Задания, решения и критерии заключительного этапа олимпиады школьников Ugra CTF School 2022

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-2022-school.

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 800 и более баллов.
- Призёрами олимпиады II степени были признаны участники, набравшие 300 баллов.
- Призёрами олимпиады III степени были признаны участники, набравшие 250 баллов.

Задания и решения

CACAPTCHA

Программирование, 250 очков.

> CACAPTCHA *(Completely Automated Completely Automated Public Turing test to tell Computers and Humans Apart)* —полностью автоматизированный полностью автоматизированный публичный тест Тюринга для различения компьютеров и людей.
>
> Совершенно понятно, что данный дважды полностью автоматизированный тест совсем не рассчитан на прохождение человеком. Сможете заставить свой компьютер пройти его?
>
> *https://cacaptcha.s.2022.ugractf.ru/99c7dd88de68198f/*

Решение

Дают сайт. На сайте просят поиграть в игру, которая состоит из 1337 уровней. В каждом уровне игры даны правила, которым запрос должен удовлетворять на любом уровне игры:

Method | Post |
Host | *В адресной строке не видно разве?* |
Content-Type | application/x-www-form-urlencoded |
Поле в FORM DATA | `data` |
Содержимое поля | *Произвольное.* |
Доп. ограничения | Запрос не должен пресекаться фильтром HAX()RW4LL |

А также конфигурация WAF-фильтра *HAX()RW4LL,* изменяющаяся от уровня к уровню:

```
(ALL (MUST-CONTAIN :REGEX /\(II\).*/)
     (MUST-CONTAIN :REGEX /.*"Yes, and I wanted to ask you----".*/))
```

Выходит, нам нужно отправить POST-запрос, содержащий в поле `data` некую строку. Однако, строка не может быть совсем какой угодно: она должна удовлетворять правилам фильтра *HAX()RW4LL*. Разобравшись с синтаксисом, которым описываются эти правила, приходим к выводу: в поле `data` должен быть такой текст, какой будет содержать в себе подстроки, удовлетворяющие данным регулярным выражениям.

Из правил самой игры также следует, что мы можем ошибиться три раза —тогда уровень «ламерства» перевалит за отметку 75%, и придётся начинать всё сначала.

Задача нехитрая, давайте напишем решалку. План такой:

1. Парсить страницу с игрой, извлекая из неё нужные нам данные: номер текущего уровня, наше «ламерство», список регулярных выражений, а также флаг, если таковой есть.
2. Генерировать строку для поля `data` по данным регулярным выражением.
3. Отправлять POST-запрос.
4. Возвращаться к шагу (1), пока мы не получим флаг.

Чтобы генерировать строку по данному регулярному выражению, можно, конечно, написать свою функцию, которая бы парсила эти выражения на правила и подбирала бы подходящие символы. Но у нас на всё про всё пять часов, а ещё ведь надо и роутер починить, и в крестики-нолики выиграть, и в Питер скататься··· Поэтому воспользуемся готовым пакетом exrex, который как раз умеет то, что нам надо:

```
>>> exrex.getone('\d{4}-\d{4}-\d{4}-[0-9]{4}')
'3096-7886-2834-5671'
```

Парсить страницу попробуем тоже регулярными выражениями, раз такая тема! Запросы будем отправлять стандартной библиотекой `requests`.

> Стоит, однако, помнить, что регулярными выражениями можно обрабатывать не всё: HTML не является регулярным языком, поскольку, как минимум, содержит открывающие и закрывающие теги, которые создают уровни вложенности. См. подробное объяснение проблемы: You can’ t parse HTML with regex.

