находим чёткий момент, где нервное напряжение резко сменяется ликованием. Первый твит про гол написан в 17:41:05 — сам гол явно случился в пределах нескольких секунд. Следовательно, 45:10 игрового времени случилось явно не раньше 17:45 астрономического.

Получается, время момента — 17:45 или, возможно, 17:46; вполне можно в пределах восьми попыток перебрать оба варианта (на самом деле, принимался любой из них). Часовой пояс Москвы — +03:00. Аккуратно запишем ответ в нужном формате: `2018-07-01T17:46+03:00`, отправим — и получим флаг.

Флаг: **ugra_from_friendship_in_sports_to_the_world_on_the_land_207fc9373540**

Hantaton Live

Стеганография, 350 баллов.

> Читателям нашего курса очень понравилась музыка, которая его сопровождала. Так уж и быть, мы опубликовали секретную разработку, которая позволила нам её написать. Только никому! Мы проверим.
>
> https://hantatonlive.q.2021.ugractf.ru/token

Решение

Нам предлагается *программный секвенсер*, работающий прямо в браузере. Можно сочинить свою мелодию, используя инструменты, которыми сыграна завершающая мелодия из титров последнего видеоразбора нашего курса. Правда, нельзя задать ни темп мелодии, ни даже её продолжительность, можно только включать и выключать различные звуки.

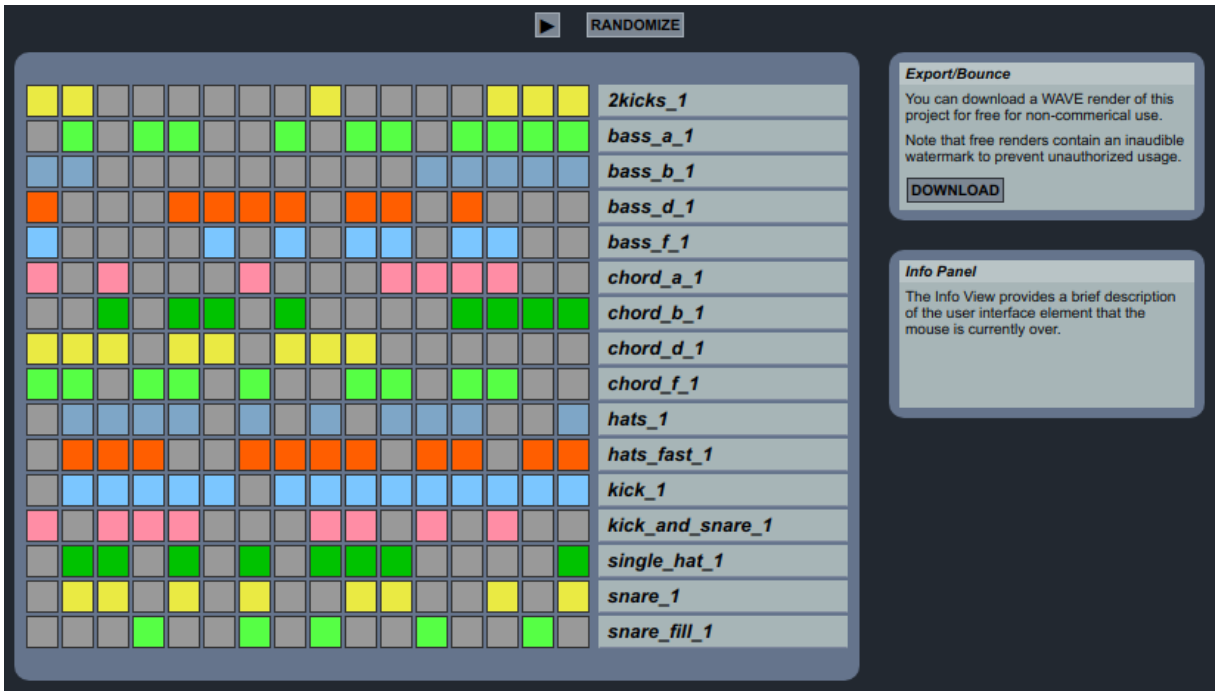


Рис. 4: Интерфейс

Из этого следует, что загружаемые файлы всегда имеют одинаковую длину. Нас предупреждают: *free renders contain an inaudible watermark*. Но то, что нельзя услышать на фоне звуков, может быть различимо на фоне тишины. По крайней мере, программными средствами.

При генерации звукового файла Hantaton Live пропускает такты, где не было выбрано ни одного звука. Поэтому выберем 16 раз самый короткий звук, чтобы между его повторениями была самая долгая доступная для анализа тишина. Это звук kick_1.

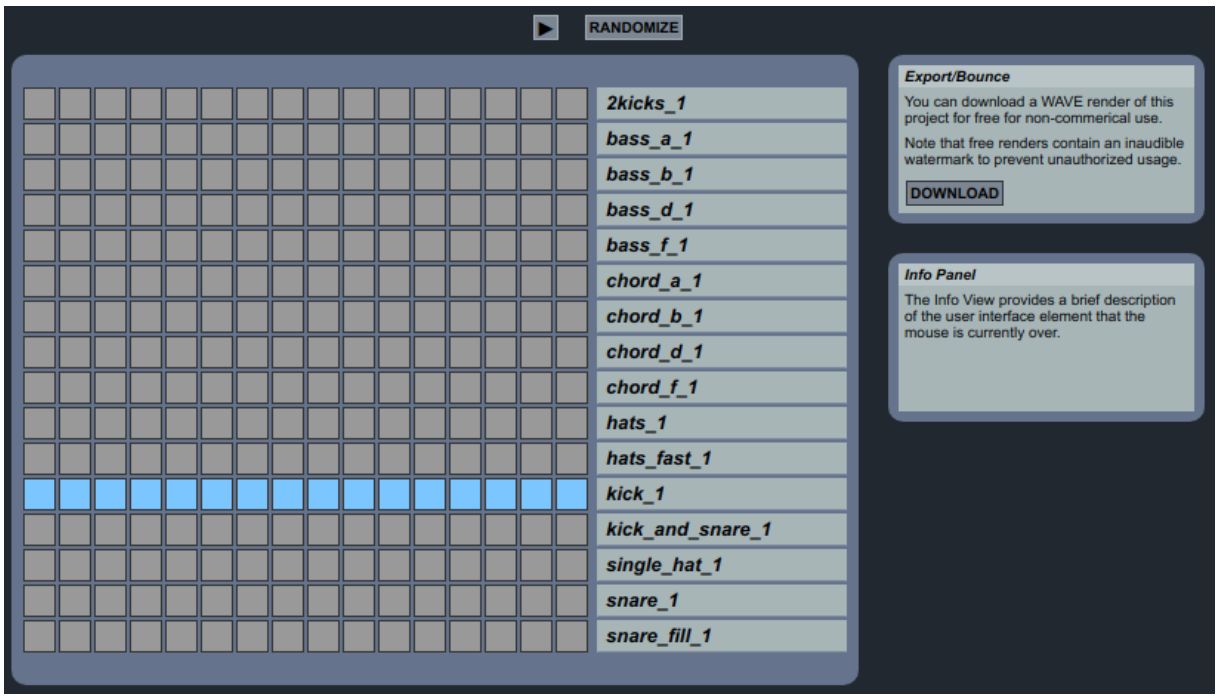


Рис. 5: Выбран нужный звук

Скачаем файл — это файл формата .wav, то есть в нём после заголовка просто записаны подряд байты, обозначающие амплитуду. Таким образом, абсолютной тишине между звуками должны соответствовать нулевые байты.

Попробуем найти области тишины (или почти тишины). Прочитаем файл в виде битового массива.

```
import numpy as np
file_bytes = np.array(list(open('bounce.wav', 'rb').read()), dtype=np.uint8)
bits = np.unpackbits(file_bytes)
```

Разобьём весь файл на сегменты, например, по 1000 бит и просуммируем биты в каждом таком сегменте. Поскольку тишина кодируется нулями, а у нас есть длительные периоды тишины, мы должны увидеть большое количество сегментов, все биты в которых — нули (а значит, и сумма нулевая). А может быть, в некоторых из этих сегментов есть что-нибудь интересное, и тогда мы увидим не ноль, но какое-нибудь маленькое число (потому что тишина всё равно должна звучать как тишина, и сегмент в любом случае должен содержать гораздо больше нулей, чем единиц).

```
bit_sums = sum(bits[i : i + 1000]) for i in range(0, len(bits), 1000)]
```

Выведем массив и посмотрим издалека.

Действительно — мы видим регулярные крупные группы нулевых сумм, среди которых есть редкие суммы, равные 1 (то есть на 1000 идущих подряд бит пришлось 999 нулей и одна единица). Может, как раз эти одиночные единицы и являются частью водяного знака? Вычислим позиции таких единиц.

```
single_bit_thousands = np.where(np.array(bit_sums) == 1)[0] * 1000
positions = []
for start_pos in single_bit_thousands:
    positions.append(start_pos + np.where(bits[start_pos : start_pos + 1000] == 1)[0][0])
positions = np.array(positions)
```

Получается набор чисел.

```
array([  353442,   392674,   549602,   588834,   667298,   706530,
         745762,   863458,  1295010,  1412706,  1491170,  1530402,
        1608866,  1648098,  1765794,  1844258,  2236578,  2275810,
        2471970,  2550434,  2589666,  2628898,  2746594,  3178146,
        3217378,  3492002,  3531234,  3609698,  4119714,  4158946,
        4237410,  4315874,  4433570,  4472802,  4590498,  4629730,
```

Рис. 6: Битовые суммы групп по тысяче

```
       4668962,  5061282,  5100514,  5178978,  5296674,  5375138,
       5414370,  5453602,  5571298,  5610530,  6002850,  6238242,
       6316706,  6395170,  6434402,  6473634,  6512866,  6552098,
       6944418,  7179810,  7258274,  7297506,  7375970,  7415202,
       7493666,  7885986,  8003682,  8199842,  8278306,  8317538,
       8356770,  8396002,  8435234,  8827554,  9023714,  9141410,
       9180642,  9298338,  9376802,  9769122,  9886818,  9965282,
      10004514, 10122210, 10161442, 10239906, 10279138, 10318370,
      10710690, 10946082, 11063778, 11103010, 11181474, 11220706,
      11259938, 11887650, 12005346, 12044578, 12123042, 12162274,
      12201506, 12789986, 12907682, 12946914, 13064610, 13143074,
      13653090, 13770786, 13888482, 13927714, 14045410, 14084642,
      14673122, 14790818, 14830050, 14947746, 14986978])
```

Внимательно посмотрев на него, можем заметить, что некоторые числа как будто бы отстоят друг от друга на одинаковом расстоянии. Проверим эту догадку, вычтя массив из самого себя со сдвигом на единицу:

```
position_deltas = positions[1:] - positions[:-1]
```

```
array([ 39232, 156928,  39232,  78464,  39232,  39232, 117696, 431552,
       117696,  78464,  39232,  78464,  39232, 117696,  78464, 392320,
        39232, 196160,  78464,  39232,  39232, 117696, 431552,  39232,
       274624,  39232,  78464, 510016,  39232,  78464,  78464, 117696,
        39232, 117696,  39232,  39232, 392320,  39232,  78464, 117696,
        78464,  39232,  39232, 117696,  39232, 392320, 235392,  78464,
        78464,  39232,  39232,  39232,  39232, 392320, 235392,  78464,
        39232,  78464,  39232,  78464, 392320, 117696, 196160,  78464,
        39232,  39232,  39232,  39232, 392320, 196160, 117696,  39232,
       117696,  78464, 392320, 117696,  78464,  39232, 117696,  39232,
        78464,  39232,  39232, 392320, 235392, 117696,  39232,  78464,
        39232,  39232, 627712, 117696,  39232,  78464,  39232,  39232,
       588480, 117696,  39232, 117696,  78464, 510016, 117696, 117696,
        39232, 117696,  39232, 588480, 117696,  39232, 117696,  39232])
```

Видим много чисел 39232, но среди них есть и другие. Убедимся, что все они делятся на 39232 нацело:

```
position_deltas_mod = (positions[1:] - positions[:-1]) % 39232
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0])
```

Получился массив нулей, а значит, единицы действительно отстоят друг от друга на расстояние, кратное 39232. Можно предположить, что биты водяного знака распределены по файлу равномерно, и в тех местах, где между единицами расстояние больше, на самом деле находятся нули водяного знака, неотличимые в данном файле от нулей тишины.

В таком случае самый первый бит водяного знака должен находиться на позиции 353442 % 39232, то есть 354. Попробуем взять биты, начиная с позиции 354, с шагом 39232, и упаковать их в байты.

```
watermark_bytes = np.packbits(bits[354::39232])
```

```
array([117, 103, 114,  97,  95, 101, 118, 101, 114, 121, 116, 104, 105,
       110, 103,  95, 105, 115,  95,  97,  95, 114, 101, 109, 105, 120,
        95, 101, 118, 101, 110,  95,  55, 100,  97,  55,  52,  49,  55,
       102,  54, 101,  50,  57,  51, 100,  54, 102], dtype=uint8)
```

Получился массив из чисел, похожих на коды символов. Осталось сделать из них собственно символы и обрадоваться тому, что получился флаг.

```
flag = bytes(watermark_bytes)
```

Флаг: **ugra_everything_is_a_remix_even_7da7417f6e293d6f**

Набор отрезков

Разное, 200 баллов.

> Линейные функции f_1–f_{47} определены каждая на своём отрезке множества действительных чисел. Для каждой функции известны координаты двух точек, через которые проходит график функции — они и являются границами области определения и области значений.

