WriteUp ARKAVIDIA 9.0 2025

Hakuna Metadata

hush, mshazaw , jeelichan

Muhammad Ahsan Zaki Wiryawan
Axelle Chandra
Bayu Putra Ibana

Pyrev - 100 pts

Reverse Engineering

Your average baby python reversing chall, and it's not even .pyc!

note: tested on linux, running on windows might give different results

Author: Etynso

The challenge provides us this really confusing python code.

```
lllllllllllll, llllllllllllI, lllllllllllll, lllllllllllll, lllllllllllll, llllllllllllI = map, bytes, input, enumerate, print, list

from mmap import PAGESIZE as llIIlIIIIIIIl, PROT_EXEC as lIIlIlIIIIIll, PROT_WRITE as lIIlllIIIIIII, mmap as llIIIIIIIIIll, PROT_READ as llllIlllIIIlI
from ctypes import c_int as lIIIIIIIlIIl1, CFUNCTYPE as lIIlllllIIIII, addressof as lIIllIlIIIIII
from ctypes import c_void_p as IIIIIIIIIIIl1
from base64 import b64decode as lIllIIIlIIIIl
lIIIIIIIIIIIIII = lllIIIIIIIll(-1, llIIIIIIIIIl, prot=lllllllllllll | lIIIIIIIIIII | lIIIIIIIIIll)
lIIIIIIIlIIIIII = lIlIlllIIIIIl(llIIIIIIIIIl, lIIIIIIIIIIl)
llllllIIIIIIIII = IIIIIIIIIIIl.from_buffer(lIIIIIIIIIIIIII)
lIIIIIIIIIIIIII = lIIIIIIIlIIIIIl(lIIIlIlIIIIIl(lllllIlIIIIIIl))
lllllllllllllll.write(lIIlIIIlIIIIl('UVJWSInwSPfGAQAAAHUESIPAAUmJwEiJ+Egx0kjHwQQAAABI9/FIg/oAdBJIg/oBdBJIg/oCdBVIa/9l6xNIa/8b6w1Iaf+BAAAA6wRIa/8DSQ+v+EiB5/8AAABIifheWlnD'))
lIIIIlIlIIIIIIl = lllllllllllll('Input flag here: ').strip().encode()
if lllllllllllll(llllllllllll(llllllllllll(lambda lIIIIIIIIIIIIII: lllllllllllllll(lIIIIIIIIIIIIII[1], lIIIIIIIIIIIIII[0]), lllllllllll(llllllllllll(IIIIIIIIIIIIIII))))).hex() == 'c1f6
    lllllllllllll('Correct!')
else:
    lllllllllllll('Wrong!')
```

I was really confused, so I gave it to Deepseek to translate it into more clear code.

```
import mmap
from mmap import PAGESIZE, PROT_EXEC, PROT_WRITE, PROT_READ
from ctypes import c_int, CFUNCTYPE, addressof, c_void_p
from base64 import b64decode

# Set up execution environment identical to original
exec_mem = mmap.mmap(-1, PAGESIZE, prot=PROT_READ | PROT_WRITE | PROT_EXEC)
func_type = CFUNCTYPE(c_int, c_int, c_int)
buffer_ptr = c_void_p.from_buffer(exec_mem)
shellcode_func = func_type(addressof(buffer_ptr))
exec_mem.write(b64decode('UVJWSInwSPfGAQAAAHUESIPAAUmJwEiJ+Egx0kjHwQQAAABI9/FIg/oAdBJIg/oBdB
JIg/oCdBVIa/9l6xNIa/8b6w1Iaf+BAAAA6wRIa/8DSQ+v+EiB5/8AAABIifheWlnD'))

# Target hash to match
target_bytes =
bytes.fromhex('c1f6c5430aa35fa45753aa87d30c353089fc68111217baefc1c1933177770808f8f8e8e8acac2
4249c9cc9c97f7f3535ebeb67')

# Reconstruct flag byte-by-byte
flag = []
for position in range(len(target_bytes)):
    for byte in range(256):  # Try all possible bytes
        # Get transformed value (mod 256 to handle signed/unsigned conversion)
        transformed = shellcode_func(byte, position) % 256
        if transformed == target_bytes[position]:
            flag.append(byte)
            print(f"Position {position:2d}: Found {byte:03d} ({chr(byte)})")
            break
```

```
# Convert to string (handle non-printables)
print("\nFinal Flag:", bytes(flag).decode('utf-8', errors='replace'))
```

We're given a Python script that transforms user input using custom shellcode loaded into memory. The goal is to find the input that produces the target hash:

```
c1f6c5430aa35fa45753aa87d30c353089fc68111217baefc1c1933177770808f8f8e8e8aca
c24249c9cc9c97f7f3535ebeb67
```

In this line, we got an idea to do brute force to get the flag.

```
bytes(map(lambda pair: shellcode_func(pair[1], pair[0]),
enumerate(user_input))
```

So the insight from this line : The same byte at different positions produces different results (position-dependent) and each character is processed independently (no chaining/state).