Приступим:

```python
import exrex    # генератор строк под регулярные выражения
import re       # библиотека для работы с р. в.
import requests
import html     # пригодится позже

TOKEN = '99c7dd88de68198f'
URL = f'https://cacaptcha.s.2022.ugractf.ru/{TOKEN}/game'
```

Напишем функцию, которая будет принимать текст страницы и находить в нём интересующие нас вещи:

```python
def parse_page(page: str) -> dict:
    level = re.search(r'Уровень: ([0-9]+)/', page)
    lamerness = re.search(r'L4MER L3VEL: ([0-9]+)%', page)
    flag = re.search(r'(ugra_[A-Za-z0-9_]+)', page)
    if level and lamerness:
        level = int(level.groups()[0])
        lamerness = int(lamerness.groups()[0])

    regexes_raw = re.findall(r':REGEX /(.*)/\)', page)
    regexes = []
    for regex in regexes_raw:
        regexes.append(html.unescape(regex))

    return {
        'level': level,
        'lamerness': lamerness,
        'regexes': regexes,
        'flag': flag.groups()[0] if flag else None
    }
```

Затем напишем функцию, которая найдёт подходящее решение. Чтобы удовлетворить правилу (ALL (MUST-CONTAIN ...) ...), достаточно взять все данные выражения, склеить в одно большое простой конкатенацией и скормить функции exrex.getone. Но есть один нюанс. В регулярных выражениях есть правила, указывающие, с чего должна начинаться и чем должна заканчиваться строка — выражениям ^foo и foo$ соответственно подойут только строки вида foo... и ...foo. Учтём этот нюанс, отсортировав список данных выражений по вот такому вот правилу:

```
def sort_regexes(x: str) -> int:
    if '^' in x:
        return -1 # должны быть в начале списка
    elif '$' in x:
        return 1  # должны быть в конце
    else:
        return 0  # иначе всё равно
```

Теперь сам поиск решения —всего три строчки:

```
def find_solution(regexes: list[str], **_) -> str:
    regexes = sorted(regexes, key=sort_regexes)
    the_regex = ''.join(regexes)
    return exrex.getone(the_regex)
```

Осталось написать цикл, в котором будем засылать решения и получать новые данные:

```
if __name__ == "__main__":
    session = requests.Session()
    state = parse_page(session.get(URL).text)
    while True:
        print(state['level'], end=' ', flush=True)  # показываем прогресс
        solution = find_solution(**state)
        session.post(URL, data={'data': solution})  # засылаем ответ
        state = parse_page(session.get(URL).text)
        if state['lamerness'] > 0 or state['flag']: # в нештатной ситуации останавливаемся и смотрим, что там пришло
            print(state)
            break
```

Запустим скрипт и посмотрим, какой же был флаг:

```
cacaptcha § python solver.py
1 2 3 4 5 6 7 8 9 10 11 12 13 [...] 1334 1335 1336 {'level': 1337, 'lamerness': 0, 'regexes': [], 'flag': 'ugra_did_you_know
```

Ура! В общем-то, не смотря на пугающие тексты на крутой странице крутого хакера о конечных автоматах и грамматиках, получилось всё не так уж и сложно –и совсем не страшно.

Скрипт solver.py.

Флаг: **ugra_did_you_know_that_captcha_is_a_trademark_32718c98452a0094**

Крестики-нолики

Реверс-инжиниринг, 300 очков.

> Ушлые издатели выпустили ремейк крестиков-ноликов —теперь играть можно прямо онлайн в Интернете. Правда, пока что в сети доступен только ознакомительный клиент для игры: за полную версию авторы просят пожертвования. По крайней мере вы выпросили у знакомого лицензионный ключ от его копии. Сможете обойти защиту?
>
> *tictactoe-client*
>
> *f846360dec3d7676*

Решение

Запускаем пробную версию игры. После прочтённого предупреждения об урезанных возможностях заходим в игру и играем пробную партию. Выясняем, что играть получится только за нолики.

Попытки проанализировать исполняемый файл далеко не приведут –всё приложение написано на Haskell, языке программирования, плохо поддающимся декомпиляции. Однако игра, как заявлено, сетевая, и можно изучить протокол взаимодействия с сервером.

Запустим Wireshark и клиент игры. Первое сообщение от клиента серверу выглядит так:

```
0000   00 00 00 00 00 00 cc 13 22 00 00 00 00 00 00 00   ........".......
0010   64                                                d
```

К сожалению, протокол обмена с сервером –бинарный, и не сразу получится разобраться в смысле сообщений. В таких случаях протокол можно анализировать, запуская клиент много раз и выполняя различные действия. Таким образом можно заметить, каким сообщениям соответствуют команды от клиента, и где в них хранятся параметры.