Функция	Левая точка	Правая точка
f_1	(−225, 80)	(45, 100)
…	…	…
f_{47}	(−405, 100)	(0, 70)

Решение

Можно заметить, что первая координата и левых, и правых точек почти везде делится на 45, а там, где не делится — делится на 30. Этот факт, а также разброс значений, намекает на возможность рассмотреть первую координату — аргумент функции — как угол в градусах.

Если в обычной декартовой системе координат (x, y) ограниченные линейные функции изображаются прямыми отрезками, то в полярной (ϕ, r) они могут быть частями спиралей или окружностей.

Например, построим график функции f_1. Выпишем выражение для линейной функции, помня, что углы следует записывать в радианах вместо градусов.

Делать это для всех 47 графиков было бы не очень удобно, поэтому можем открыть Jupyter Notebook и написать код, который строит график по двум заданным точкам.

```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np

def draw_polar(phi1, r1, phi2, r2):
    phi = list(range(phi1, phi2 + 1))
```

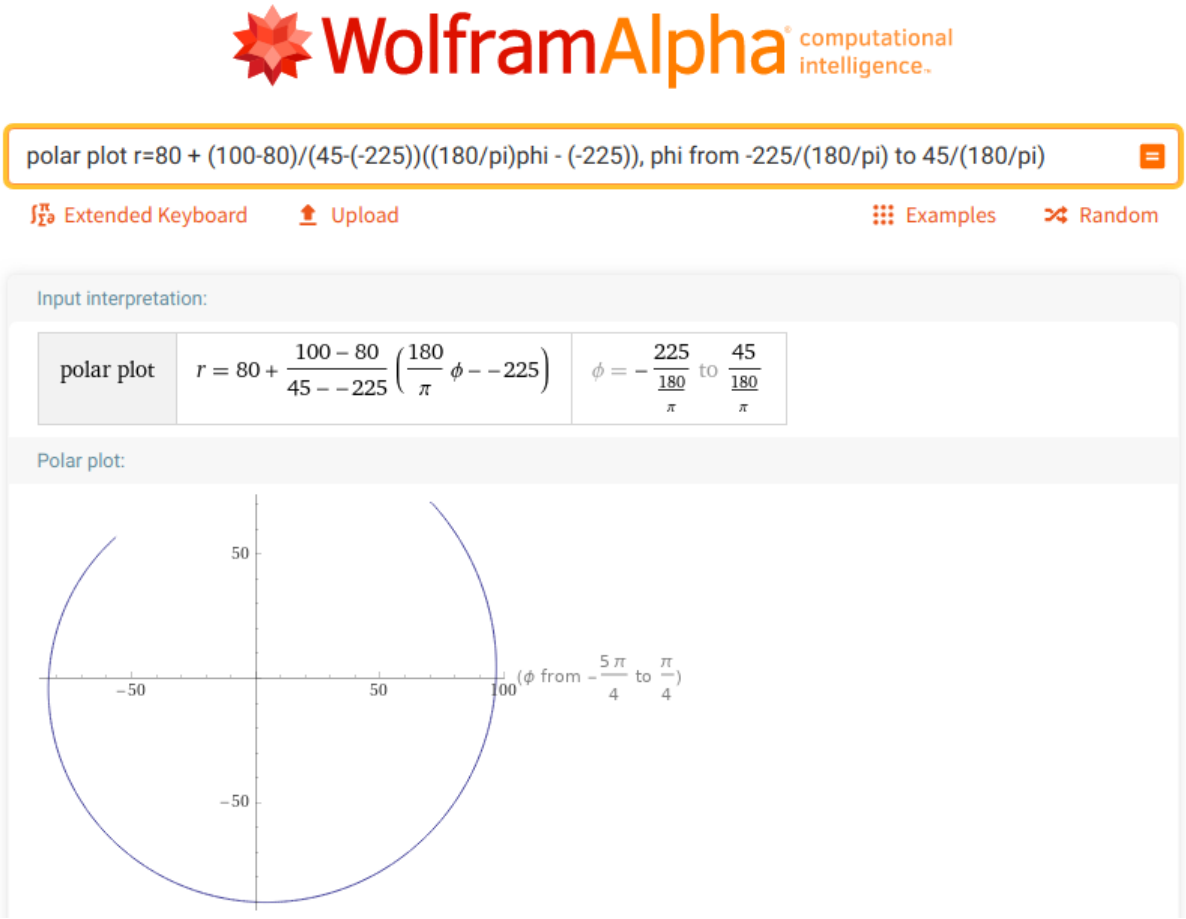


Рис. 7: График первой функции

```
    r = [r1 + (r2-r1)*(p - phi1)/(phi2-phi1) for p in phi]

    plt.polar(np.array(phi) / (180 / np.pi), np.array(r), linewidth=15)
```

И построить все графики.

```
for (phi1, r1), (phi2, r2) in [((-225, 80), (45, 100)), ((-495, 100), (-45, 20)), ...]:
    draw_polar(phi1, r1, phi2, r2)
    plt.show()
```

Посмотрим на графики чуть издалека.

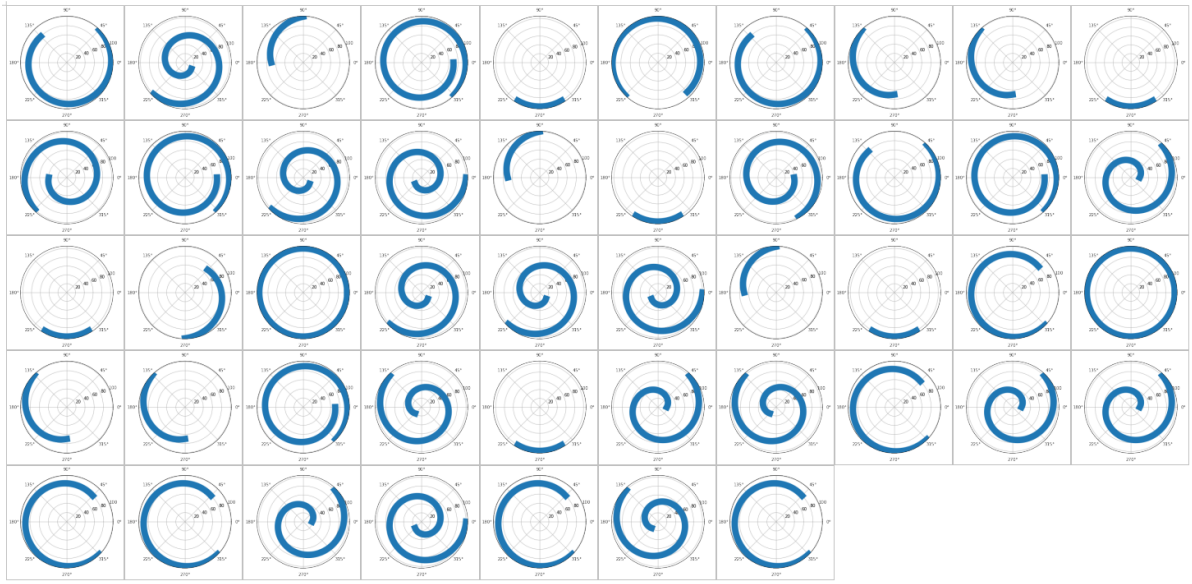


Рис. 8: Все графики

Бросается в глаза, что первые графики образуют что-то, подозрительно похожее на слово *ugra*. Дальше идёт что-то, явно пытающееся быть символом подчёркивания. Остальные графики тоже образуют в целом узнаваемые английские слова — если какие-то буквы не удаётся опознать даже после нескольких минут пристального взгляда и анализа методом исключения, можно разрешить сомнения с помощью поиска по словарю. Буквы последней части флага — в этом задании это могут быть только a, b, c, d и e — по крайней мере по разу встречались в предыдущих словах, поэтому их можно восстановить, пользуясь предположением, что одинаковые буквы закодированы одинаковыми функциями.

Флаг: **ugra_null_pager_quad_jogger_collab_dbcddccdecbc**

noteasy+82

Криптография, 250 баллов.

> С алфавитами не сложилось.
>
> *noteasy82.txt*

Решение

Задание продолжает традиционную для соревнований Ugra CTF серию *noteasy*, где флаг кодируется неким алфавитным шифром, а кроме шифртекста, не известно практически ничего.

+82 — телефонный код Южной Кореи. У корейцев с алфавитом действительно немного не сложилось: система письменности *хангыль* действительно стоит всего лишь из 24 графем, но правила композиции их в слоговые блоки таковы, что в Юникоде пришлось породить тысячи возможных комбинаций. Но мы не об этом.

Существует несколько способов передачи корейского языка латиницей, в Южной Корее используют новую романизацию. Карточка в правой части статьи Википедии рассказывает и о других способах,

в частности, SKATS. SKATS придумывался для однозначной передачи букв при передаче информации азбуке Морзе, причём латинские буквы были выбраны достаточно произвольно, без оглядки на звучание соответствующих базовых символов хангыля.

Все буквенные символы флага имеются в SKATS. Составим таблицу соответствия между символами флага из SKATS, символами хангыля и транслитерацией символов хангыля новой романизацией, при неоднозначности выбирая тот вариант романизации, при котором кодирование остаётся взаимно однозначным. Символ ㅇ используется в слоговых блоках как индикатор отсутствующего звука, поэтому ему сопоставляем встречающийся во флаге символ _.

SKATS	Хангыль	Латиница
A	ㅗ	o
B	ㄷ	d
E	ㅏ	a
F	ㄴ	n
G	ㅅ	s
H	ㅜ	u
J	ㅎ	h
K	ㅇ	_
L	ㄱ	g
M	ㅁ	m
O	ㅍ	p
P	ㅈ	j
U	ㅣ	i
V	ㄹ	r
W	ㅂ	b
X	ㅋ	k
Z	ㅌ	t

Декодируем флаг. Цифры оставляем без изменений.

H	L	V	E	K	P	H	G	Z	K	J	A	F	X	K	E	F	B	K	W	U	W	U	M
u	g	r	a	_	j	u	s	t	_	h	o	n	k	_	a	n	d	_	b	i	b	i	m

W	E	O	K	4	8	8	0	0	E	3	9	6	8	2	W	B	7	W	4
b	a	p	_	4	8	8	0	0	a	3	9	6	8	2	b	d	7	b	4

Флаг: **ugra_just_honk_and_bibimbap_48800a39682bd7b4**

Nothing to see

Веб-технологии, 100 баллов.

> Нечего там смотреть.
>
> https://token.nothing.q.2021.ugractf.ru

Решение

При попытке перейти по ссылке из условия, понимаем, что у сайта проблемы с сертификатом. В лучшем случае браузер поругается, но всё же даст возможность войти (например, Firefox), в худшем он вам вовсе запретит посещать данную страницу. Если взять во внимание тот факт, что у других тасков таких проблем нет, можно предположить, что решение этого таска должно быть связано с сертификатом.

Давайте попробуем открыть сертификат. В Firefox нужно нажать на «замочек» слева от адресной строки, нажать на стрелочку, затем **More Information**, и, наконец, **View Certificate**. В Chrome

нужно нажать на **Not secure** и выбрать **Certificate**. Видим, что на первый взгляд ничего флагоподобного в сертификате нет, или, по крайней мере, наш браузер нам этого не показывает, поэтому попытаемся посмотреть сертификат самостоятельно.

Скачаем сертификат. В Firefox в разделе **Miscellaneous** выберите **Download PEM (cert)**, в Chrome — **Details** → **Export**.

Нам поможет набор утилит `openssl` — она умеет в том числе и смотреть внутренности сертификата:

```
$ openssl x509 -in cert.pem -text -noout
```

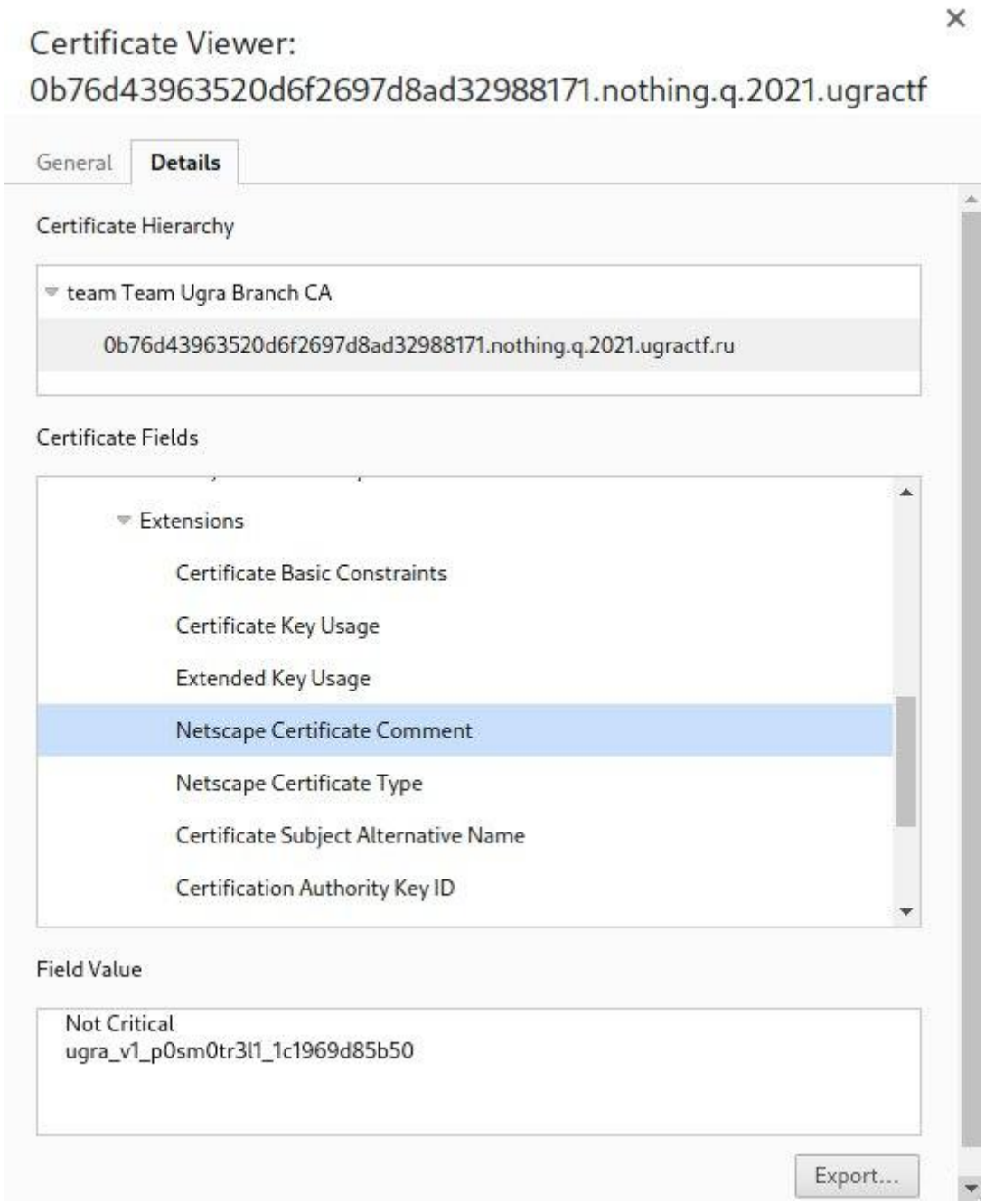
Смотрим на вывод и видим флаг:

```
...
Netscape Comment:
                ugra_v1_pOsmOtr3l1_1c1969d85b50
...
```

