**Write-up**

**команда "С нечитаемым ником"**

**Task-based**

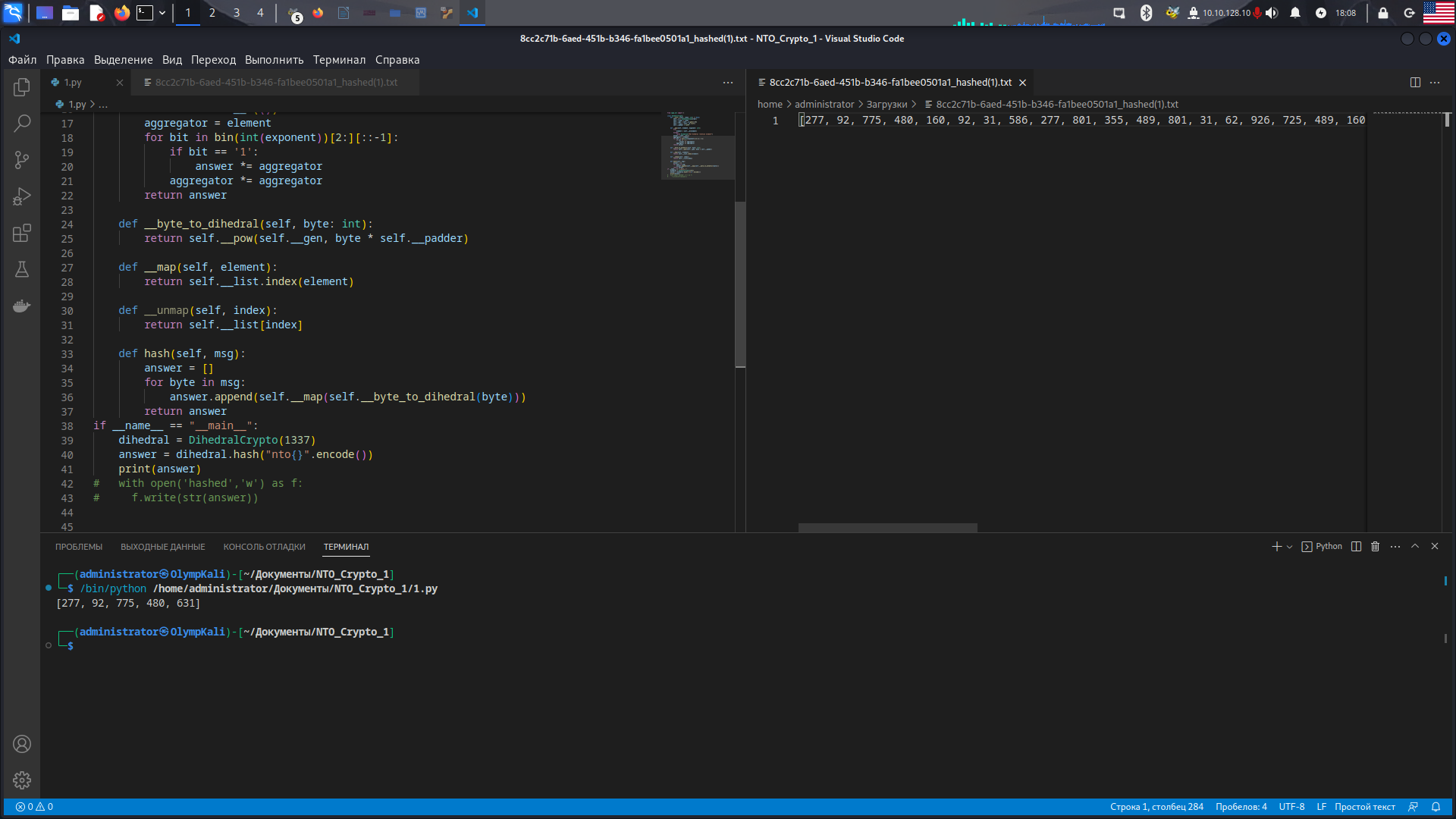
*Crypto 10*

*Флаг - nto{5tr4ng3\_gr0up\_5tr4ng3\_l0g\_and\_depressed\_kid\_zxc\_ghoul}*

Скачаем оба файла, которые нам предоставлены.