Первое сообщение остаётся одинаковым при каждом подключении –предположим, что это некоторое приветствие (*handshake*). На него сервер отвечает таким же сообщениям –во многих протоколах в таких приветствиях содержатся версии клиента и сервера, что позволяет им договориться об "общем языке" .

Следующее сообщение от клиента:

```
0000   01 00 09 74 72 69 61 6c 5f 6b 65 79              ...trial_key
```

Видим, что здесь передаётся текстом некий ключ длиной 9 символов. Сразу перед ним видим байт `09` –скорее всего, это длина строки.

Сервер отвечает:

```
0000   02                                                .
```

Видимо, подтверждая лицензию, и сразу после этого:

```
0000   08 00 1f d0 94 d0 be d0 b1 d1 80 d0 be 20 d0 bf   ............. ..
0010   d0 be d0 b6 d0 b0 d0 bb d0 be d0 b2 d0 b0 d1 82   ................
0020   d1 8c                                             ..
```

После этого клиент показывает приглашение ("Добро пожаловать") и предлагает ввести команду.

Попробуем сыграть за крестики: `new X`. На экране появляется ошибка, а новых сообщений не передаётся. Делаем вывод, что на стороне клиента функция игры за крестики отключена полностью.

Если же выбрать нолики (`new O`), то клиент посылает сообщение:

```
0000   05 01                                             ..
```

На что сервер отвечает:

```
0000   06 01 01                                          ...
```

И затем сразу:

```
0000   08 00 26 d0 a2 d0 b5 d0 ba d1 83 d1 89 d0 b0 d1   ..&.............
0010   8f 20 d0 b4 d0 be d1 81 d0 ba d0 b0 3a 0a 5f 5f   . ..........:.__
0020   5f 0a 5f 58 5f 0a 5f 5f 5f                        _._X_.___
```

На экране при этом появляются несколько сообщений:

```
Противник сделал ход: (1, 1)
Текущая доска:

---
_X_
---
```

Попробуем сопоставить сообщения от сервера консольному выводу. Поле игры в крестики-нолики –двухмерная доска, точку на которой можно описать двумя числами от 1 до 3 (или от 0 до 2). В первом пакете мы видим байты `01 01`, которые похожи на `(1, 1)` из консольного вывода. Позже мы попробуем подтвердить эту теорию. Второе же сообщение содержит часть вывода, который был показан на экране –текущую карту поля. Можно предположить, что в таких сообщениях содержатся строки, которые напрямую выводятся клиентом на экран. Такие текстовые строки часто кодируются UTF-8 –проверим эту догадку, раскодировав всё сообщение в Python:

```
>>> str = "080026d0a2d0b5d0bad183d189d0b0d18f20d0b4d0bed181d0bad0b03a0a5f5f5f0a5f585f0a5f5f5f"
>>> bytes.fromhex(str).decode("utf-8")
'\x08\x00&Текущая доска:\n___\n_X_\n___'
```

Отметим, что первый байт сообщения (08) мы уже встречали в сообщении раньше. Также отметим, что в сообщении после 08 идёт 00 26. В сетевых пакетах преимущественно применяется порядок байт *big endian*, в котором старшие байты сообщения хранятся в начале. Заметим, что 26 соответствует 38 –длине дальнейшей строки в байтах. В предыдущем сообщении после 08 шли байты 00 1f, а затем также 31 (то есть 1f в шестнадцатеричной системе счисления) байт. Раскодируем и его:

```
>>> str = "d094d0bed0b1d180d0be20d0bfd0bed0b6d0b0d0bbd0bed0b2d0b0d182d18c"
>>> bytes.fromhex(str).decode("utf-8")
'Добро пожаловать'
```

Похоже, сообщения, начинающиеся с 08 –это текстовые сообщения, которые необходимо вывести на экран. Они состоят из длины строки (2 байта в big endian) и самой строки. Поскольку сообщения в протоколах обычно обладают схожей структурой, можно предположить, что первый байт сообщения означает его тип. Лицензионный ключ из сообщения 01 кодировался схожим образом –два байта размера, и затем сама строка.