Помимо этого, `openssl` может скачать сертификат сам, и всё можно сделать в одну строку:

```
$ echo | openssl s_client -connect 0b76d43963520d6f2697d8ad32988171.nothing.q.2021.ugractf.ru:443 -showcerts |
> openssl x509 -text -noout | grep 'ugra'
        Subject: CN = 0b76d43963520d6f2697d8ad32988171.nothing.q.2021.ugractf.ru
                ugra_v1_pOsmOtr3l1_1c1969d85b50
                DNS:0b76d43963520d6f2697d8ad32988171.nothing.q.2021.ugractf.ru
```

В Chrome всё ещё проще: прямо в интерфейсе можно посмотреть все поля сертификата, включая то, что нас интересует:



Флаг: **ugra_v1_p0sm0tr3l1_1c1969d85b50**

Восьмиборье

Стеганография, 150 баллов.

> При осмотре места преступления был обнаружен диск с разнообразными файлами. Сможете отыскать на нём секретную информацию?
>
> *image.iso*

Решение

Таск — стеганография в метаданных ISO.

Скачиваем образ диска. При его осмотре выясняется, что внутри ровно восемь разнообразных файлов. Вряд ли информация спрятана в каждом из них по отдельности, да и ISO-образ тут был дан неспроста. На это же намекает файл `cdrtools-source.zip`. Поэтому предплагаем, что информация сокрыта в метаданных диска.

Найти зацепку можно было несколькими способами:

- С помощью утилиты `isoinfo`:

```
$ isoinfo -f -i image.iso
/PFPWC427.003;1
/MQ4DANDE.007;1
/L5RGKOJT.006;1
/NZXXIX3B.001;1
/OVTXEYK7.000;1
/MVSF65DP.005;1
/ONPWKYLT.002;1
/NF2F65LT.004;1
```

- С помощью утилиты `isodump`:

```
$ isodump image.iso
144 [ 1]    1c     2048 02/*.            [SP,RR=1,PX=2,TF,CE=[1e,0,237][ER]]
110 [ 1]    1c     2048 02/*..           [RR=1,PX=2,TF]
150 [ 1]    1f    73529 00/ PFPWC427.003;1[RR=1,NM=2d_rigid_bodies.ipynb,PX=1,TF]
148 [ 1]    43  2165175 00/ MQ4DANDE.007;1[RR=1,NM=big_buck_bunny.webm,PX=1,TF]
148 [ 1]   465  3796533 00/ L5RGKOJT.006;1[RR=1,NM=cdrtools-source.zip,PX=1,TF]
142 [ 1]   ba3  1162810 00/ NZXXIX3B.001;1[RR=1,NM=challenge.png,PX=1,TF]
146 [ 1]   ddb     8631 00/ OVTXEYK7.000;1[RR=1,NM=iris-dataset.xlsx,PX=1,TF]
148 [ 1]   de0   144165 00/ MVSF65DP.005;1[RR=1,NM=olympiads_order.pdf,PX=1,TF]
146 [ 1]   e27   105243 00/ ONPWKYLT.002;1[RR=1,NM=sound_example.ogg,PX=1,TF]
152 [ 1]   e5b   693170 00/ NF2F65LT.004;1[RR=1,NM=youll_need_this_too.swf,PX=1,TF]

 Zone, zone offset:     1c 0000
```

- С помощью шестнадцатеричного редактора, просматривая таблицу файлов на диске:

```
0000e0ca 02 EB 03 00 00 00 00 03 EB C9 00 00 00 00 00 00 C9 1C 00 00 00 00 00 ........................
0000e0e1 00 1C 54 46 1A 01 0E 79 02 1A 12 1D 2F 0C 79 02 1A 13 2D 2E 0C 79 02 ..TF...y..../.y...-..y.
0000e0f8 1A 12 1F 1C 0C 00 96 00 1F 00 00 00 00 00 00 1F 39 1F 01 00 00 01 1F ................9.......
0000e10f 39 79 02 1A 12 1D 32 0C 00 00 00 01 00 00 01 0E 50 46 50 57 43 34 32 9y....2.........PFPWC42
0000e126 37 2E 30 30 33 3B 31 00 52 52 05 01 89 4E 4D 1A 01 00 32 64 5F 72 69 7.003;1.RR...NM...2d_ri
0000e13d 67 69 64 5F 62 6F 64 69 65 73 2E 69 70 79 6E 62 50 58 2C 01 A4 81 00 gid_bodies.ipynbPX,.....
0000e154 00 00 00 81 A4 01 00 00 00 00 00 00 01 EB 03 00 00 00 00 03 EB C9 00 ........................
0000e16b 00 00 00 00 00 C9 1F 00 00 00 00 00 00 1F 54 46 1A 01 0E 79 02 1A 12 ..............TF...y...
0000e182 1D 32 0C 79 02 1A 12 1F 1C 0C 79 02 1A 12 1F 1C 0C 00 94 00 43 00 00 .2.y......y.........C..
0000e199 00 00 00 00 43 B7 09 21 00 00 21 09 B7 79 02 1A 12 1D 32 0C 00 00 00 ....C..!..!..y....2....
0000e1b0 01 00 00 01 0E 4D 51 34 44 41 4E 44 45 2E 30 30 37 3B 31 00 52 52 05 .....MQ4DANDE.007;1.RR.
0000e1c7 01 89 4E 4D 18 01 00 62 69 67 5F 62 75 63 6B 5F 62 75 6E 6E 79 2E 77 ..NM...big_buck_bunny.w
0000e1de 65 62 6D 50 58 2C 01 A4 81 00 00 00 00 00 81 A4 01 00 00 00 00 00 01 ebmPX,.................
0000e1f5 EB 03 00 00 00 00 03 EB C9 00 00 00 00 00 00 00 C9 43 00 00 00 00 00 .................C......
0000e20c 43 54 46 1A 01 0E 79 02 1A 12 1D 32 0C 79 02 1A 12 1F 1C 0C 79 02 1A CTF...y....2.y......y..
0000e223 12 1F 1C 0C 00 94 00 65 04 00 00 00 00 04 65 35 EE 39 00 00 39 EE 35 .......e......e5.9..9.5
0000e23a 79 02 1A 12 1D 32 0C 00 00 00 00 01 00 00 01 0E 4C 35 52 47 4B 4F 4A 54 y....2..........L5RGKOJT
0000e251 2E 30 30 36 3B 31 00 52 52 05 01 89 4E 4D 18 01 00 63 64 72 74 6F 6F .006;1.RR...NM...cdrtoo
0000e268 6C 73 2D 73 6F 75 72 63 65 2E 7A 69 70 50 58 2C 01 A4 81 00 00 00 00 ls-source.zipPX,.......
```

Рис. 9: Вид таблицы

В любом случае, можно было заметить, что сокращённые названия файлов (так называемые имена в формате 8.3) сильно отличаются от тех, которые мы видим на диске. Эти названия также можно увидеть, смонтировав образ специальным образом: `mount -o norock,nojoliet image.iso /mnt/image`.

Названия файлов представляют собой флаг, закодированный в base32, а числа в расширении намекают на порядок. Обнаружить используемый алгоритм можно, например, с помощью CyberChef. В нём есть рецепт Magic, автоматически ищущий кодировку на основе энтропии. Подробнее об этом можно почитать в нашем курсе.

Флаг: **ugra_not_as_easy_as_it_used_to_be93d804d**

Служба одного порта

Сетевые технологии, 200 баллов.

> Открыта новая социальная служба в информационной сети «Интернет», обеспечивающая комплексное обслуживание населения в удобных условиях. Сможете воспользоваться всеми услугами?
>
> https://onestop.q.2021.ugractf.ru
> Пароль: ...

Решение

Таск о скрытии нескольких сервисов за одним портом с помощью `sslh`.

Заходим на сайт службы. Видим новость, в которой сообщается, что служба открыла административный отдел. Это, а также название таска, намекает нам о возможности зайти на тот же порт по SSH. Если сделать это, мы обнаружим сообщение:

```
$ ssh ctf@onestop.q.2021.ugractf.ru -p 443
Добро пожаловать в систему одного порта, отдел административный!

Доступные также защищённые отделы: информационный, текстовый.
ctf@onestop.q.2021.ugractf.ru: Permission denied (publickey).
```

Как мы видим из описания, всего на одном порту действуют три службы: административная, информационная и текстовая. Мы уже получили доступ к двум службам из трёх; осталась лишь текстовая служба. Вместе с информационной она описывается как «защищённая», что намекает на HTTPS или TLS в принципе.

Ищем в Интернете информацию, как в принципе реализуются такие службы. Находим и открываем сайт sslh — наиболее популярного демона для реализации диспетчеризации по портам. Здесь в примере конфигурации описываются различные способы диспетчеризации подключения. Смотрим на варианты диспетчеризации по TLS: используется либо SNI hostname, либо ALPN. Поскольку «отделами» в таске называются разные протоколы, логично предположить, что используется как раз ALPN. Для экспериментов можно использовать утилиту `openssl`, например: `openssl s_client -alpn example onestop.q.2021.ugractf.ru:443`.

Пробумем название сервиса `текстовый`: бинго! Видим приглашение `PWD?`. Введя в него пароль из задания, получаем искомый флаг.

Флаг: **ugra_social_services_for_the_masses_ff76c5513401**

Мощный шифр

Криптография, 250 баллов.

> Мощный шифр требует мощного компьютера для расшифрования. Мы уже написали скрипт, но запускать не на чем. Может быть, у вас найдётся подходящий компьютер?
>
> *powerful.key*
> *decrypt.py*
> Флаг: ...

Решение

В этом задании участникам предлагалось расшифровать некоторое сообщение. Был дан ключ, само зашифрованное сообщение и даже скрипт для расшифровки. Кажется, что всё более чем элементарно — просто запустить скрипт и всё готово. Однако, не тут-то было:

```
$ python3 decrypt.py powerful.key
RRSSA DECODER
============
WARNING! This script might work slowly, please wait patiently and don't interrupt the execution!
Enter your encrypted data: 7694194 15390722 14368614 19069969 23723442 16624648 2394527 34888380 22598911 11441468
8290797 17392772 22533699 3593467 8223017
Your data is: ugr
```

На трёх символах скрипт зависает и не подаёт никаких признаков жизни, кроме потребления 100% мощностей одного ядра.

Прервав скрипт (несмотря на явное прупреждение о том, что делать этого не надо), видим, что выполнение скрипта падает на строке

```
x = (a ** b) ** (c ** d)
```

На Python `**` означает возведение в степень. Давайте посмотрим, что же возводится, и в какую степень. Видим, что дешифрование производится блоками по три символа. a — как раз очередной элемент из потока зашифрованных данных.

Остальные переменные берутся из `powerful.key` — это файл с данными, закодированными в base64, внутри которого лежит JSON. В `private_key` лежат пары `(c, d)`, а в `common_key` — пары `(b, n)`. Число n используется чуть позже: для получения символов используется не само число x, а его остаток от деления на n.

Первое число вычисляется быстро: в ключе `b = c = d = 1`, и мы получаем первые три символа флага: `ugr`. Все же остальные числа имеют длину 20–30 бит (7–9 десятичных знаков), и даже одно возведение в степень уже вряд ли вычислится на среднем современном процессоре.

Дальше будут использованы несколько математических терминов. Под *вычислением по модулю n* мы будем понимать вычисление остатка от деления на n. Два числа называются *сравнимыми* или *равными по модулю n*, если их остатки от деления на n равны. Разумеется, везде, где не сказано иное, подразумеваются целые неотрицательные числа. *Взаимно простыми* числами называются два таких числа, у которых нет общего делителя, кроме единицы.

Давайте придумаем, как соптимизировать вычисление. Во-первых, довольно логично, что раз мы результат берём по модулю, можно брать по модулю результат на каждой итерации. В Python это проще всего записать так: `pow(a ** b, c ** d, n)`. Функция `pow` в Python с тремя аргументами эффективно вычисляет степень числа по нужному модулю.