Откроем код (первый файл)

Проанализируем код и поймем, что флаг кодируется посимвольно, далее напишем в строчку <answer = dihedral.hash(flag)> вместо *flag* → “*nto{}“.encode()* (nto{ это начало нашего флага, а } конец). Посмотрим, что у нас вывелось и что находиться в hashed.txt (второй файл) (Скришнот 1). Мы заметили, что несколько цифр совпало, и у нас появилась идея как посторить решение, которое мы реализовали в crypto-1.py

Скриншот 1

Последовательность действий такая:

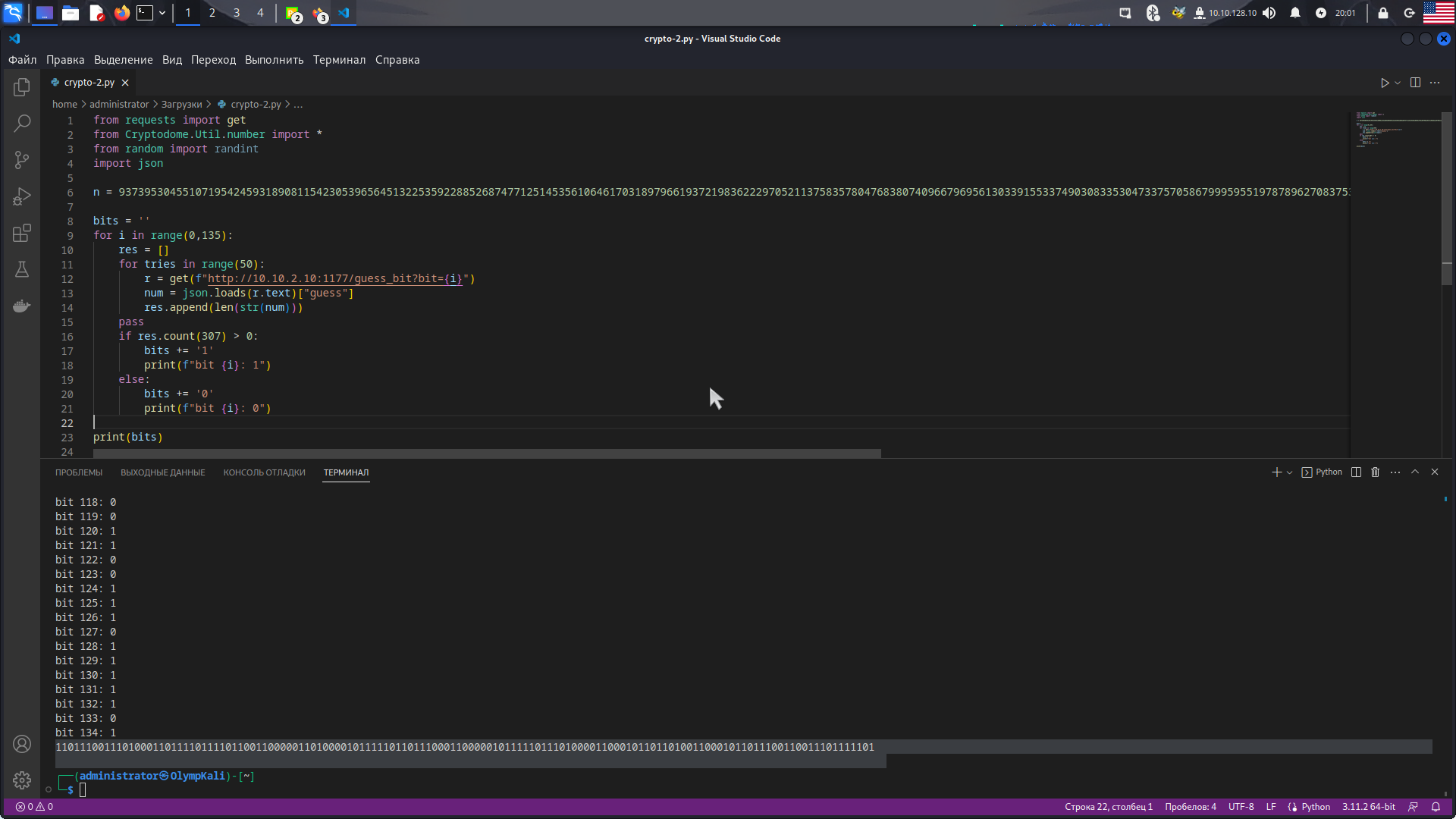
1. Строим таблицу соответствия символа и кода после шифрования символа
2. Расшифровываем флаг по таблице

# Код с решением - crypto-1.py

------------------------------------------------------------------------------------------------------------------------