Попробуем сделать ход в точку (2, 0). Клиент посылает сообщение:

```
0000   06 02 00                                          ...
```

Сравнивая с предыдущим сообщением на 06, подтверждаем нашу догадку –в сообщениях типа 06 передаётся ход клиента или сервера. Следующие два байта кодируют координату хода по X и Y. Сервер на это отвечает:

```
0000   06 02 01                                          ...

0000   08 00 26 d0 a2 d0 b5 d0 ba d1 83 d1 89 d0 b0 d1   ..&.............
0010   8f 20 d0 b4 d0 be d1 81 d0 ba d0 b0 3a 0a 5f 5f   . ..........:.__
0020   4f 0a 5f 58 58 0a 5f 5f 5f                        O._XX.___
```

А на экране мы видим:

```
Противник сделал ход: (2, 1)
Текущая доска:
__O
_XX
---
```

Попробуем сходить на уже занятую клетку (1, 1). Сервер отвечает:

```
0000   04 03                                             ..
```

А на экране мы видим:

```
% Ошибка: IllegalTurn
```

Что показывает нам, что сообщение 04 –ошибка, а 03 –возможно, её код.

Доиграем партию и посмотрим, как оканчивается игра. После очередного нашего хода компьютер занимает последнюю ячейку. После этого приходит такое сообщение:

```
0000   07 00                                             ..
```

А на экране видим:

```
Игра закончена, победитель: ничья
```

Попробуем проиграть и посмотрим, что изменится (выиграть за нолики, кажется, не получится). В конце игры получаем:

```
0000   07 01 00                                          ...
```

Похоже, сообщение 07 означает конец игры, а следующие байты кодируют победителя (пока непонятно, как).

Всей полученной информации достаточно, чтобы попробовать написать собственный клиент игры с урезанным функционалом. В первой итерации приветствие и лицензионный ключ мы отправим,

как есть. Новую игру также начнём известным пакетом (за нолики), а генерировать сами будем только пакеты с текущим ходом. Такой клиент будет успешно работать!

Попробуем сыграть за крестики. В сообщении начала игры (05) передаётся один байт данных (01). Попробуем заменить его на 00, предположив, что это код игры за крестики. Получаем сообщение:

```
0000   04 02                                             ..
```

Мы уже знаем, что 04 –это сообщение об ошибке, но пока непонятно, в чём она заключается.

Однако, в самом задании нам был также предоставлен лицензионный ключ. Попробуем использовать его вместо найденного trial_key. В этом случае на запрос 05 00 сервер отвечает уже известным нам сообщением 02, которым он также отвечал на лицензионный ключ. Попробуем сделать ход – сервер воспринимает его, и отвечает так же, как мы наблюдали до этого!

В конце игры сервер пришлёт новое текстовое сообщение:

```
0000   08 00 80 d0 9f d0 be d0 b4 d0 b0 d1 80 d0 be d1   ................
0010   87 d0 bd d1 8b d0 b9 20 d0 ba d0 be d0 b4 20 d0   ....... ...... .
0020   bd d0 b0 20 d0 b4 d1 80 d1 83 d0 b3 d0 b8 d0 b5   ... ............
0030   20 d0 b8 d0 b3 d1 80 d1 8b 20 d0 bd d0 b0 d1 88    ........ ......
0040   d0 b5 d0 b9 20 d0 ba d0 be d0 bc d0 bf d0 b0 d0   .... ...........
0050   bd d0 b8 d0 b8 3a 20 75 67 72 61 5f 74 69 63 5f   .....: ugra_tic_
0060   74 61 63 5f 72 65 76 65 72 73 65 5f 74 61 6b 65   tac_reverse_take
0070   73 5f 74 69 6d 65 5f 32 31 30 37 63 33 65 30 34   s_time_2107c3e04
0080   62 33 62                                          b3b
```

Раскодируем его:

```
Подарочный код на другие игры нашей компании: ugra_tic_tac_reverse_takes_time_2107c3e04b3b
```

Флаг: **ugra_tic_tac_reverse_takes_time_2107c3e04b3b**

Лестница

Сетевая разведка, 300 очков.