After doing this brainstorming idea, we get back to Deepseek and ask it to make a python script to do the attack.

```
import mmap
from mmap import PAGESIZE, PROT_EXEC, PROT_WRITE, PROT_READ
from ctypes import c_int, CFUNCTYPE, addressof, c_void_p
from base64 import b64decode

# Create executable memory region
exec_mem = mmap.mmap(-1, PAGESIZE, prot=PROT_READ | PROT_WRITE | PROT_EXEC)

# Define function prototype: int func(int, int)
func_type = CFUNCTYPE(c_int, c_int, c_int)
# Create function pointer to our executable memory
buffer_ptr = c_void_p.from_buffer(exec_mem)
shellcode_func = func_type(addressof(buffer_ptr))

# Write decoded shellcode into memory
exec_mem.write(b64decode('UVJWSInwSPfGAQAAAHUESIPAAUmJwEiJ+Egx0kjHwQQAAABI9
/FIg/oAdBJIg/oBdBJIg/oCdBVIa/9l6xNIa/8b6w1Iaf+BAAAA6wRIa/8DSQ+v+EiB5/8AAABI
ifheWlnD'))

# Target hash to match
target_hex =
'c1f6c5430aa35fa45753aa87d30c353089fc68111217baefc1c1933177770808f8f8e8e8ac
ac24249c9cc9c97f7f3535ebeb67'
target_bytes = bytes.fromhex(target_hex)
```

```python
# Brute-force each character
flag = []
for idx in range(len(target_bytes)):
    found = False
    for byte in range(256):  # Try all possible byte values
        # Calculate result (take modulo 256 to get single byte)
        result = shellcode_func(byte, idx) % 256
        if result == target_bytes[idx]:
            flag.append(byte)
            found = True
            break
    if not found:
        flag.append(0)  # Fallback if no match found
    print(f"Position {idx}: Found byte {flag[-1]} -> '{chr(flag[-1])}'")

# Convert bytes to string (assuming UTF-8)
flag_str = bytes(flag).decode('utf-8', errors='replace')
print("\nFinal Flag:", flag_str)
```

```
Position 32: Found byte 104 -> 'h'
Position 33: Found byte 104 -> 'h'
Position 34: Found byte 104 -> 'h'
Position 35: Found byte 104 -> 'h'
Position 36: Found byte 116 -> 't'
Position 37: Found byte 116 -> 't'
Position 38: Found byte 116 -> 't'
Position 39: Found byte 116 -> 't'
Position 40: Found byte 116 -> 't'
Position 41: Found byte 116 -> 't'
Position 42: Found byte 63 -> '?'
Position 43: Found byte 63 -> '?'
Position 44: Found byte 63 -> '?'
Position 45: Found byte 63 -> '?'
Position 46: Found byte 63 -> '?'
Position 47: Found byte 63 -> '?'
Position 48: Found byte 63 -> '?'
Position 49: Found byte 63 -> '?'
Position 50: Found byte 125 -> '}'

Final Flag: ARKAV{its_just_python_riiiiggghhhhhhttttttt????????}
```

After run this code, we got the flag.

ARKAV{its_just_python_riiiiggghhhhhhttttttt????????}

Weird Format - 100 pts

Cryptography

Have you studied discrete mathematics ⊂(◉‿◉)つ

Author: *Etynso*

We're given a custom encryption scheme where the flag is hidden behind two ciphertexts, c1 and c2. The encryption uses a modulus n = p * q (two large 384-bit primes). To decrypt the flag, we need to exploit how c1 and c2 are computed.