*Crypto 20*

Перейдем по ссылке (<http://10.10.2.10:1177/>), и проанализируем код, и увидим, что нем строиться двоичная запись флага и потом по ней строиться число n.  
Напишем свой код, мы узнаем длину флага подбором максмального значения guess\_bit в браузере, получим 135 + 1 ( первый бит 0 ), также было замечено, что длина числа сгенерированного функцией pow либо 307, либо 308, а длина числа сгенерированного функцией randint фиксированно 308, чтобы удостовериться на сервер отправляется по 50 запросов, и если среди них есть число длиной 307, то оно точно сгенерировано функцией pow, а значит бит равен еденице, поэтому если среди полученных чисел, есть числа, длина которых равна 307, то добавляем 1, иначе 0.

В начало добавим 0 для успешного декодирования, получим: 0110111001110100011011110111101100110000011010000101111101101110001100000101111101110100001100010110110100110001011011100110011101111101

После декодирования в CyberChef получим флаг: *nto{0h\_n0\_t1m1ng}*

# Код с решением - crypto-2.py

------------------------------------------------------------------------------------------------------------------------

*Web 10*

*#flag — nto{w3bs0ck3ts\_plu5\_xx3\_1s\_l0v3}*

Мы перешли по ссылке ([http://10.10.2.10:](http://10.10.2.10:3002/)8080) и проанализировали сайт

1. Мы открыли Burp Suite
2. Введем программу (web-1.js) в консоль, первая часть которой позволит нам выводить в консоль расшифрованное сообщение от сервера
3. Вторая часть кода отправляет данные на сервер в формате XML, в который встроен XXE: XML external entity
4. По итогу мы прочитываем данные, которые включают в себя флаг. (Скриншот 1)



Скриншот 1

# Код с решением - web-1.js

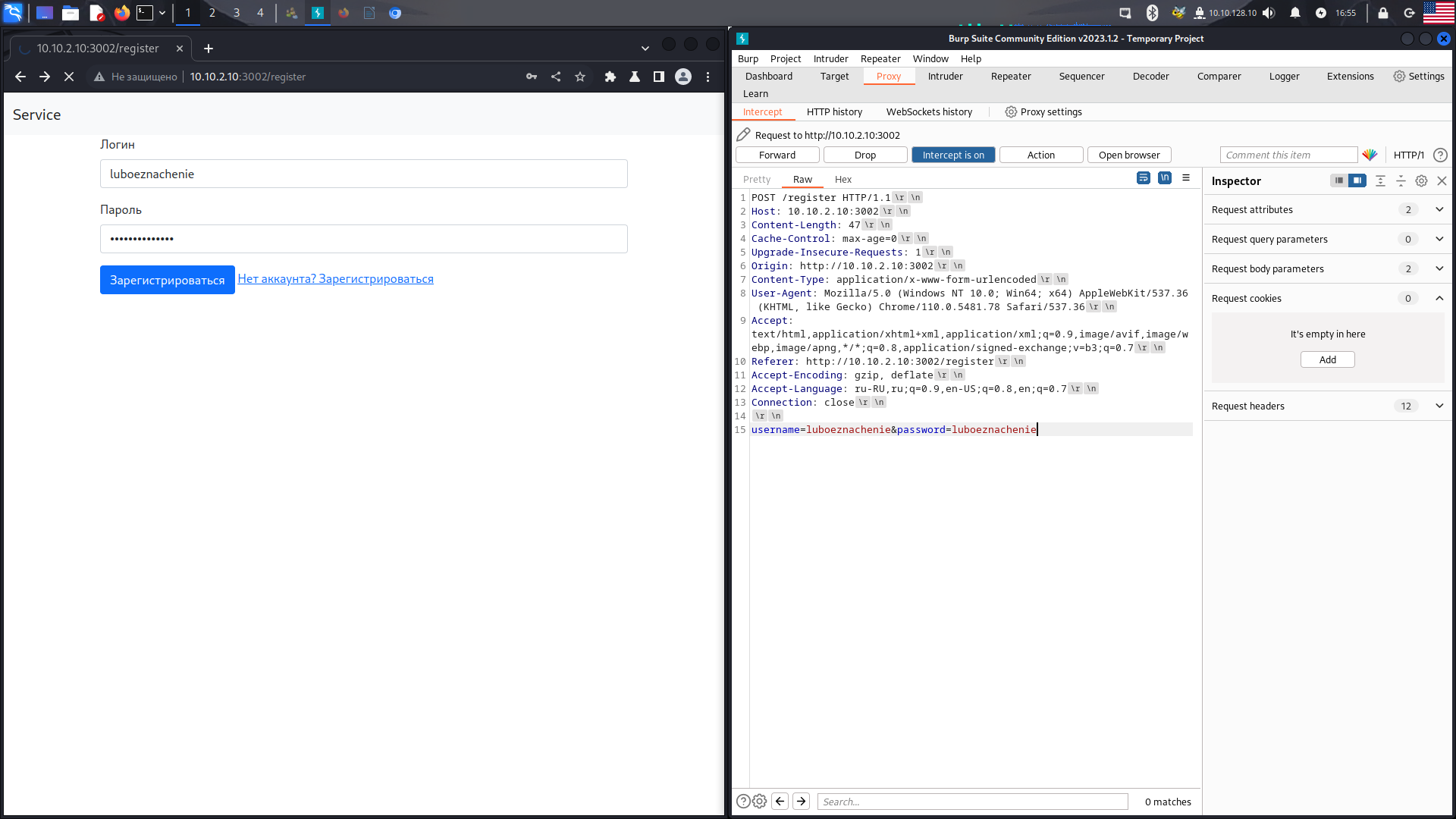
------------------------------------------------------------------------------------------------------------------------