> Как всегда, только один вопрос: *где это?*
>
> При решении вас спросят одноразовые коды или, как их *тогда* было принято называть, шифры —у вас их есть 8 штук.
>
> *https://stairwell.s.2022.ugractf.ru/b005d1fe6547a821/*

Решение

Мы наблюдаем вход ~~в подъезд~~ конечно же, на лестницу где-то в Петербурге с характерной петербургской деталью —табличкой с указанием, на каком этаже какие квартиры (во многих старых домах, особенно в центре, где квартиры объединялись и разделялись, нумерация бывает совершенно непредсказуемой). Табличек висит аж две: более старая и более современная. Дверь такая хорошая, добротная, явно стоит там не менее 60 лет. (После 1960, при Хрущёве и далее, таких уже не делали.)

Немного поискав, можно обнаружить сайт dominfospb.ru. Мы знаем из фотографии, что ~~подъезд~~ лестница находится в доме 33. Поиск на сайте по строке «33» в адресе даёт порядка 350 домов. В них есть немного лишних (дома с номерами 133, например).

Соберём для каждого такого дома страничку с информацией. Для фильтрации будем использовать минимальную и максимальную этажность (мы знаем, что этажей на нашей лестнице пять), а также год постройки (чтобы отсечь массово строившиеся после 1960 пятиэтажки).

Можем преобразовывать страницы в текст и вычленять данные таким регулярным выражением: `Адрес дома:\xa0(.*?)Год постройки:\xa0(.*?).*Наибольшее количество этажей, ед.:\xa0(.*?)\s*Наименьшее количество этажей, ед.:\xa0(\d*)`

До фильтрации:

```
...
[('г. Санкт-Петербург, ул. Бестужевская, д. 33, к. 1, лит. А', '1972', '14', '14')]
[('г. Санкт-Петербург, пр-кт Лермонтовский, д. 33, лит. А', '1879', '5', '4')]
...
```

После фильтрации:

```
г. Санкт-Петербург, пр-кт Большой П.С., д. 33а, лит. А
г. Санкт-Петербург, пр-кт Большой Сампсониевский, д. 33/1, лит. А
г. Санкт-Петербург, пр-кт Костромской, д. 33 литер А
г. Санкт-Петербург, пр-кт Лиговский, д. 33, к. 35
г. Санкт-Петербург, пр-кт Малый В.О., д. 33, лит. А
г. Санкт-Петербург, пр-кт Обуховской Обороны, д. 33, к. 1
г. Санкт-Петербург, пр-кт Обуховской Обороны, д. 33, к. 2
г. Санкт-Петербург, ул. 6-я Советская, д. 33
г. Санкт-Петербург, ул. 8-я Советская, д. 33
г. Санкт-Петербург, ул. Блохина, д. 33, лит. А
г. Санкт-Петербург, ул. Большая Пороховская, д. 33, лит.А
г. Санкт-Петербург, ул. Ждановская, д. 33А
г. Санкт-Петербург, ул. Жуковского, д. 33, стр. А
г. Санкт-Петербург, ул. Зайцева, д. 33, лит. А
г. Санкт-Петербург, ул. Итальянская, д. 33, лит. А
г. Санкт-Петербург, ул. Можайская, д. 33, к. А
г. Санкт-Петербург, ул. Некрасова, д. 33
г. Санкт-Петербург, ул. Профессора Попова, д. 33, лит. А
г. Санкт-Петербург, ул. Разъезжая, д. 33
г. Санкт-Петербург, ул. Розенштейна, д. 33, лит. А
г. Санкт-Петербург, ул. Рылеева, д. 33
г. Санкт-Петербург, ул. Седова, д. 33
г. Санкт-Петербург, ул. Съезжинская, д. 33, лит. А
```

Ещё можно заметить торчащее на указателе «ULITSA» и отбросить проспекты. Остаётся уже вполне обозримое количество домов. Можно было бы обойти все дома, например, на Панорамах, но это достаточно трудоёмко.