Эта оптимизация уже сильно упрощает процесс, но этого мало. Рассмотрим выражение $a^b \bmod n$, где $a > n$. a представимо в виде $a = kn+x$, где $x = a \bmod n$. Тогда $(kn+x)^b = kn^b+b(kn)^{b-1}x+...+x^b$. И, поскольку все элементы, кроме последнего, делятся на n, то выражение сравнимо с x^b по модулю n.

Таким образом, от выражения a^b нас интересует только остаток от деления на n, и наше выражение превращается в `pow(pow(a, b, n), c ** d, n)`. Соптимизировать c^d таким же образом не выйдет — к сожалению, мы получим другой результат.

Назовём наше число `pow(a, b, n)` буквой q. Мы хотим посчитать остаток от деления e в некоторой степени на n. Рассмотрим последовательность: $q \bmod n$, $q^2 \bmod n$, $q^3 \bmod n$, ... Все эти числа лежат в диапазоне от 1 до $n-1$.

> Здесь мы используем тот факт, что q и n взаимно просты. Это было верно для наших сообщений (и в целом, подобные условия нередко являются обязательными для различных криптографических систем), но в общем случае такие вещи стоит проверять, чтобы не ошибиться.

Очевидно, что рано или поздно остатки начнут повторяться — хотя бы на n-той операции. При этом если мы какой-то повторяющийся остаток снова умножим на q, то получим то же самое число — таким образом, последовательность зацикливается. Если мы найдём длину этого цикла, возводить в нужную степень и не придётся — все итерации цикла можно просто пропустить, вычислив вместо нужной степени остаток от её деления на длину цикла.

Поскольку наш модуль был небольшим, можно было просто перебрать все степени q, пока мы не встретим этот самый цикл. Но, на самом деле, за нас уже всё сделал Леонард Эйлер — его теорема гласит о том, что длина такого цикла для взаимнопростых q и n равна функции Эйлера числа n. Функция Эйлера $\phi(n)$ — это количество натуральных чисел, меньших n и взаимно простых с ним.

Опять же, её можно посчитать и наивно, но с помощью несложных математических преобразований можно вычислить функцию Эйлера, зная делители n.

Таким образом, итоговая формула приобретает такой вид:

```
pow(pow(a, b, n), pow(c, d, phi(n)), n)
```

Используя её, мы мгновенно получаем флаг.

Флаг: **ugra_it_is_too_powerful_rsa_right_f753a15a12c**

Вокруг да около

Реверс-инжиниринг, 150 баллов.

> Компания “Круглые решения” представила свой новый программный продукт — аналоговые часы. Сможете найти в нём секретное послание?
>
> *round.zip*

Решение

Таск — реверс с поиском строки в приложении.

Видим приложение под Windows на Qt. Запускаем приложение, и видим круглые часы — ничего больше. Нам говорят о некоем секретном послании в часах, а значит, придётся открывать отладчик. В разборе мы используем Cutter. Про реверс-инжиниринг с самых азов можно почитать у нас в курсе.

Запукаем Cutter, и видим некоторое количество функций в программе. Давайте осмотримся, просматривая функции по одной, и грубо разделяя их на следующие категории:

- Бизнес-логика: в таких функциях много вызовов других функций и внешних библиотек;
- Вычислительные: в таких функциях много разнообразных инструкций, в том числе с использованием векторных операций (MMX/SSE), и мало вызовов библиотек;
- Другие: короткие функции с парой вызовов. Иногда такие генерируются компилятором, иногда это — короткие процедуры-обёртки.

Во время изучения можно сразу переименовывать функции по их примерному содержанию. Например, если сверху вы увидите упоминание какого-то внешнего вызова, вы можете примерно догадаться о назначении функции. Неизвестные внешние функции (в основном, из Qt) лучше сразу искать в документации на библиотеку. Итого выделим интересные функции, сконцентрировавшись на функциях с внешними вызовами (здесь и далее адреса даются по расположению в исходном файле; при запущенном отладчике они будут различаться);

- `0x00401020`: Функция, где фигурирует `QWidget::constructor`, и `AnalogClock`: должно быть, компонент часов;
- `0x00401520`: Функция, где упоминается `QApplication::constructor`: начало работы программы;
- `0x00401cf0`: Функция с упоминанием `QMainWindow::constructor`, то есть создание главного окна программы;
- `0x00402240`: Интересная функция, в которой вызываются memset и memcpy;
- `0x004035e0`: Ещё одна функиця с memcpy;
- `0x004037f0`: Функция, которая явно делает что-то со шрифтами: в ней вызываются методы `QFontMetrics`;
- `0x00403dd0`: Ещё одна функция, выполняющая действия со шрифтами. В ней также вызывается `QLabel::constructor`, то есть создаются надписи где-то в интерфейсе программы;

Ниже последней находятся в основном функици, сгенерированные компилятором; нам они не так интересны.

В самой программе при этом мы не наблюдали никаких текстовых строк — это наводит на мысль, что у неё есть скрытые элементы интерфейса (возможно, открывающиеся по какому-то событию).

Проверим, что функция `0x00403dd0` вызывается: поставим на неё точку останова и запустим отладчик. Спустя несколько нажатий «Continue» мы попадаем в эту функцию, что означает, что метки и правда создаются во время создания главного окна. Изучая документацию, находим, что текст в QLabel устанавливается через вызов `setText`. Ищем его (он находится по адресу `0x004040e2`) и устанавливаем туда точку останова. Запускаем программу, и снова видим попадание — получается, каким-то меткам и правда присваивается текст.

Дальше было несколько вариантов размышления. Первый заключался в том, чтобы найти, где эти метки отрисовываются на форме. Тут поможет знание, что в современных интерфейсах круглые (и вообще фигурные) окна на самом деле не являются круглыми — это прямоугольные окна с заданной маской. Поискав, можно сразу же найти метод `setMask`, который позволяет задать такую маску в приложении на Qt. Находим вызов этого метода в Cutter (он находится по аресу `0x00401dca`) и отключаем его, заполняя NOPами (правая кнопка мыши *to* Edit *to* Nop instruction). Закрываем Radare, открываем приложение, и видим флаг вокруг часов в прямоугольном окне. На это решение намекает название и легенда к заданию.

Рис. 10: Окно с отключённой маской

Второе решение заключается в извлечении самого текста. Метод `setText` принимает `QString`. Смотрим выше, и находим метод `fromAscii_helper`, а ещё выше — несколько вызовов `memcpy`. Первый метод явно намекает на преобразование строк из классического формата Си (указатель на нультерминированную строку); на это указывают и его аргументы. Остановим программу перед его вызовом (адрес `0x00404072`). Запускаем программу, и смотрим значение вверху стека: им оказывается искомый флаг.

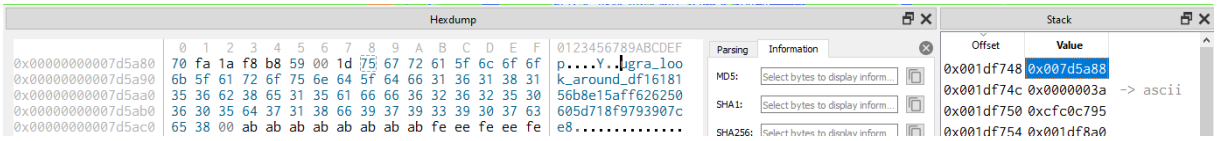


Рис. 11: Стек во время остановки на вызове `fromAscii_helper`

Флаг: **ugra_look_around_df1618156b8e15aff626250605d718f9793907ce8**

Огни города поменьше

Сетевая разведка, 400 баллов.

> Вам нужно найти точку съёмки этого прекрасного вида. На этот раз в файле даже есть геометка, вот только она совершенно бесполезна.
>
> *Примечание 1.* У вас есть восемь попыток. Проверяющая система беспощадна и принимает лишь точный ответ — прочитайте правила ввода и не тратьте попытки зря.
>
> *Примечание 2.* Проверяющая система не содержит известных авторам уязвимостей. Настойчивые попытки получить флаг способом, отличным от ввода верного ответа в рамках предоставленного количества попыток, считаются атаками на инфраструктуру с соответствующими последствиями для команды.
>
> IMG_0586.HEIC
> https://smallercitylights.q.2021.ugractf.ru/token/

Решение

Итак, дана фотография, и требуется найти точку съёмки панорамы. Не самой фотографии — она, как напрямую следует из метаданных, сделана в Икее Тёплый Стан — а вот той панорамы, которая помещена на фон постера.

Не удаётся даже понять, что это за город. На кадре нет каких-то узнаваемых достопримечательностей, которые можно было бы легко скормить поиску по картинкам, только какие-то невнятные церкви вдалеке и торчащие острые уголки некоего здания.

Что там на фоне, совсем далеко — вовсе непонятно. При всей невнятности бросается в глаза, что видимая область плотно застроена относительно высокими домами (9, 10, 12 этажей), и эти дома явно не относятся к новостройкам; точка съёмки находится ещё выше, на уровне не ниже 14–15 этажа. В маленьких городах такой застройки сложиться не могло, значит, мы имеем дело с каким-то довольно крупным городом бывшего СССР, вероятно, областным центром или столицей государства.

Вспоминаем (или берём из Википедии списком) крупные города и ищем виды с высоты в надежде узнать здание с уголками или — а вдруг повезёт! — панельные дома с их декором.

Рано или поздно наш перебор доходит до Омска — и на какой-нибудь из панорам мы видим то самое здание с уголками — Омский государственный музыкальный театр.

Находим его на карте *(10 лет Октября, 2)*, включаем на полную катушку пространственное мышление и прикидываем по ориентации уголков примерное направление поиска.

Было бы неплохо найти кирпичный дом с закруглением, он, в отличие от серых панельных домов, явно такой один, без раскиданной по всему району армии похожих домов, с которыми его можно перепутать. К тому же, панельные дома хоть и обладают отличительными признаками, в полусумраке и при освещении фонарей их может быть не узнать на снятой днём панораме.

Пытаемся найти дом похожей топологии на карте и на спутниковом снимке, придерживаясь оценённой области. Однако найти ничего не можем — удаляемся всё дальше и дальше. В отчаянии

