Rainbombash Adventure - Writeup

Author	Title	Category	Difficulty
barrythecanary	Rainbombash Adventure	re	-

competed with Fondue Overflow

Attachments

rainbombashadventure

Solution

We are given a Visual Novel game made in Ren'Py. Playing through the game it's a somewhat silly story till we get to the point where we have to help Rainbom find the fastest path to hit all the clouds and return.

Looking into the game folder we can also find the source code inside script.rpy. Overall It seems to be a traveling salesman problem (tsp) and to get the flag the program XORs a sha256 hash of the correct path with a byte array in the code.

```
flag = b""


def xor(target, key):
    out = [c ^ key[i % len(key)] for i, c in enumerate(target)]
    return bytearray(out)


def key_from_path(path):
    return hashlib.sha256(str(path).encode()).digest()


def check_path(path, enc_flag):
    global flag
    flag1 = xor(enc_flag, key_from_path(path))
    flag2 = xor(enc_flag, key_from_path(list(reversed(path))))
    if flag1.startswith(b"BtSCTF"):
        flag = flag1
        print(flag)
        flag = bytes(flag).replace(b"{", b"{{").decode('ascii')
        return True
    if flag2.startswith(b"BtSCTF"):
        flag = flag2
```

```
        print(flag)
        flag = bytes(flag).replace(b"{", b"{{").decode('ascii')
        return True
    return False


is_correct = check_path(nodes, bytearray(b'\xc2\x92\xf9\xf66\xe8
\xa5\xa6\x17\xb6mGE\xcfQ\x90Mk:\x9a\xbb\x905&\x19\x8e
\xc4\x9a\x0b\x1f\xf8C\xf4\xb9\xc9\x85R\xc2\xbb\x8d\x07\x94
[R_\xf5z\x9fAl\x11\x9c\xbb\x9255\x08\x8e\xf6\xd6\x04'))
```

To make solving it easier I let ChatGPT make me dictionaries for each cloud with the distance to every other cloud based on the source code. After that I used it to make a tsp solver and copied over the XOR functions to decrypt the flag.

rainbom-solver.py

```
import hashlib

distances = {
    # dictonary for each cloud
}
def solve_tsp():
    n = 20
    start = 0

    dp = [[(float('inf'), -1) for _ in range(n)] for _ in range(1 << n)]
    dp[1 << start][start] = (0, -1)

    for mask in range(1 << n):
        for u in range(n):
            if not (mask & (1 << u)):
                continue
            current_cost, _ = dp[mask][u]
            if current_cost == float('inf'):
                continue
            for v in range(n):
                if mask & (1 << v):
                    continue
                new_mask = mask | (1 << v)
                new_cost = current_cost + distances[u][v]
                if new_cost < dp[new_mask][v][0]:
                    dp[new_mask][v] = (new_cost, u)

    full_mask = (1 << n) - 1
    min_cost = float('inf')
```

```
    last_node = -1

    for u in range(n):
        if u == start:
            continue
        cost, _ = dp[full_mask][u]
        total_cost = cost + distances[u][start]
        if total_cost < min_cost:
            min_cost = total_cost
            last_node = u

    if last_node == -1:
        return None

    # Reconstruct path
    path = []
    mask = full_mask
    current_node = last_node
    while current_node != -1:
        path.append(current_node)
        _, prev_node = dp[mask][current_node]
        mask ^= (1 << current_node)
        current_node = prev_node

    path.reverse()
    path.append(start)  # Return to start to complete the cycle

    return path


def xor(target, key):
    out = [c ^ key[i % len(key)] for i, c in enumerate(target)]
    return bytearray(out)

def key_from_path(path):
    return hashlib.sha256(str(path).encode()).digest()

def check_path(path, enc_flag):
    flag1 = xor(enc_flag, key_from_path(path))
    flag2 = xor(enc_flag, key_from_path(list(reversed(path))))
    if flag1.startswith(b"BtSCTF"):
        return flag1
    if flag2.startswith(b"BtSCTF"):
        return flag2
    return None
```

```
enc_flag = bytearray(b'\xc2\x92\xf9\xf66\xe8\xa5\xa6\x17\xb6mGE\xcfQ\x90Mk:\x9a\xbb\x905&\x1

path = solve_tsp()
if path:
    print("Optimal path:", path)
    print(str(path).encode())
    flag = check_path(path, enc_flag)
    if flag:
        print("Flag:", flag.decode())
    else:
        print("No flag found with this path.")
else:
    print("No path found.")
```

This then gave me the Flag with following Output.

```
Optimal path: [0, 12, 15, 2, 1, 5, 11, 14, 17, 7, 19, 13, 9, 10, 3, 8, 16, 18, 4, 6, 0]
Flag: BtSCTF{YOU_are_getting_20_percent_c00ler_with_this_one_!!_B)}
```