У дома есть одна очень редкая особенность: квартира номер 4 находится аж на пятом этаже. Обычно на этаж каждой лестницы приходится всё-таки хотя бы по две квартиры (чаще всего три-четыре — а во многих домах и больше). Здесь же на этаж приходится по одной квартире, а на первом этаже ничего нет вовсе (хотя в прежние времена всё-таки было —со странным номером 19).

Ещё один источник информации об объектах недвижимости —Росреестр. Там, в частности, можно узнать по номеру квартиры, на каком она этаже (а ещё много всего интересного). Официальный поиск Росреестра — «Справочная информация по объектам недвижимости в режиме online» — позволяет после ввода капчи и адреса в относительно свободной форме (лишь бы поиск отработал) открыть информацию о конкретной квартире.

Перебором находим, что такая квартира есть в доме **улица Жуковского, 33** (в Росреестре почему-то записан как **33 литер А**). Смотрим на Яндексе координаты входа: **59.935800, 30.356483**. Можно с Панорам подтвердить, что табличка действительно та (а номер дома успел переехать с того места, где он оказался так кстати).

Не забываем, что нас просят указать долготу относительно Пулковского меридиана. Согласно Википедии, это 30.326053° восточной долготы —это число надо вычесть из долготы относительно общепринятого в современности нулевого меридиана. Так что в форму надо вводить ответ **59.935800, 0.030430**.

Флаг: **ugra_and_she_is_buying_one_to_heaven_62a92f831b14**

Бонус: карта со всеми попытками сдать задание. Последствия некорректного отсчёта координат от Пулковского меридиана были скорректированы —такие координаты были приведены, насколько это возможно, к правдоподобным.

Случай в парикмахерской

Форензика, 200 очков.

> Наш агент стригся в парикмахерской и услышал разговор, доносившийся откуда-то из вентиляции:
>
> — Как там это делать? Вай-фай какой?

— Вот этот. Пароль — цифры подряд, как наш телефон: восемь, сорок с···

—Да знаю я, не ори ты! На весь Липецк разорался тут. Восемь···а-а-а-га···та-та, та-та, та-та. Так, подключилось, что теперь?

— Открывай сайт, ищи там урок и прикладывай реферат. Давай быстрей, опаздываю уже!

—Открыть сайт, найти там урок и приложить файл «Реферат». Отправить. Ну всё, беги.

—Спасибо! Неужели я наконец-то буду первый!

Кроме этого, он *кое-что ещё* подслушал, но не ушами.

overheard.cap.gz

Решение

Подслушанное *кое-что ещё* — ни что иное как перехват данных, звучавших в эфире вайфай-сети.

Из легенды и обрывков речи мы знаем, что её пароль — номер городского телефона в Липецке, записанный, начиная с восьмёрки, после которой следует код города (номер без кода был бы слишком коротким для пароля — 6 символов при минимальных 8). Иначе говоря, пароль имеет вид `84742??????`. Неизвестны 6 цифр, это всего миллион вариантов.

Для анализа трафика беспроводных сетей существует набор утилит под общим названием *aircrack-ng*. Главная из них, собственно *aircrack-ng*, позволяет для сетей с WPA2 при наличии перехваченной сессии аутентификации («четырёхстороннее рукопожатие» по протоколу EAPOL) перебирать возможные пароли со скоростью порядка нескольких тысяч вариантов в секунду (то есть нескольких миллионов в час). Мало-мальски хорошо защищённую сеть так не сломать, однако совсем уж ненадёжный —или известный частично — пароль таким образом вполне можно подобрать.