After doing some mathematical works, this is the insight i got:

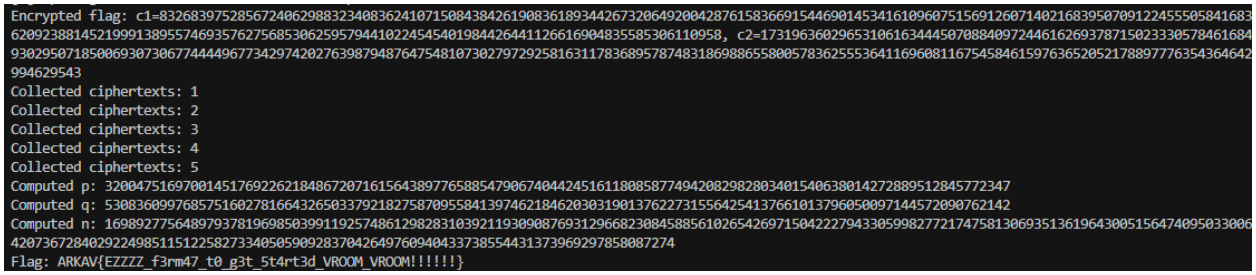
1. The Structure:
    a. The code computes g1 = g^(r1*(p-1)) mod n and g2 = g^(r2*(q-1)) mod n
    b. By Fermat's Little Theorem, since p and q are primes:
        i. g1 = 1 mod p (because g^(p-1) ≡ 1 mod p)
        ii. g2 ≡ 1 mod q.
2. The Encryption Process
    a. c1 = m * g1^s1 mod n → c1 ≡ m mod p (since g1^s1 ≡ 1 mod p).
    b. c2 = m * g2^s2 mod n → c2 ≡ m mod q.
3. How to Recover p and q
    a. If we encrypt m=1 multiple times, every c1 ≡ 1 mod p and c2 ≡ 1 mod q.
    b. The difference between two c1 values (e.g., c1_1 - c1_2) is a multiple of p.
    c. Taking the GCD of these differences reveals p. Repeat for c2 to get q

Then, we brought this to His Highness Deepseek, and asked him what we needed.

```
import math
from itertools import combinations
from pwn import *
from Crypto.Util.number import long_to_bytes

def extended_gcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = extended_gcd(b % a, a)
        return (g, x - (b // a) * y, y)

def crt(a, m, b, n):
    g, x, y = extended_gcd(m, n)
    assert g == 1, "Moduli must be coprime"
    return (a * y * n + b * x * m) % (m * n)

def get_gcd(numbers):
    gcd = 0
    for num in numbers:
```

```
        gcd = math.gcd(gcd, num)
    return gcd


# Connect to the server
conn = remote('20.195.43.216', 8555)

# Receive the encrypted flag
conn.recvuntil(b'Encrypted Flag: ')
encrypted_flag = conn.recvline().strip().decode()
c1f, c2f = eval(encrypted_flag)
print(f"Encrypted flag: c1={c1f}, c2={c2f}")

# Collect ciphertexts for m=1
c1_list = []
c2_list = []
num_requests = 5  # Number of encryption requests to collect

for _ in range(num_requests):
    conn.recvuntil(b'>> ')
    conn.sendline(b'1')
    conn.recvuntil(b'Plaintext: ')
    conn.sendline(b'1')
    conn.recvuntil(b'Encrypted: ')
    encrypted = conn.recvline().strip().decode()
    c1, c2 = eval(encrypted)
    c1_list.append(c1)
    c2_list.append(c2)
    print(f"Collected ciphertexts: {len(c1_list)}")

# Compute GCD of differences for c1 and c2
c1_differences = [abs(a - b) for a, b in combinations(c1_list, 2)]
p = get_gcd(c1_differences)
print(f"Computed p: {p}")

c2_differences = [abs(a - b) for a, b in combinations(c2_list, 2)]
q = get_gcd(c2_differences)
print(f"Computed q: {q}")

n = p * q
print(f"Computed n: {n}")

# Compute m mod p and m mod q
m_p = c1f % p
m_q = c2f % q

# Apply CRT
m_flag = crt(m_p, p, m_q, q)

# Convert to bytes and print the flag
flag = long_to_bytes(m_flag)
print(f"Flag: {flag.decode()}")
```

After running that code, we got the flag.



ARKAV{EZZZZ_f3rm47_t0_g3t_5t4rt3d_VROOM_VROOM!!!!!!}

Beat Frendy - 100 pts

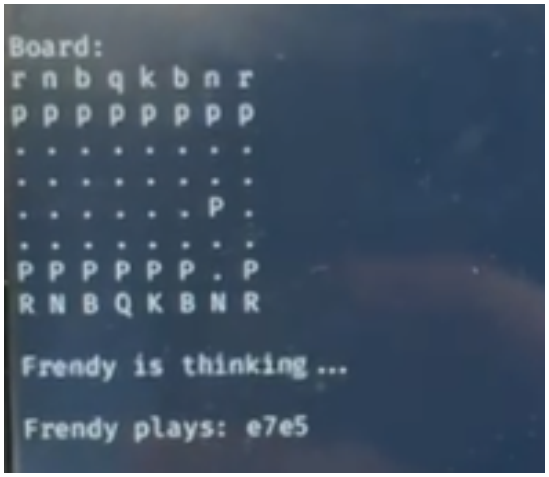
Miscellaneous

Beat me using g2g4 opening.