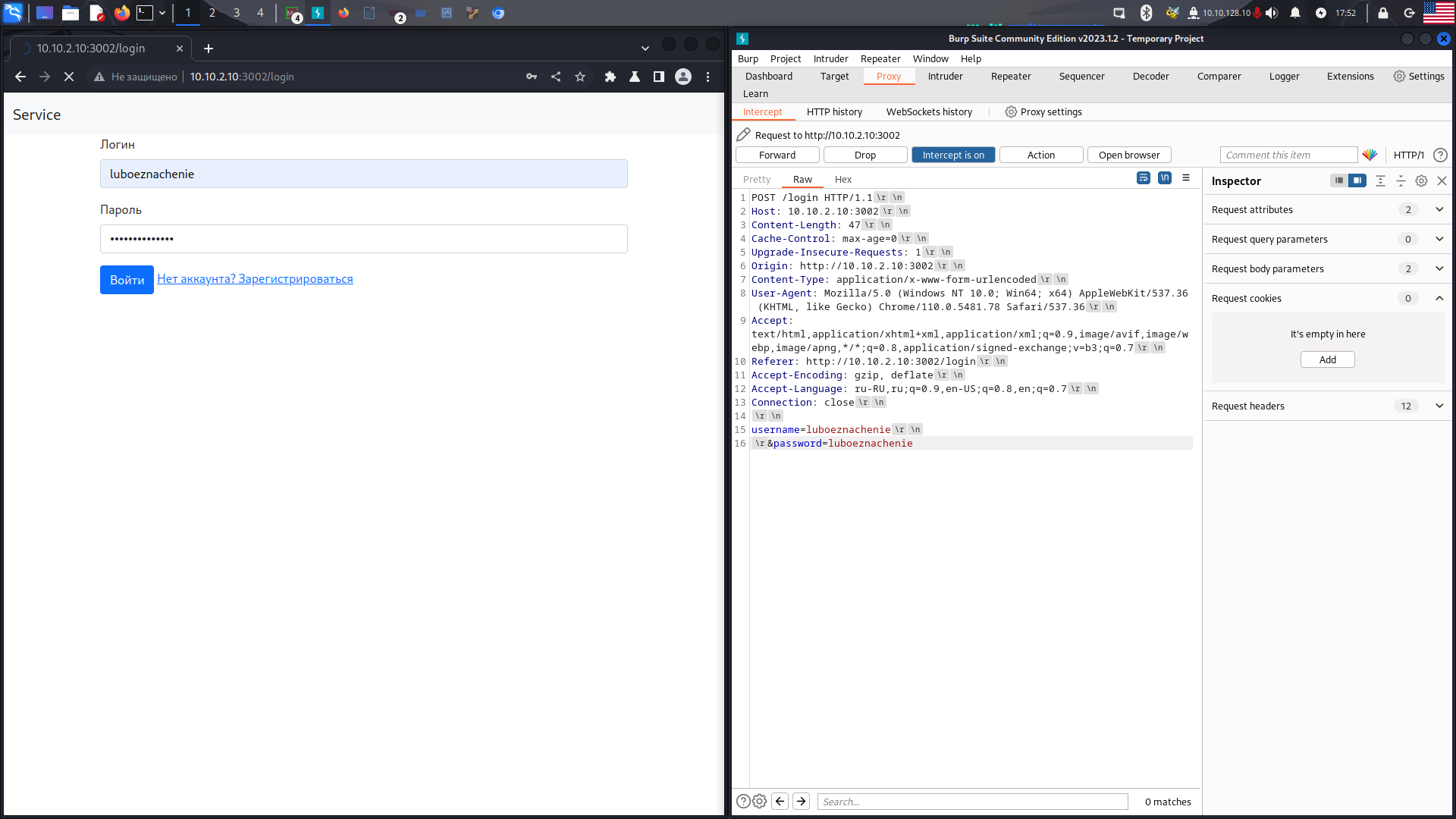
*Web 20*

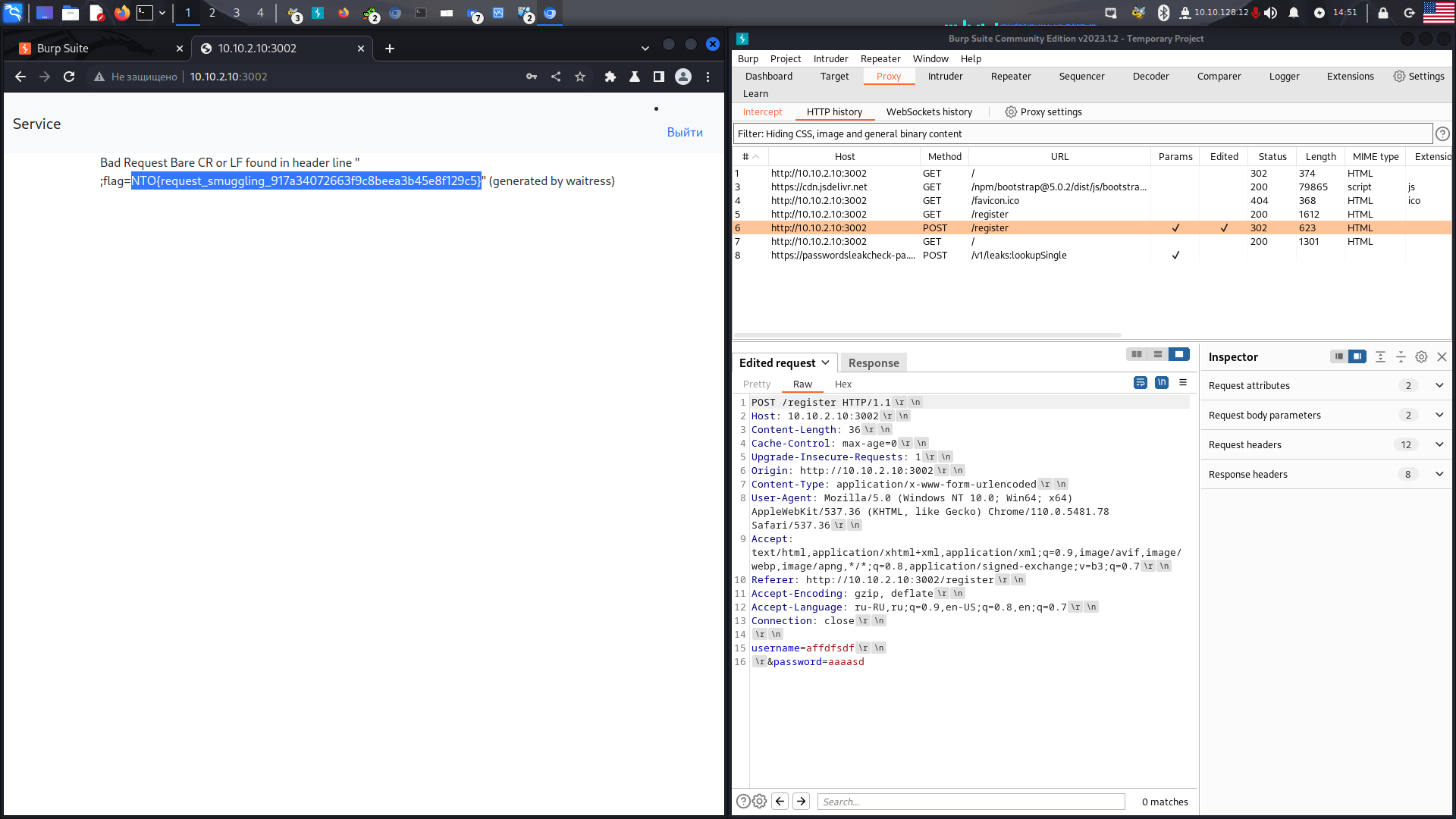
Проанализировав исходники нам стало понятно, что мы должны проэксплотировать уязвимость на втором сервисе ([http://10.10.2.10:3002](http://10.10.2.10:3002/)), а именно его систему регистрации.

