NOVA CTF 2023 WRITE UP

FIGHT FOR THE GET

In this CTF challenge, we need to exploit parameter pollution vulnerability in a web application. The goal is to retrieve the keys and values for a parameter by finding the sixth consecutive number generated by a random function and decoding morse code audio respectively and then retrieve the flag using keys and values obtained.

Parameter Pollution Vulnerability:

Parameter pollution is a vulnerability that arises when multiple values are assigned to the same parameter. In this vulnerability, an attacker can manipulate the values of a parameter to perform unauthorized actions. It can occur due to improper handling of user input, insufficient validation, or poor coding practices.

Finding the Keys and Values:

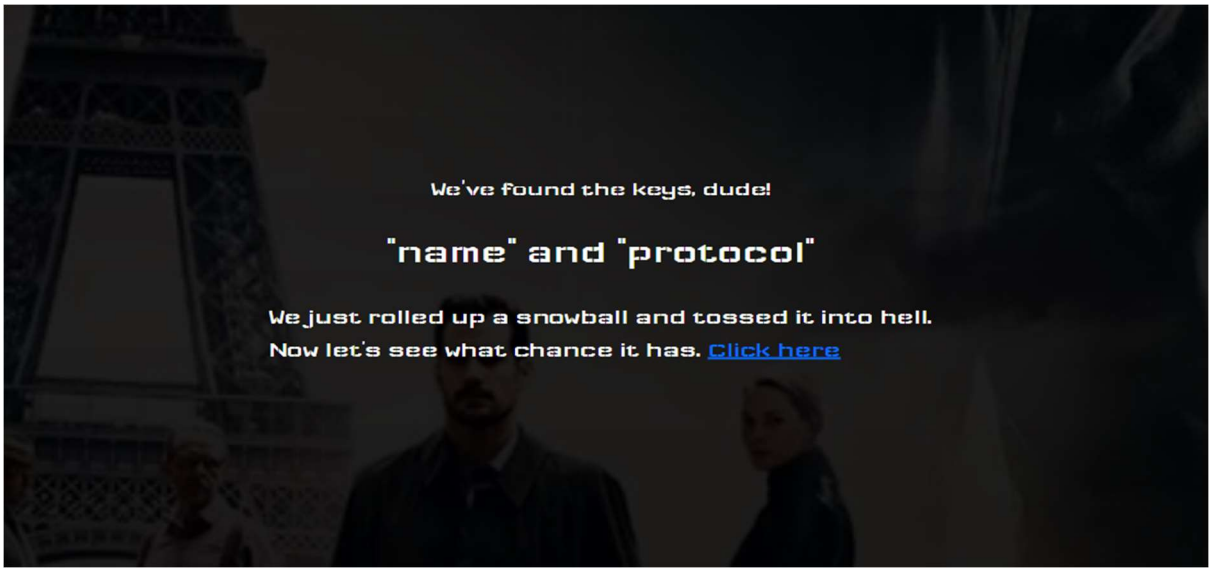
The first part of the challenge involves finding the keys. The web application generates random numbers that are used to retrieve the parameter keys. The task is to identify the sixth consecutive number generated by this random function. Once you have found the number, it can be used as a key to retrieve the corresponding parameter value.