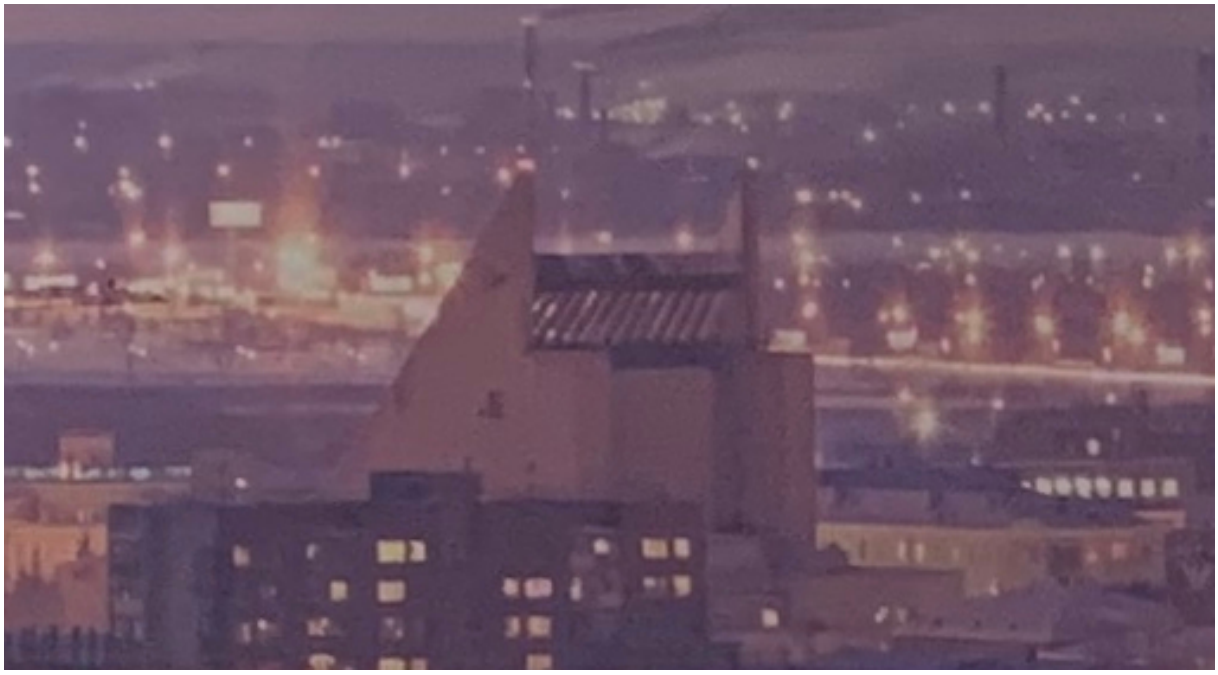


Рис. 12: Острые уголки здания

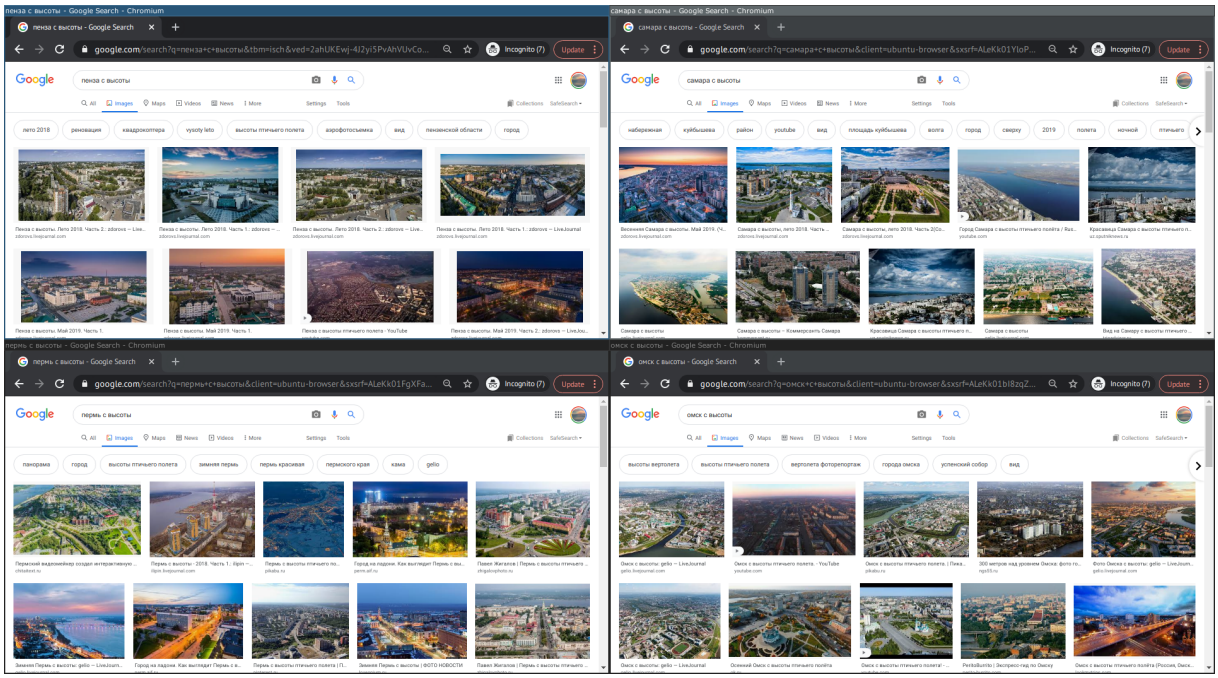


Рис. 13: Некоторые города с высоты

Рис. 14: Музыкальный театр

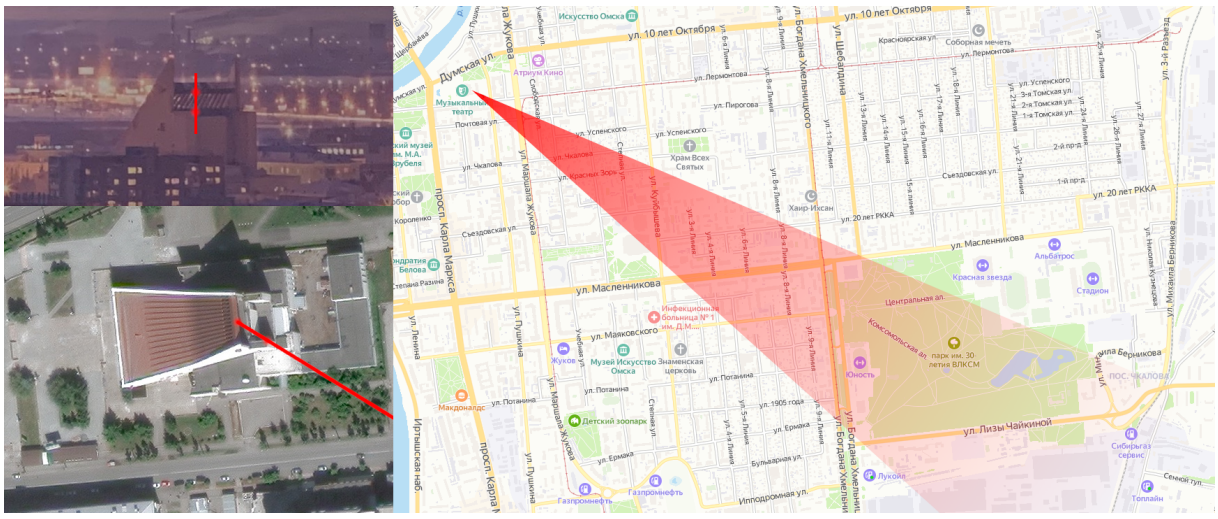


Рис. 15: Область поиска

Рис. 16: Кирпичный дом

разглядываем вообще любые дома, которые по спутниковому изображению могут показаться похожими на те, которые видно на панораме, удаляясь всё дальше и дальше от предполагаемой зоны поиска.

В какой-то момент нам попадается тот самый скруглённый дом, на который мы возлагали большие надежды *(10 лет Октября, 50)*. Обнаружив его совсем не там, всматриваемся и понимаем, что дом действительно тот, однако крыло слева от закругления на самом деле короче крыла справа, а на картинке — наоборот.

Теперь понятно, почему у нас ничего не получалось — **дизайнер, делавший постер для Икеи, зеркально отразил всю панораму**. Отражаем её обратно, чтобы не путаться, и смотрим на эту разотражённую нами версию. Снова призываем уголки театра для определения области поиска, на этот раз настоящей.

Разглядывая спутниковые снимки в поисках похожих по форме крыш и сверяясь с панорамами, методично находим дома вблизи: дом с тремя кругами над подъездами и тёмной полоской на четвёртом сверху этаже *(Иркутская, 3)* и, например, дом с тёмными рамками вокруг подъездных окон *(Омская, 110)*.

Здесь мы можем захотеть поискать похожие картинки в интернете: вдруг кадр на постере на самом деле взят не полностью, и есть какие-нибудь здания поближе, которые помогут лучше понять обстановку?

И действительно, находим версию с другой обрезкой и ближними домами, снятую хоть и в чуть другое время, но абсолютно точно с той же самой точки (неточности при наложении объясняются особенностями съёмки в Икее, однако мы чётко видим, что дома относительно друг друга расположены точно так же; съёмка даже из другой комнаты той же квартиры выглядела бы иначе). Заодно узнаём автора снимков.

Оглядываясь по сторонам в окрестности, опознаём ближайший дом *(Арнольда Нейбута, 14)*, дальний панельный дом с кирпичной пристройкой *(Омская, 117/2)* и ближний панельный дом с кирпичной пристройкой и выступами на балконах *(Омская, 119)*.

Рисуем на карте линии между точками, которые находятся на одной прямой относительно точки съёмки — и находим точку пересечения.

Линии указывают между домами 38 и 38/2 на улице Богдана Хмельницкого. Оба дома достаточно высокие. Однако рисование линий между даже самыми северными и западными границами дома 38 и различными домами панорамы приводит совсем не туда, куда приводит линия взгляда на снимке. Так, синяя линия через угол дома Арнольда Нейбута, 14 уверенно проходит через угол дома Омская, 119, но если мы попытаемся нарисовать линию от Богдана Хмельницкого, 38 через угол Арнольда Нейбута, 14, то дом Омская, 119 она обойдёт мимо.

Рис. 17: Кирпичный дом на панораме и в жизни

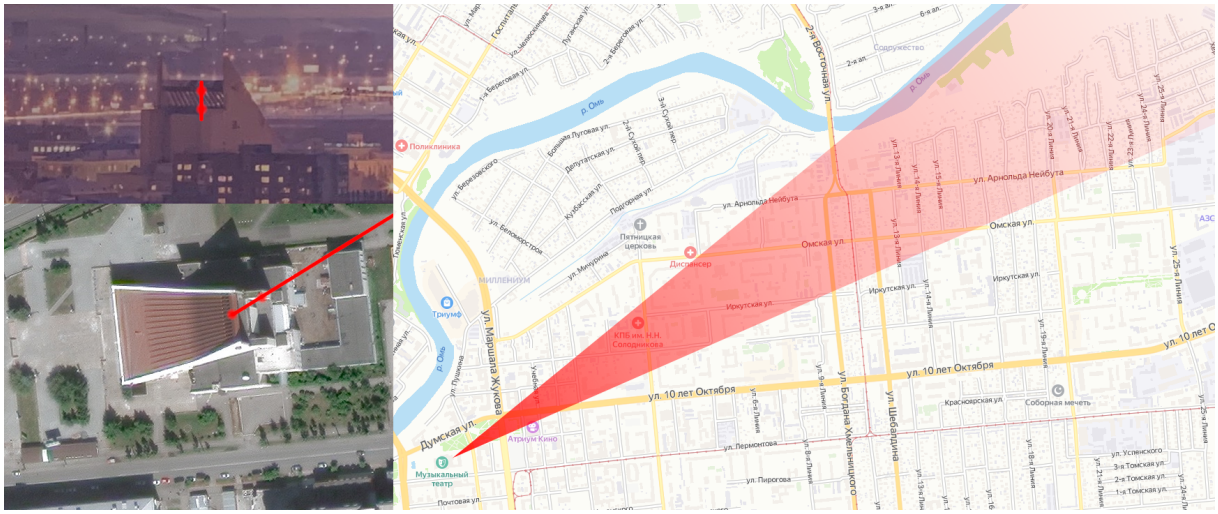


Рис. 18: Настоящая область поиска

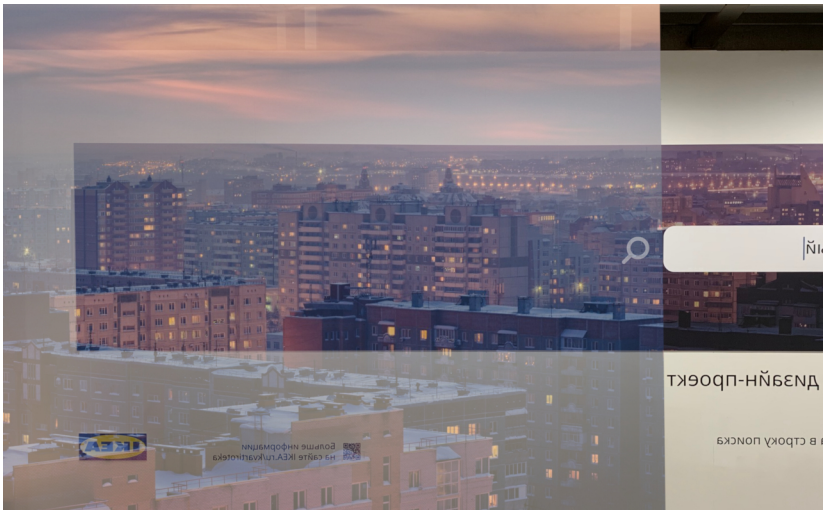


Рис. 19: Сопоставление панорамы и другой версии

Рис. 20: Линии на панораме

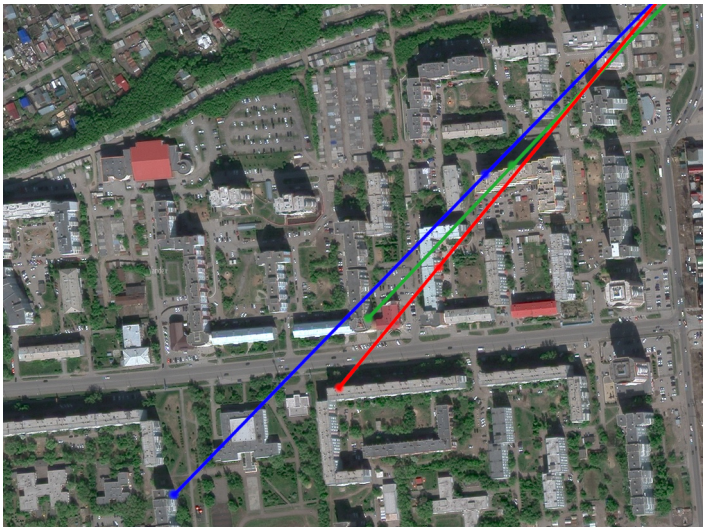


Рис. 21: Линии на карте

Рис. 22: Контрольная линия от дома 38

Таким образом, фотография сделана из дома 38/2 на улице Богдана Хмельницкого.

Кстати, на картах Гугла по неизвестной причине адреса указаны некорректно: дом 38 назван домом 38/2, а дом 38/2 — домом 38-корпус-2. Убедиться в том, что прав именно Яндекс, можно по панорамам — с определённых ракурсов можно разглядеть таблички 38 и 38/2, висящие на самих домах.

Итоговый ответ на задание в формате, ожидаемом проверяющей системой — **Россия, Омская область, г. Омск, ул. Богдана Хмельницкого, д. 38/2**.

Флаг: **ugra_we_will_go_we_will_swim_we_will_crawl_3fbdcedf2381**

Бонус: карта и список всех попыток сдать задание.

ЦНИИВТ

Стеганография, 150 баллов.

> Устроился системным администратором в Центральный НИИ вычислительной техники. На второй неделе работы обнаружил в локальной сети какой-то странный компьютер. Поспрашивал у коллег — все впервые слышат о нём, даже не знают, на каком этаже он стоит.
>
> Искал пару дней, в итоге нашёл его в подсобке под слоем пыли — такое ощущение, что лет 30 к нему никто не прикасался. Самое удивительное — это то, что он каким-то образом подключен к интернету, и на нём даже открыт порт. При подключении просит пароль, я его раздобыл в архиве. Но вместо данных сервер шлёт какой-то мусор.
>
> Поможете разобраться? Вдруг там что-то важное!

`nc soviet.q.2021.ugractf.ru 17792`
Пароль: ...

Решение

Участникам предлагается подключиться по TCP к порту. При подключении сервер спрашивает пароль, и в ответ выдаёт какой-то набор непечатаемых символов.

Судя по тому, что пароль спрашивали текстом, можно предположить, что и дальше идёт какой-то текст. Очевидно, что кодировка не UTF-8 — коды символов не начинаются ни с `0`, ни с `10`.