1. Открываем Burp Suite, проходим во вкладку Proxy, включаем Intercept и открываем браузер
2. Проходим по ссылке данной выше и идём проходить регистрацию
3. В полях логин и пароль пишем любые значения (Скриншот 1)
4. Далее мы пытаемся произвести HTTP Host header attack (Скриншот 2)
5. Данная уязвимость позволяет нам получить флаг (Скриншот 3)

*Флаг: NTO{request\_smuggling\_917a34072663f9c8beea3b45e8f129c5}*

Скриншот 1

Скриншот 2

Скриншот 3

------------------------------------------------------------------------------------------------------------------------

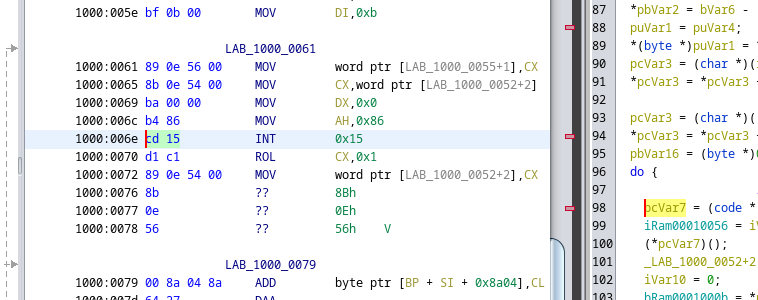
*Reverse 10*

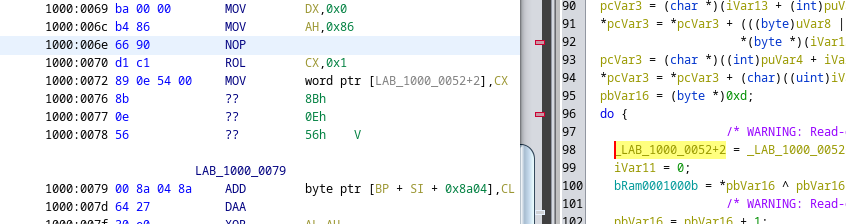
*Флаг: nto{h3ll0\_n3w\_5ch00l\_fr0m\_0ld!!}*

Откроем наш файл в Ghidra и проанализируем его. Закончив анализ, мы поняли, что

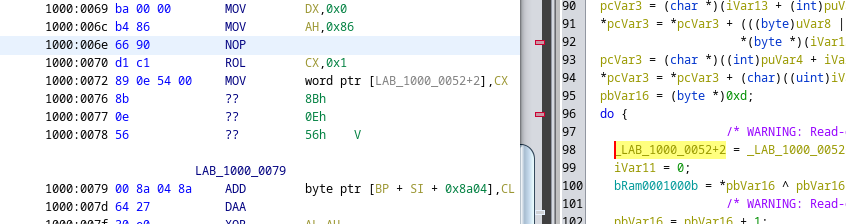
нужно пропатчить выделенную строку, заменив прерывание на NOP (Скриншот 1, 2).

Далее запустим пропатченный файл и получим флаг (Скриншот 3).

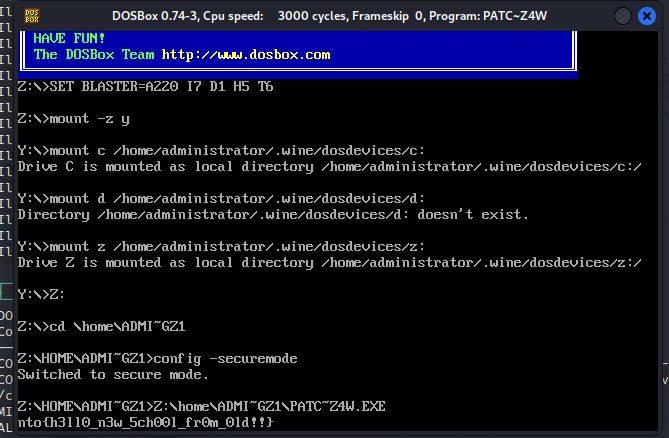
**



Скриншот 1



Скриншот 2

Скриншот 3