Author: frennn

To start things off, I do apologize for the bad quality of the documentation. I missed my chance to document it through screenshots, so some of this documentation may look weird.

Upon connecting to the server, we would instantly see a prompt inviting me to beat them in a chess match.



The first thing I thought of was of course the chess bot stockfish, and coincidentally there's a stockfish library in kali-linux. So I installed it and implemented it in my script here to play chess for me and acquire the flag.

```
import time
import socket

# Server details
HOST = "20.195.43.216"
PORT = 8091

# Stockfish engine path
STOCKFISH_PATH = "/usr/bin/stockfish"

def connect_to_server():
   s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
   s.connect((HOST, PORT))
   return s

def send_move(s, move):
```

```
    print(f"Sending move: {move}")
    s.sendall(move.encode() + b'\n')

def receive_data(s):
    data = b""
    while True:
        try:
            chunk = s.recv(4096)
            if not chunk:
                break
            data += chunk
            if b"Your move" in data or b"Frendy plays" in data:
                break
        except socket.timeout:
            break
    return data.decode(errors='ignore')

def extract_opponent_move(data):
    lines = data.split("\n")
    for line in lines:
        if "Frendy plays:" in line:
            return line.split(": ")[-1].strip()
    return None

def main():
    s = connect_to_server()
    print("Connected to server.")
    time.sleep(1)  # Wait before sending the first move

    send_move(s, "g2g4")
    time.sleep(2.5)  # Give the server time to respond

    stockfish = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    stockfish.connect(("localhost", 5000))  # Ensure Stockfish is
running
    stockfish.sendall(b"uci\n")
    stockfish.sendall(b"isready\n")

    moves = ["g2g4"]

    while True:
        data = receive_data(s)
        print("Server:", data)

        opponent_move = extract_opponent_move(data)
        if not opponent_move:
            print("Failed to extract opponent's move, retrying...")
            continue

        print(f"Opponent played: {opponent_move}")
        moves.append(opponent_move)
```

```
        time.sleep(0.5)  # Pause before sending Stockfish's move

        stockfish.sendall(f"position startpos moves {'
'.join(moves)}\n".encode())
        stockfish.sendall(b"go movetime 500\n")

        stockfish_response = b""
        while True:
            chunk = stockfish.recv(4096)
            stockfish_response += chunk
            if b"bestmove" in stockfish_response:
                break

        best_move =
stockfish_response.decode(errors='ignore').split("bestmove
")[-1].split()[0]
        send_move(s, best_move)
        moves.append(best_move)

        time.sleep(2.5)  # Wait for server's response

    stockfish.close()
    s.close()
    print("Game over.")

if __name__ == "__main__":
    main()
```

Flag: ARKAV{hanya_sepuh_yang_berani_make_opening_g2-g4}

Minji Anak UNPAD - 424 pts

Miscellaneous
Last night, I had a call with my gf, Minji. She told me that she was going to perform at Padjajaran University tonight. She said that she'll send me an important message, But suddenly, her phone was disconnected, ugh.
Author: w1ntr

We are given a Python script implementing a custom encryption scheme for a flag. The encryption uses XOR and rotations with keys derived from HMAC-SHA256. The critical constraint is that no ciphertext byte can match the plaintext (flag) byte in the same position. Our goal is to exploit this constraint to recover the original flag.

Because ciphertext bytes cannot be same as flag bytes, for each position in ciphertexts, the flag byte is the one that never appear in all ciphertexts we collect.