Попробуем различные кодировки. Намёки в условии указывают на то, что кодировка русскоязычная — таких не очень-то и много. В кодировке CP866 на первый взгляд ничего интересного. Однако, приглядевшись, мы видим, что это на самом деле QR-код без переносов строк: каждый символ соответствует области размером 1×2 пикселя. Подбираем ширину флага так, чтобы код без труда читался — его ширина равна 41 пикселю.

Это сходится и с легендой: действительно, кодировка CP866 была разработана в Советском Союзе и активно использовалась до прихода многобайтовых кодировок.

Декодируем QR-код (рис. 23) и получаем флаг.

Флаг: **ugra_soviet_technologies_are_eternal_6e6cc9c6e93b**

База знаний

Разное, 250 баллов.

> Для оперативного решения проблем клиентов международной платёжной системы UgraPay был открыт Департамент клиентской поддержки с представительством в платформе для обмена сообщениями Telegram.
>
> Правда, сотрудников в него так и не наняли...
>
> https://t.me/ugrapaysupportbot?start=token

Решение

Нам был дан Telegram-бот. Изучив его возможности, видим, что он умеет отвечать на 18 вопросов про некую платёжную систему Ugra Pay. На все команды и сообщения он отвечает, что операторов нет, и предлагает поискать вопрос в списке заготовленных.

Большего от бота добиться не удаётся — поэтому идём в документацию Telegram узнавать, что в целом умеют боты. Из ещё не изученных интерфейсов ввода нам встречается инлайн-режим: возможность отправки в чаты сообщений, подготовленных ботом. Например, многие пользуются этим режимом для отправки картинок или анимированных изображений:

Неожиданно мы выясняем, что эту же возможность поддерживает и наш бот для поддержки платёжной системы:

При выборе сообщения оно отправляется в чат:

Мы обращаем внимание, что поиск находит все вопросы, содержащие наш запрос. Но как же устроен поиск? Пробуем вводить различные данные и обнаруживаем, что при вводе апострофа (`'`) поиск перестаёт выводить результаты. Это может означать как ошибку, так и отсутствие результатов.

Нередко апострофы используются в качестве индикаторов начала и конца текстовой строки, в частности, в языке запросов SQL. Проверить, что мы реально «выбрались» из строки, можно разными способами. Мне, например, нравятся два таких:

Рис. 23: Вывод в кодировке CP866

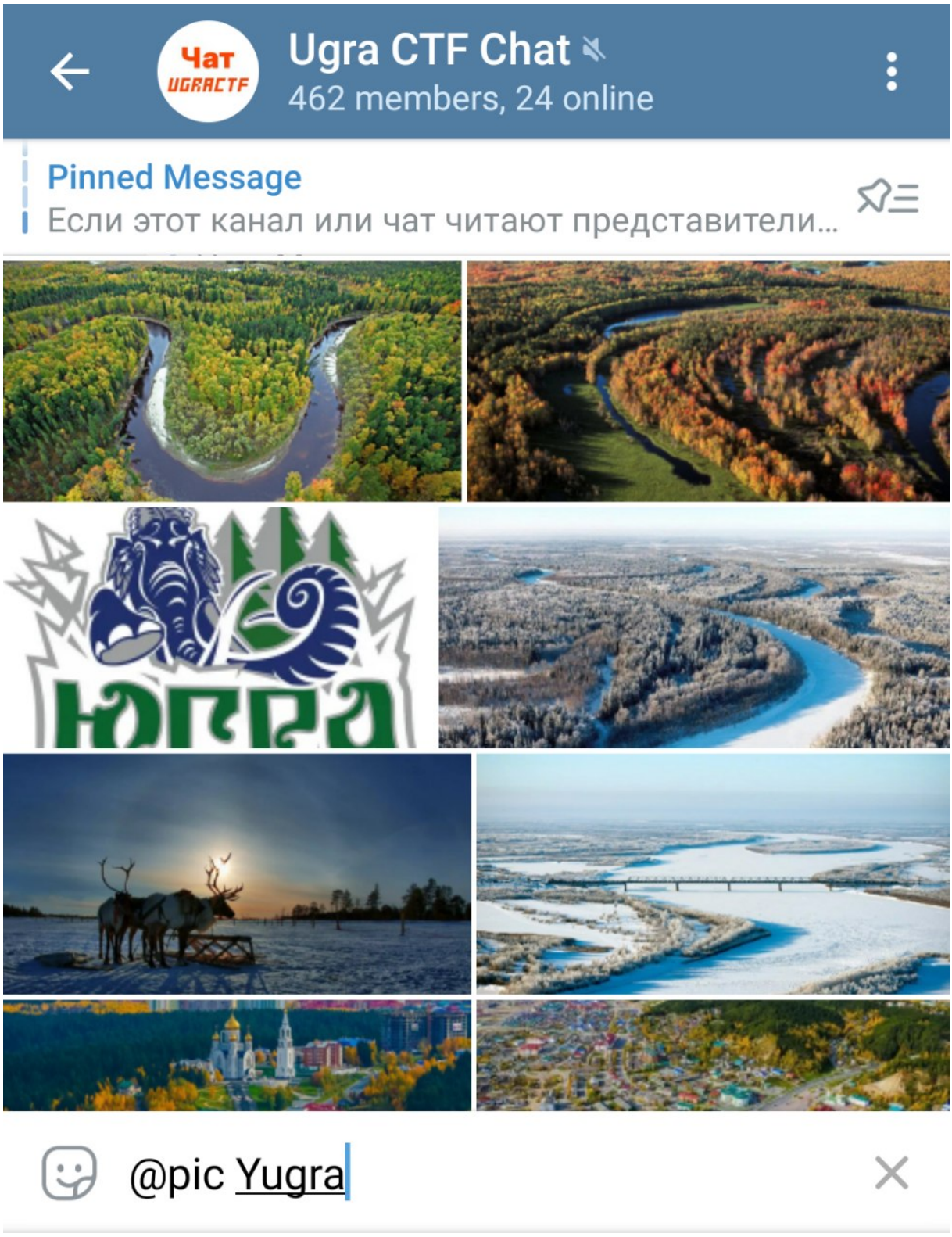


Рис. 24: Можно вставить картинку через инлайн-режим бота @pic

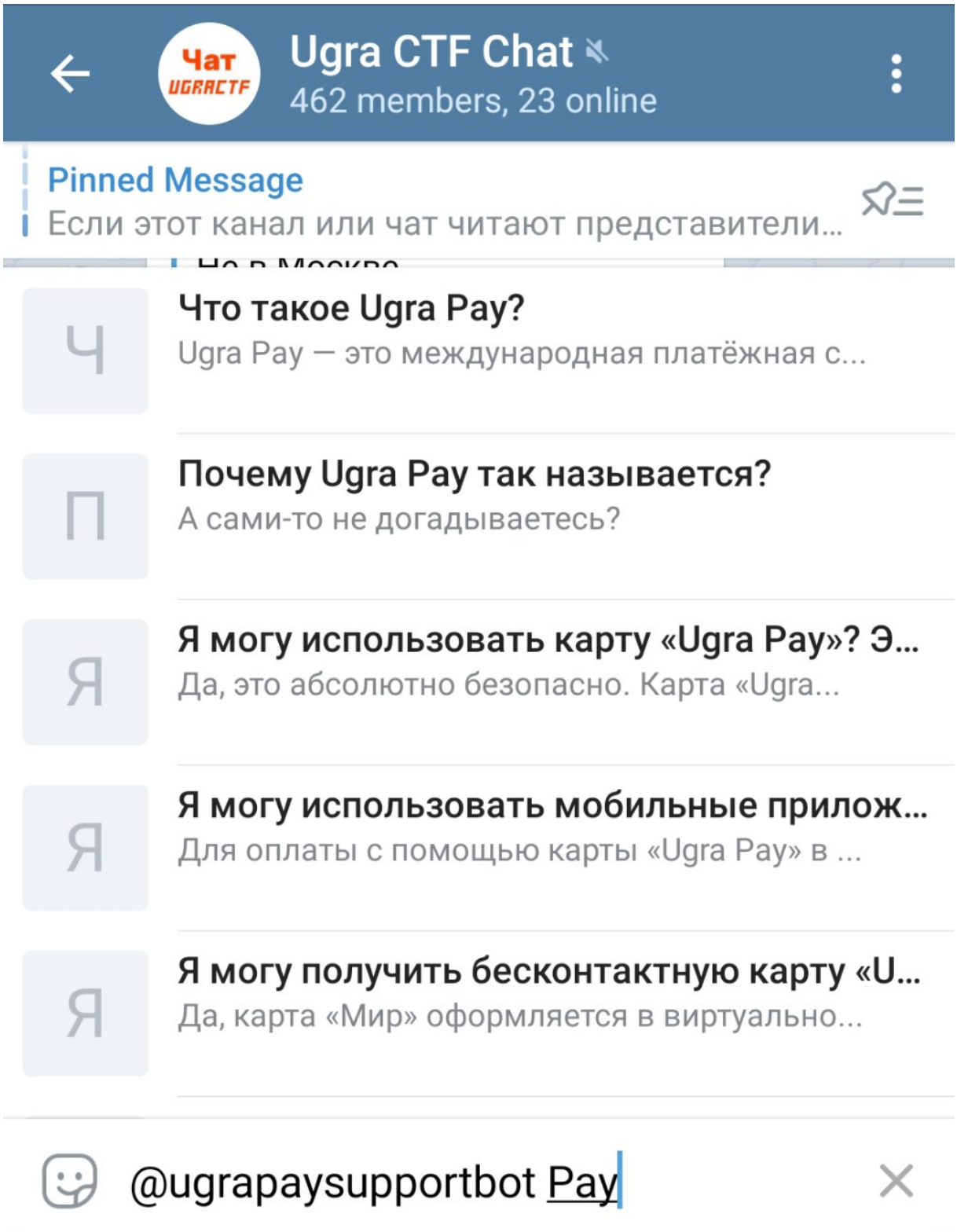


Рис. 25: Инлайн-режим бота

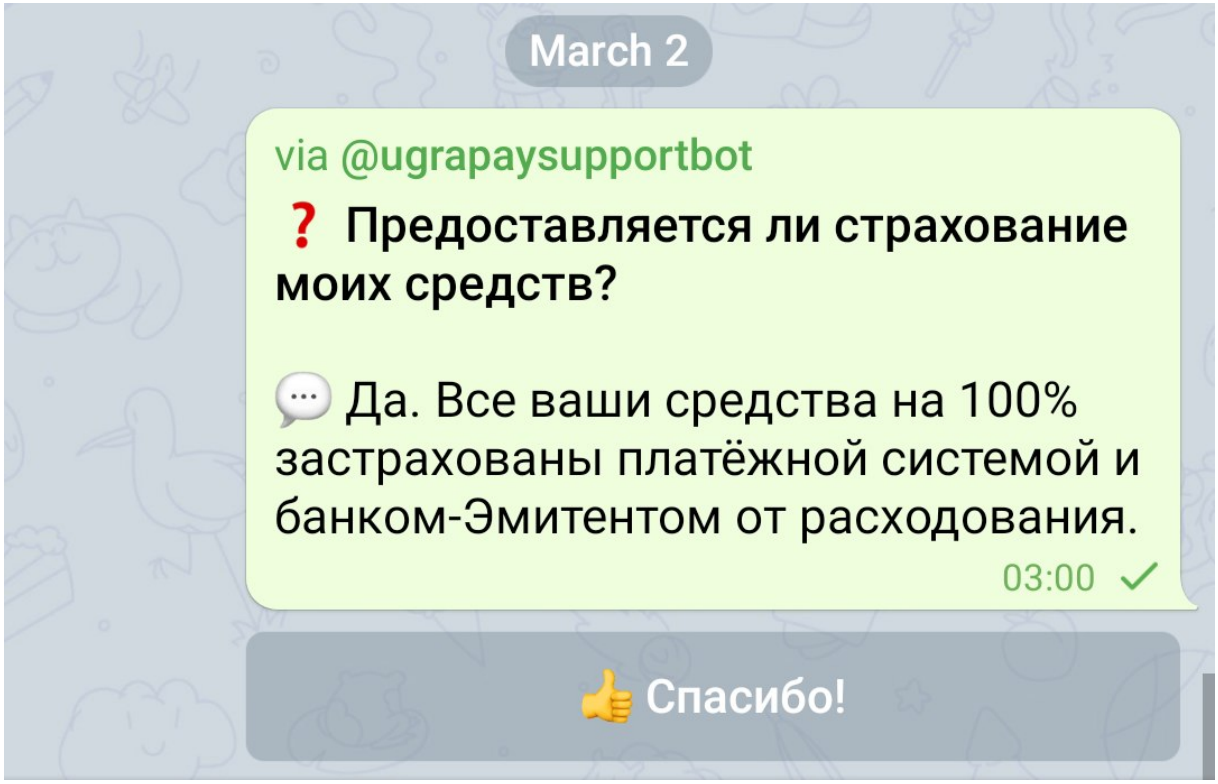
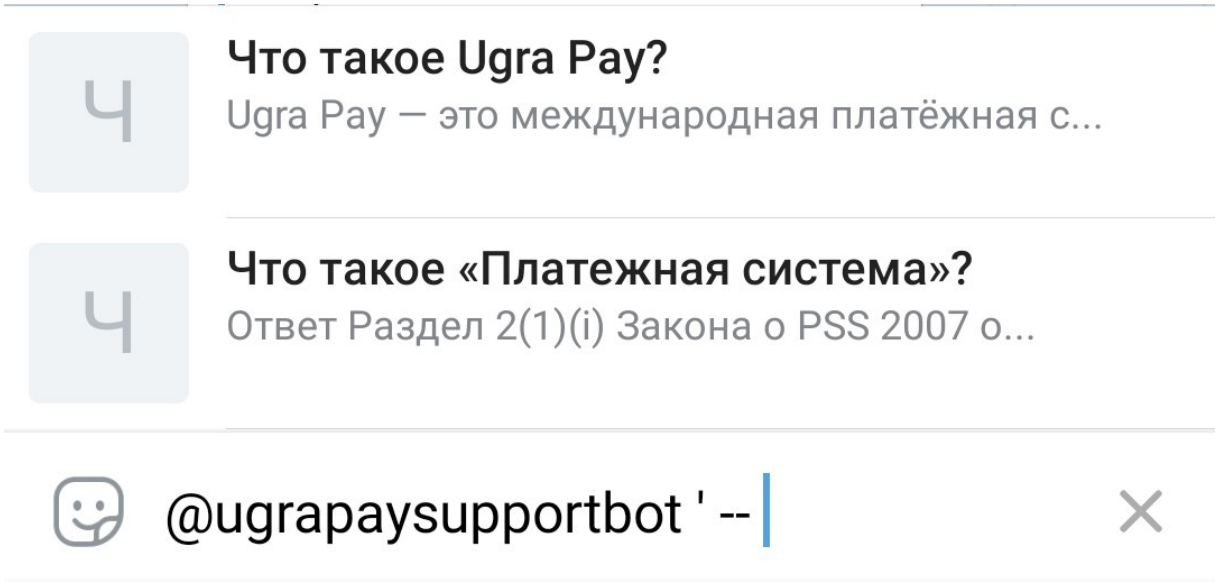