For finding the random number, the following code can be used

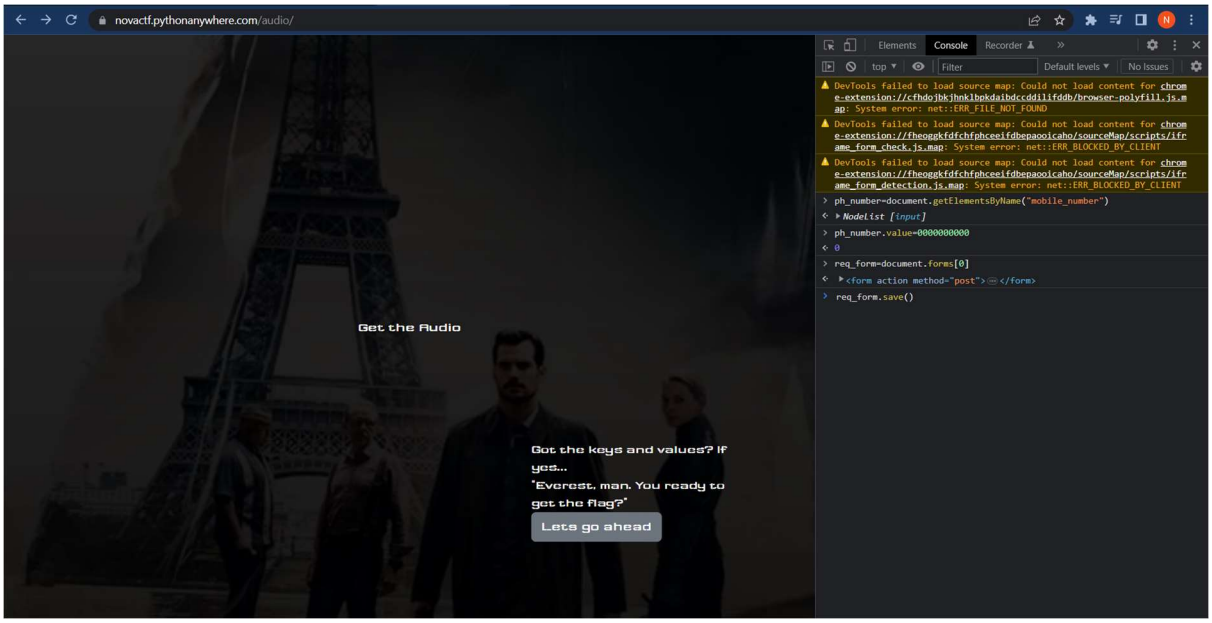
```
import z3
import struct
import sys
sequence = [0.9202030820029297 , 0.07831142293729076 ,
0.97881067034441135 ,0.27630837640608874 ,
0.8181456836734036]
sequence = sequence[::-1]
solver = z3.Solver()
se_state0, se_state1 = z3.BitVecs("se_state0 se_state1", 64)
num_of_prev_values=len(sequence)
for i in range(num_of_prev_values):
    se_s1 = se_state0
    se_s0 = se_state1
    se_state0 = se_s0
    se_s1 ^= se_s1 << 23
    se_s1 ^= z3.LShR(se_s1, 17)
```

```
    se_s1 ^= se_s0
    se_s1 ^= z3.LShR(se_s0, 26)
    se_state1 = se_s1
    float_64 = struct.pack("d", sequence[i] + 1)
    u_long_long_64 = struct.unpack("<Q", float_64)[0]
    mantissa = u_long_long_64 & ((1 << 52) - 1)
    solver.add(int(mantissa) == z3.LShR(se_state0, 12))
if solver.check() == z3.sat:
    model = solver.model()
    states = {}
    for state in model.decls():
        states[state.__str__()] = model[state]
    state0 = states["se_state0"].as_long()
    u_long_long_64 = (state0 >> 12) | 0x3FF0000000000000
    float_64 = struct.pack("<Q", u_long_long_64)
    req_value = struct.unpack("d", float_64)[0]
    req_value -= 1
    print(req_value)
```

Reference: https://github.com/PwnFunction/v8-randomness-predictor



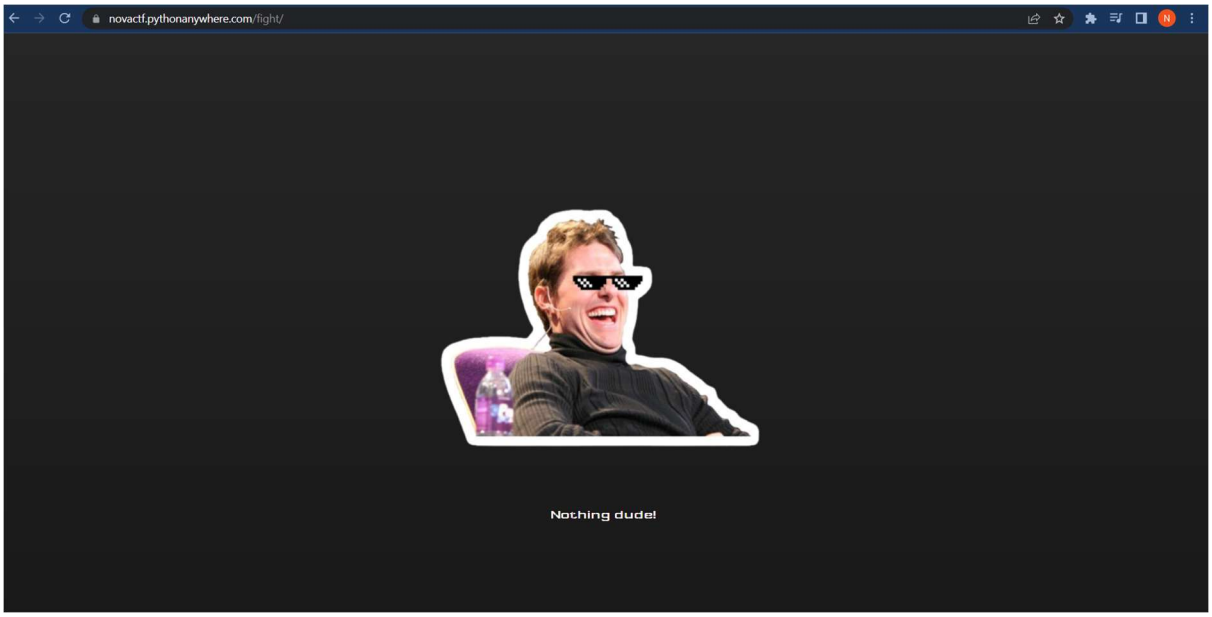
On clicking the given link, it takes to /audio URL. On looking into the elements, we can see the form with mobile_number input element. Set the mobile_number input to your whatsapp number and submit the form using java script.