Создадим словарь возможных вариантов и запустим перебор (выберем ту сеть, для которой у нас есть все данные).

```
$ for i in {84742000000..84742999999}; do echo $i >> phones.dict; done
$ aircrack-ng -w phones.dict *.cap

      [00:03:04] 769112/1000000 keys tested (4247.36 k/s)

      Time left: 54 seconds                                     76.91%

                        KEY FOUND! [ 84742723793 ]


      Master Key     : 02 4D 38 20 4B 64 22 D5 0B 0B 79 F2 93 41 0E 66
                       3B 9A 57 64 26 E1 68 9E A5 38 9E 0B 6D 2F 4E 44

      Transient Key  : 33 D5 DB 7E D7 4A C7 00 00 00 00 00 00 00 00 00
                       00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                       00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                       00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

      EAPOL HMAC     : C0 01 02 98 FF 24 53 FD A0 8E 33 8A F7 54 C5 2F
```

Ближе к концу перебор нашёл пароль —`84742723793`. Что теперь? Оказывается, зная пароль сети и умудрившись перехватить криптографический обмен, можно расшифровать трафик! Сделать это позволит входящая в тот же набор утилита *airdecap-ng*. Укажем ей известный нам теперь пароль, а также идентификаторы точки доступа.

```
$ airdecap-ng -p 84742723793 -b 10:62:EB:8D:2C:A5 -e DIR-825 *.cap
Total number of stations seen            2
Total number of packets read          6381
Total number of WEP data packets         0
Total number of WPA data packets      2596
Number of plaintext data packets         1
Number of decrypted WEP  packets         0
```

```
Number of corrupted WEP  packets         0
Number of decrypted WPA  packets      2572
Number of bad TKIP (WPA) packets         0
Number of bad CCMP (WPA) packets         0
```

Рядом появится ещё один .cap-файл с уже расшифрованными пакетами. Откроем его через wireshark.

Среди кучи служебного трафика, а также трафика по https, который не расшифровать никак, видим общение с неким весьма липецко-школьным на вид сайтом. Находим POST-запрос, в котором на сайт отправляется документ формата .docx. Извлекаем содержимое файла из соответствующего TCP-потока, отрезаем HTTP-заголовки и иную лишнюю информацию, открываем документ — и читаем реферат про флаг России, в котором можно заодно узнать флаг, интересующий нас в задании.

Флаг: **ugra_cracking_be_in_the_air_tonight_f7ec913ae430**

Свято место пусто не бывает

Веб-программирование, 50 очков.

> Вы пришли в гости, и тут же вас, зная о вашей технической подкованности, попросили посмотреть интернет, а то что-то не работает…Поможете?

https://sacredplace.s.2022.ugractf.ru/fbef8850e6e83c3b/

Решение

Домашние беспроводные точки доступа знамениты тем, что мало кто грамотно их настраивает.

Если перейти по ссылке —увидим страницу авторизации для некоего устройства DIR-825ACG1. Домашние беспроводные точки доступа также знамениты тем, что неприлично часто их оставляют со стандартными логином и паролем. Попробуем нагуглить:

#	Username	Password
1	admin	(blank)
2	admin	admin
3	(blank)	(blank)
4	(blank)	admin
5	admin	password
6	admin	1234
7	create in initial setup	create in initial setup
8	(blank)	created in initial setup
9	admin	root
10	user	user
11	Admin	leave blank
12	admin	refer to router label
13	admin	telus
14	root	(blank)
15	ipbbx	ipbbx
16	admin	Same as network key or WIFI key. Can be found on a

Рис. 1: Таблица «Топ-10 популярных паролей от роутера». Источник: https://www.192-168-1-1-ip.co/router/dlink/dir-825/9679/

Пробуем все варианты, начиная с первого: логин `admin`, пароль пустой. Увы, домашние беспроводные точки доступа ещё и знамениты тем, что качество их программного обеспечения оставляет желать лучшего: веб-страница убеждена, что свято место (поле ввода пароля) пусто не бывает (хотя на самом деле бывает, конечно):

```
<input name="password" required="">
```

Уберём атрибут `required` и попробуем-таки войти с пустым паролем. Получилось, видим таблицу с подключёнными клиентами, название одного из которых начинается с `ugra_`…

Флаг: **ugra_unless_it_is_not_really_that_sacred_000bded066ad72dd**