Рис. 26: Вопрос



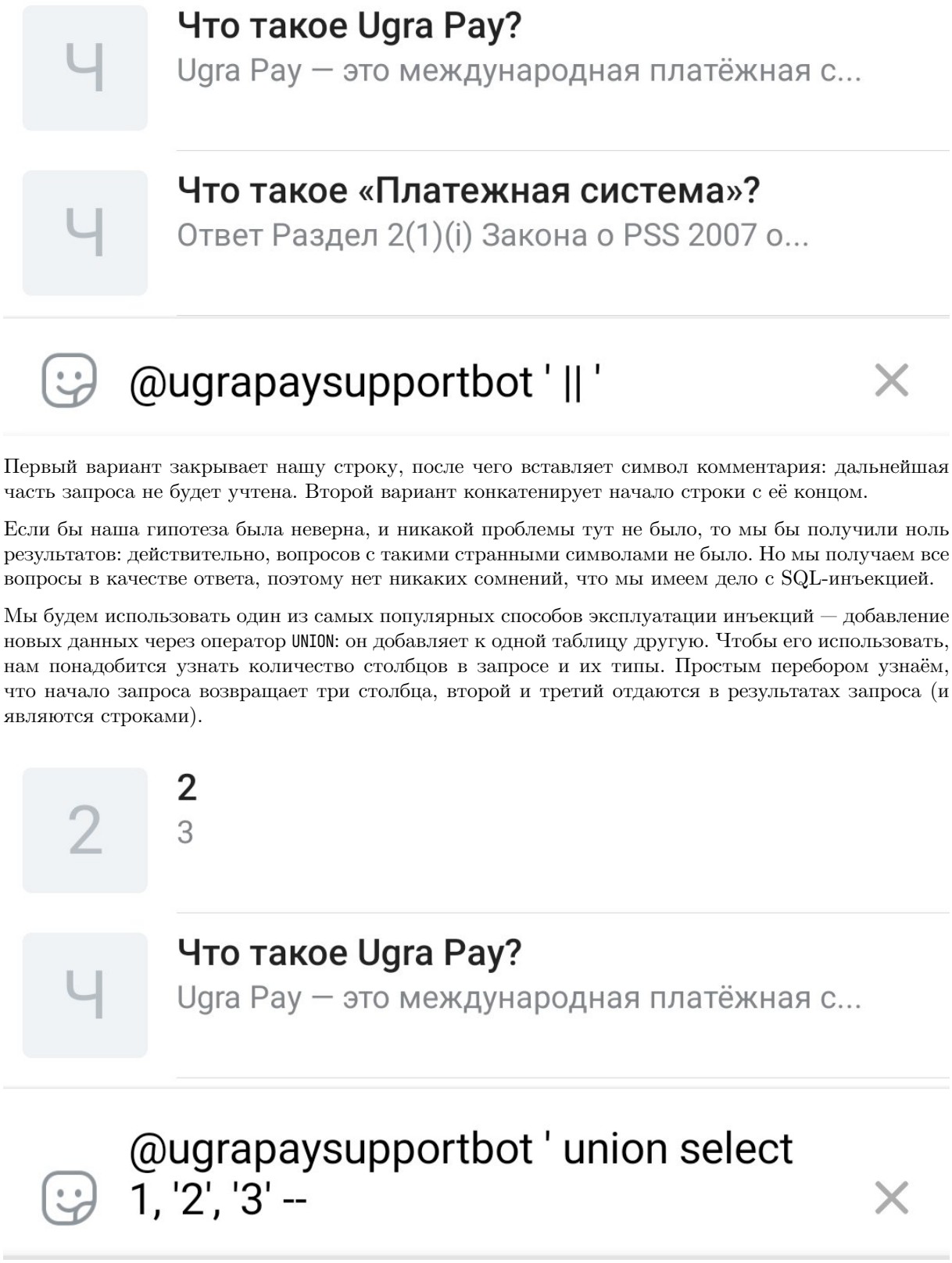


Рис. 27: Определение числа столбцов

Чтобы успешно проэксплуатировать SQL-инъекцию, нам нужно узнать, какая конкретно СУБД находится на сервере. В этом нам помогут различные способы извлечь версию базы данных. Перебираем способы для всех популярных СУБД, и узнаём, что это SQLite:

На этом шаге можно перед `UNION` добавить в инъекцию условие `AND 1=0`, чтобы нам не мешали ответы на вопросы. Правда, это не даст нам различать ошибку сервера и отсутствие результатов — для нас в обоих случаях будет пустой ответ.

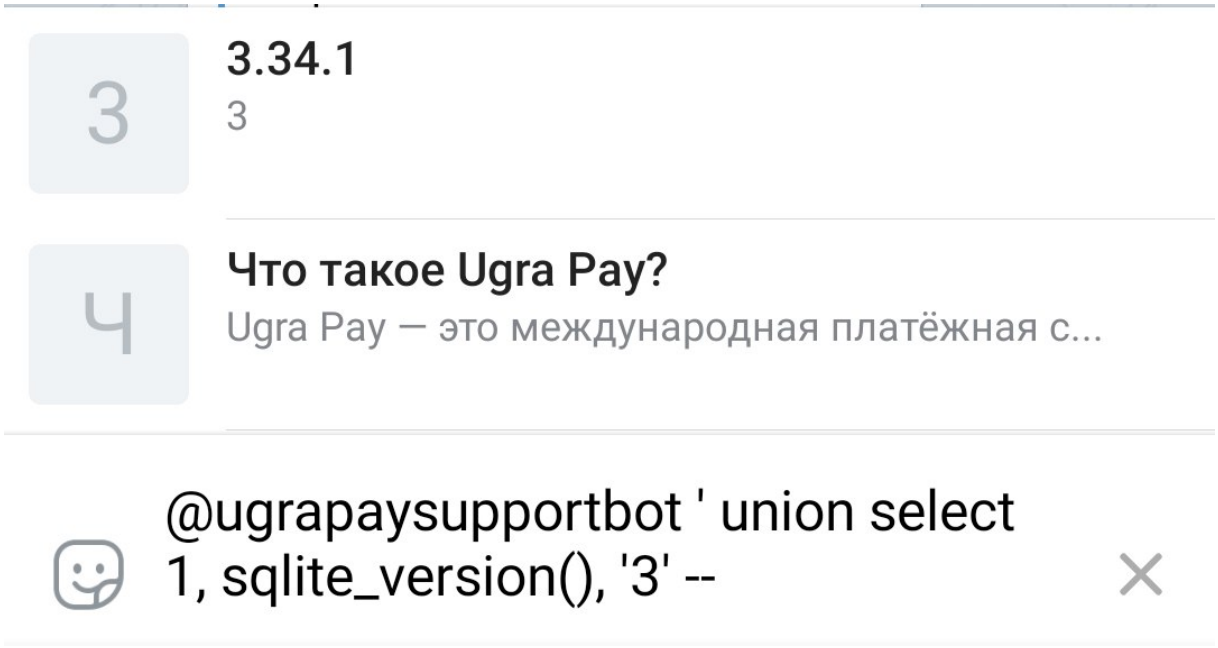


Рис. 28: Версия СУБД

Теперь попытаемся понять, что же можно достать из базы данных. В большинстве СУБД информация о структуре базы данных хранится рядом, в служебных таблицах. В SQLite таблица со структурой базы называется `sqlite_master`:

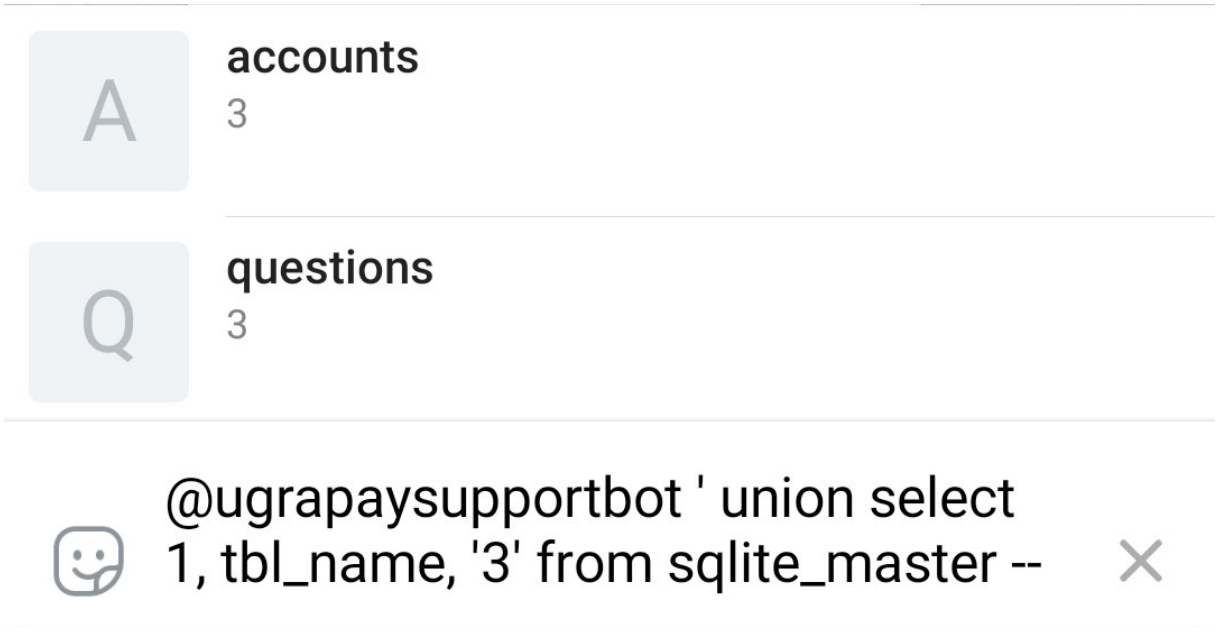
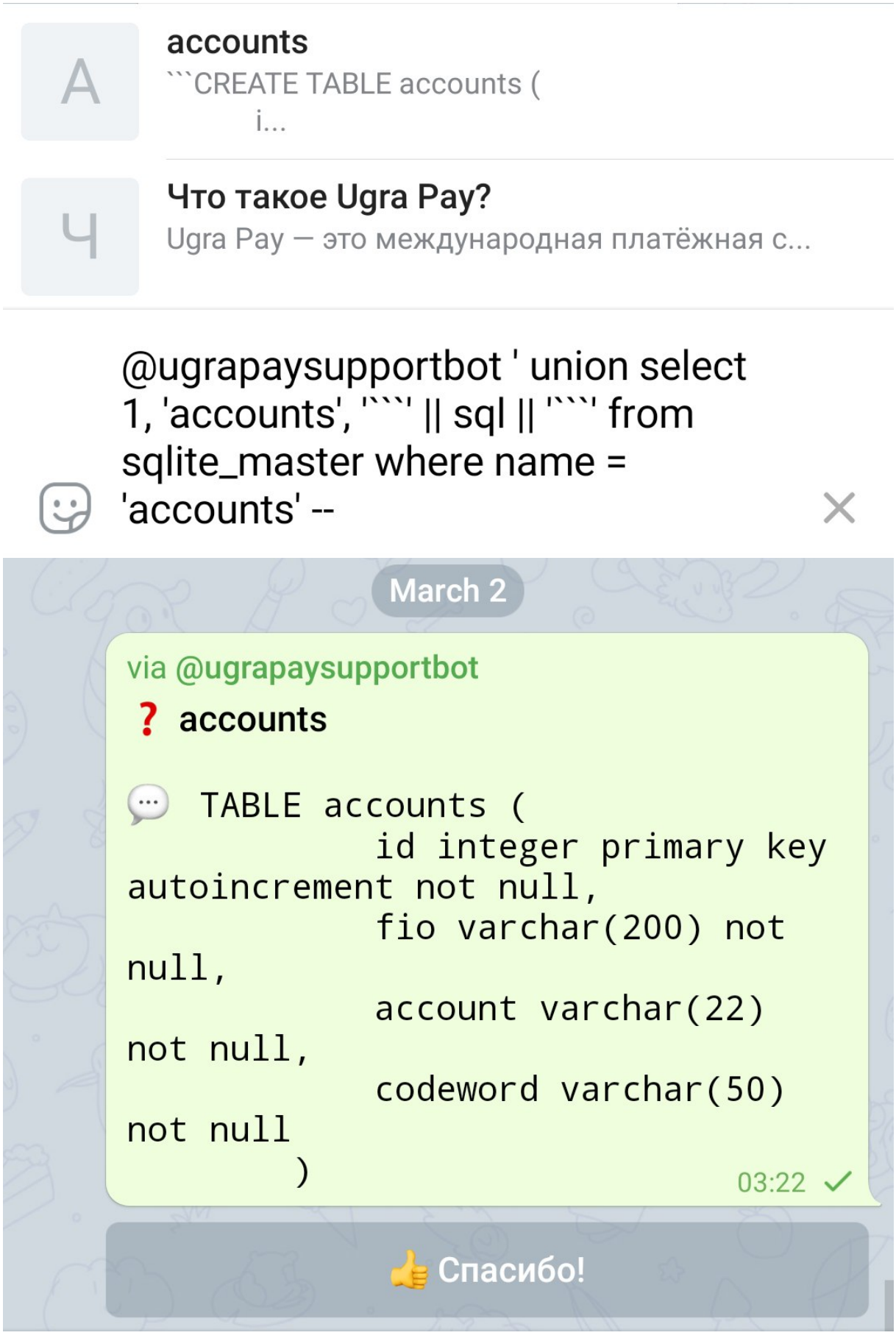


Рис. 29: Список таблиц

Видим, что кроме таблицы `questions` есть таблица `accounts`. Вероятно, это то, что нам нужно. Из этой же таблицы можно достать информацию о структуре таблицы. При этом, чтобы не получать ответ по частям, просто отправим сообщение — оно вмещает гораздо больше информации, чем предпросмотр:



В таблице `accounts` четыре столбца — идентификатор, имя, номер счёта и кодовое слово. Извлекаем кодовые слова всех пользователей, замечаем идущих подряд пользователей с весьма говорящими словами.