You will receive an audio in WhatsApp.

The second part of the challenge involves decoding Morse code audio to retrieve the parameter value. The audio file contains a Morse code message that needs to be decoded. Morse code is a method of transmitting text information as a series of on-off tones, lights, or clicks. You can use any audio decoder or online tool to decode the Morse code.

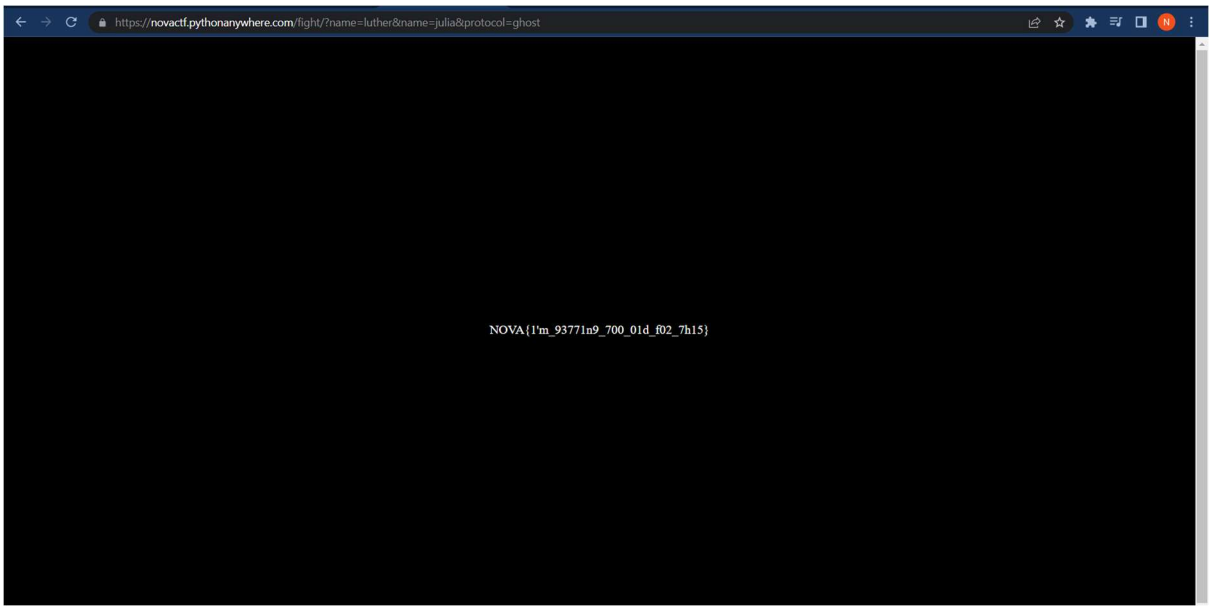
On decoding the audio using morse code decoder, we can get the values – luther, julia, ghost. Then , on clicking on the lets go ahead button, we can see the following page.



To complete the challenge, you will need to combine the two parts.

On using the keys and values we got, make the following get request.

https://novactf.pythonanywhere.com/fight/?name=luther&name=julia&protocol=ghost



In conclusion, this CTF challenge provides an opportunity to learn and practice exploiting parameter pollution vulnerability in a web application. It is important to note that parameter pollution is a real-world vulnerability that can have severe consequences if not properly mitigated.

And the hunt ends!