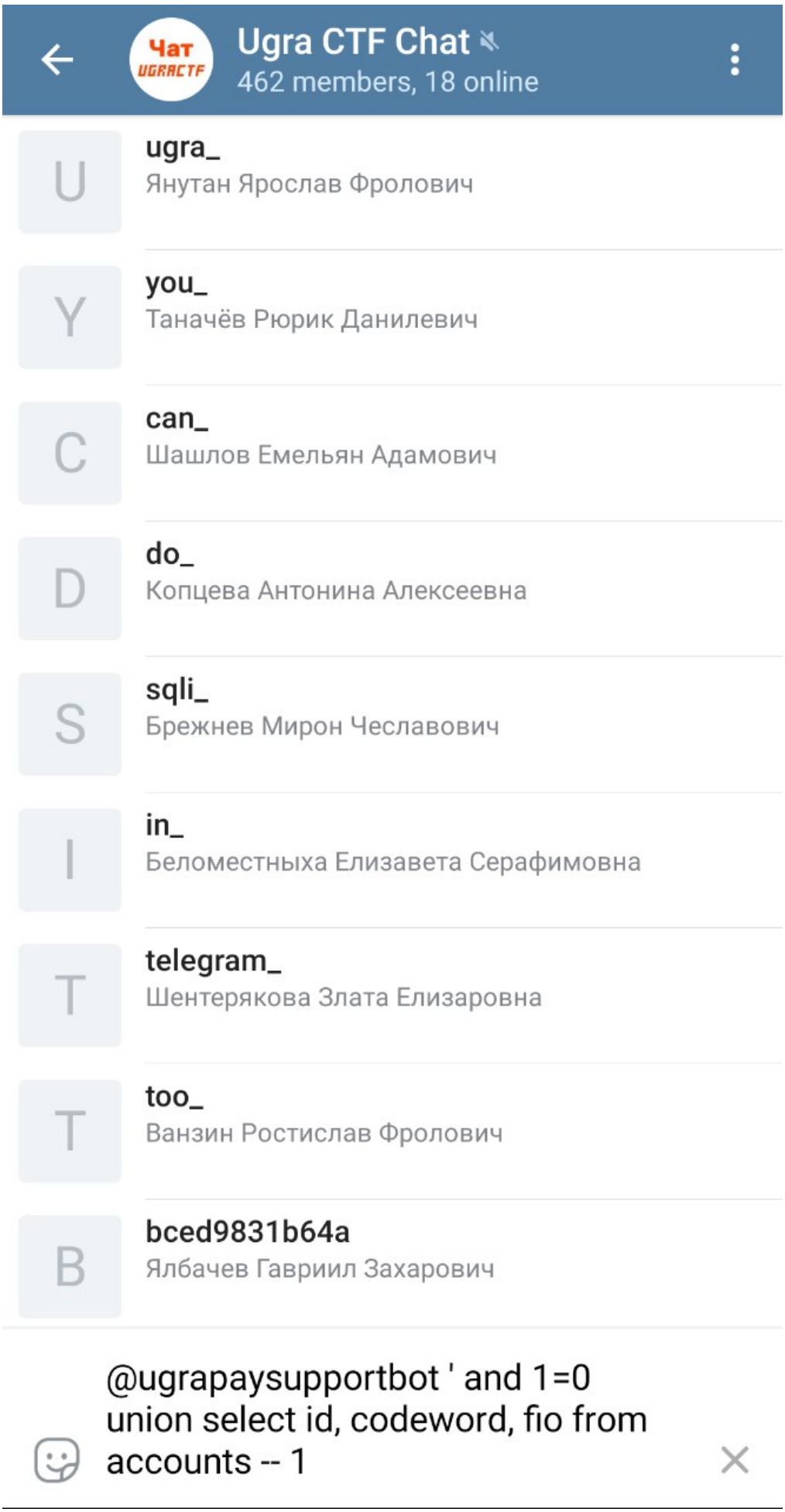

Ugra CTF Chat
462 members, 18 online
U
ugra_
Янутан Ярослав Фролович
Y
you_
Таначёв Рюрик Данилевич
C
can_
Шашлов Емельян Адамович
D
do_
Копцева Антонина Алексеевна
S
sqli_
Брежнев Мирон Чеславович
I
in_
Беломестныха Елизавета Серафимовна
T
telegram_
Шентерякова Злата Елизаровна
T
too_
Ванзин Ростислав Фролович
B
bced9831b64a
Ялбачев Гавриил Захарович
@ugrapaysupportbot ' and 1=0
union select id, codeword, fio from
accounts -- 1

Рис. 30: Флаг

Флаг: **ugra_you_can_do_sqli_in_telegram_too_bced9831b64a**

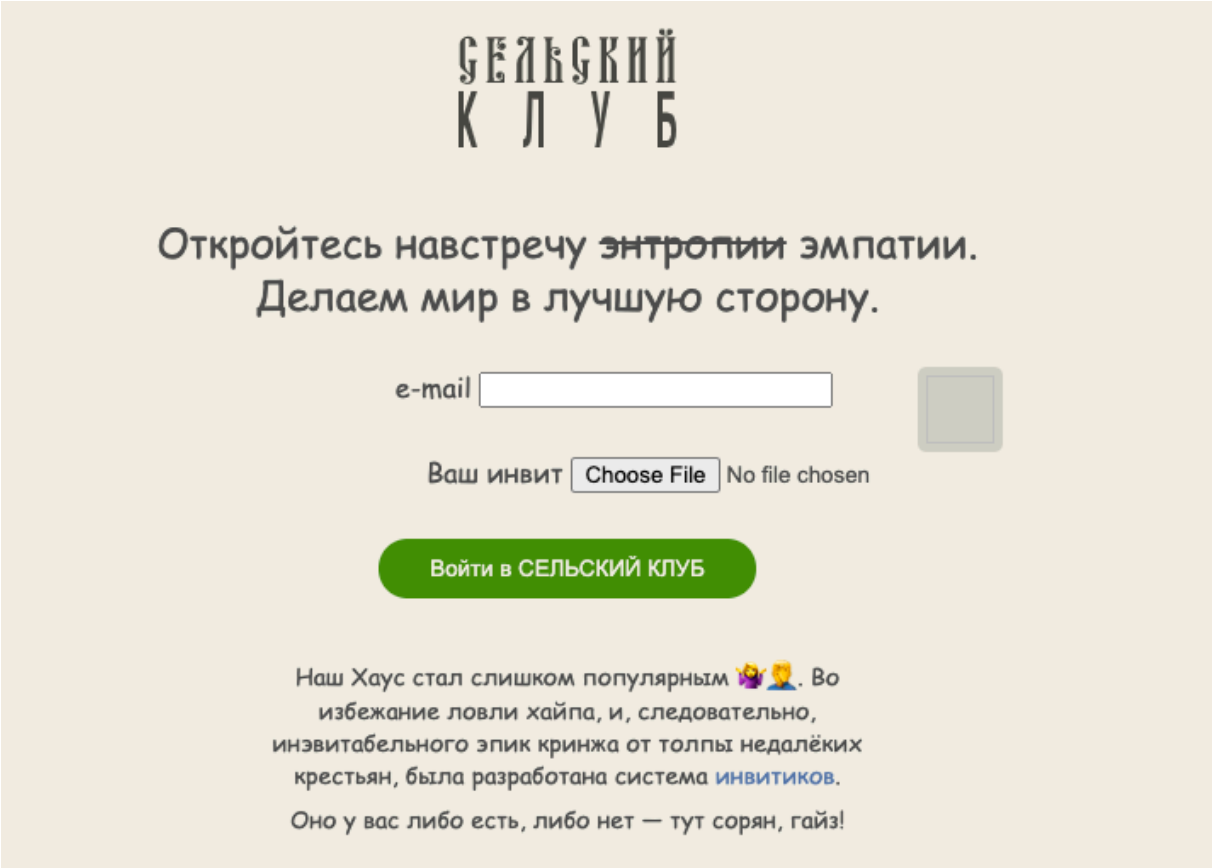
Сельский клуб

Веб-технологии, 300 баллов.

> Лучшее место для того, чтобы понять, как устроен медиабизнес. С одной стороны, это невероятно хайповый сервис, который сейчас переживает ошеломляющую волну популярности, а с другой стороны, много людей ещё не знают, что это.
>
> На одной из сургутских тусовок наш агент совершил акт шпионажа, в ходе которого было получено фото приглашения в эту загадочную социальную сеть. Надеемся, он ещё не протух!
>
> photo.png
> https://thevillage.q.2021.ugractf.ru/token/

Решение

Видим сайт, отчаянно пытающийся подражать модному нынче «Клабхаусу».



Чтобы попасть внутрь, необходимо где-то раздобыть инвайт. Благо, в условии таска дана шпионская фотография из-за плеча:

Если попробовать загрузить её как есть, сервер ругнётся ошибкой `413 Request entity too large`. Если, однако, кадрировать её по QR-коду, сервер примет её, но скажет, что «инвитик староват».

Можно выяснить, что закодировано в *инвитике*, воспользовавшись утилитой *zbarimg*, которая доступна на многих платформах и поддерживает большое количество одно- и двухмерных штрихкодов. Итак, QR-код на фотографии содержит в себе строковое представление числа `313337`, закодированное в *Base64*:

```
$ zbarimg photo_cropped.png
QR-Code:MzEzMzM3
scanned 1 barcode symbols from 1 images in 0.02 seconds
$ echo MzEzMzM3 | base64 -d
313337
```

Предположим, что:

- это порядковый номер инвайта,
- инвайтов ограниченное количество,
- есть свободные инвайты.

Значит, можно решить задачу перебором. Это правда лишь отчасти, потому что сервер не даёт выполнить больше одного запроса в секунду, а пространство перебора может оказаться очень большим. Попробуем нащупать его, заслав несколько случайных *инвитиков* и посмотрев на ответ сервера:

Закодированное число	Ответ
0	`Ошибка: инвитик староват!!`
−1	`Ошибка: инвитик староват!!`
10000	`Ошибка: инвитик староват!!`
6185084	`Error: Hi! У вас инвит недействительный! Good bye!`
99999999	`Вы что вообще заслали? Ну-ка быстренько исправляемся!`

Ошибки бывают трёх родов:

1. код уже использован и староват (есть новее),
2. код уже использован (предположим, что это означает, что кодов, новее нет),
3. код неизвестен серверу (предположим, что это означает, что число инвайтов меньше, чем порядковый номер засланного *инвитика*).

То есть, сервер сообщает достаточное информации, чтобы можно было найти неиспользованный код бинарным поиском за относительно небольшое число попыток (если, конечно, озвученные выше предположения верны).

1. Пробуем заслать какой-нибудь случайный код, например, 10000. Получаем ошибку первого рода. Значит, надо брать больше.
2. Пробуем заслать код с номером *вдвое* больше: 20000. Снова получаем ошибку первого рода.
3. Повторяем шаг 2, пока не получим другую ошибку. Узнаём, что свободные *инвитики* точно лежат где-то между 5120000 и 10240000.
4. Пробуем середину интервала [5120000, 10240000]: `10240000 - (10240000 // 4)`, число 7680000. Получаем ошибку второго рода — новее *инвитиков* нет. Это позволяет нам уточнить верхнюю границу интервала: [5120000; 7680000].
5. Повторяем шаг 4 и уточняем интервал, пока (осознанно или случайно) не наткнёмся на свободный код.

Ура, мы внутри. Заходим в комнату, слушаем занимательный диалог: персонажи сперва обсуждают бытовые проблемы, а затем один из гостей «Сельского клуба» диктует другому флаг.

Флаг: **ugra_come_for_tasks_stay_for_f05da37e97c3**