To solve this we need to get as many ciphertexts as we can and find missing Bytes for each byte by looking for the byte which never show up in that position. Our best friend Deepseek help us in the scripting part.

```
import base64
import json
import asyncio

async def collect_ciphertexts(host, port, max_ciphertexts=5000):
    ciphertexts = []
    for count in range(max_ciphertexts):
        try:
            reader, writer = await asyncio.open_connection(host, port)
            # Read the initial prompt
            await reader.readuntil(b"!\n")
            # Send request
            writer.write(json.dumps({"msg": "request"}).encode() + b'\n')
            await writer.drain()
            # Read response
            data = await reader.readuntil(b'\n')
            response = json.loads(data.decode().strip())
            if 'ciphertext' in response:
                ct = base64.b64decode(response['ciphertext'])
                ciphertexts.append(ct)
                print(f"Collected {len(ciphertexts)} ciphertexts", end='\r')
            writer.close()
            await writer.wait_closed()
        except Exception as e:
            print(f"Error: {e}")
            continue
    return ciphertexts
```

```python
def find_flag(ciphertexts):
    if not ciphertexts:
        return None
    flag_len = len(ciphertexts[0])
    possible_bytes = [set() for _ in range(flag_len)]
    for ct in ciphertexts:
        for i, byte in enumerate(ct):
            possible_bytes[i].add(byte)
    flag = bytearray()
    for s in possible_bytes:
        all_bytes = set(range(256))
        missing = all_bytes - s
        if len(missing) != 1:
            print("\nWarning: Missing byte not unique. Collect more samples.")
            flag.append(0)
        else:
            flag.append(missing.pop())
    return bytes(flag)

async def main():
    host = '20.195.43.216'
    port = 8090
    ciphertexts = await collect_ciphertexts(host, port, 5000)
    flag = find_flag(ciphertexts)
    print("\nFlag:", flag.decode())

if __name__ == "__main__":
    asyncio.run(main())
```

After running the code we got the flag:

```
PS C:\Users\VICTUS> & C:/Users/VICTUS/AppDa
Collected 4426 ciphertexts
Flag: ARKAV{mmm_hmm_wh4t's_y0ur_ET4?!}
```

ARKAV{mmm_hmm_wh4t’s_y0ur_ET4?!}

Wibu - 100 pts

Katanya di ITB banyak wibunya.

Author: msfir

We are given a .beam file and an encoded string:
“渮珁禂绰统绮苭肯蔆褄纯蕯襆堇腡襯蝩衜菡揵涖”

To decompile the .beam file, I used **iex**.
Then i used this command

:beam_disasm.file(Wibufication)

The result was this **erlang** code:

```
{:beam_file, Wibufication,
 [
   {:__info__, 1, 2},
   {:main, 0, 13},
   {:module_info, 0, 18},
   {:module_info, 1, 20}
 ], [vsn: [35892632921203535753721582434066279718]],
 [
   version: ~c"8.2.6.4",
   options: [:no_spawn_compiler_process, :from_core, :no_core_prepare,
           :no_auto_import],
   source: ~c"/home/msfir/Documents/ProbsetCTFArkav9/quals/reverse
engineering/Wibu/src/wibufication.exs"
 ],
 [
   {:function, :__info__, 1, 2,
           [
           {:label, 1},
           {:line, 0},
           {:func_info, {:atom, Wibufication}, {:atom, :__info__}, 1},
           {:label, 2},
           {:select_val, {:x, 0}, {:f, 9},
           {:list,
           [
           atom: :attributes,
           f: 8,
           atom: :compile,
           f: 8,
           atom: :deprecated,
```

```
        f: 7,
        atom: :exports_md5,
        f: 6,
        atom: :functions,
        f: 5,
        atom: :macros,
        f: 7,
        atom: :md5,
        f: 8,
        atom: :module,
        f: 4,
        atom: :struct,
        f: 3
        ]}},
        {:label, 3},
        {:move, {:atom, nil}, {:x, 0}},
        :return,
        {:label, 4},
        {:move, {:atom, Wibufication}, {:x, 0}},
        :return,
        {:label, 5},
        {:move, {:literal, [main: 0]}, {:x, 0}},
        :return,
        {:label, 6},
        {:move,
        {:literal,
        <<157, 208, 121, 73, 0, 9, 40, 187, 215, 180, 75, 123, 132, 8, 152,
        227>>}, {:x, 0}},
        :return,
        {:label, 7},
        {:move, nil, {:x, 0}},
        :return,
        {:label, 8},
        {:move, {:x, 0}, {:x, 1}},
        {:move, {:atom, Wibufication}, {:x, 0}},
        {:call_ext_only, 2, {:extfunc, :erlang, :get_module_info, 2}},
        {:label, 9},
        {:call_only, 1, {Wibufication, :"-inlined-__info__/1-", 1}}
        ]},
  {:function, :convert, 1, 11,
        [
        {:line, 1},
        {:label, 10},
        {:func_info, {:atom, Wibufication}, {:atom, :convert}, 1},
        {:label, 11},
        {:allocate, 0, 1},
        {:line, 2},
        {:call_ext, 1, {:extfunc, String, :codepoints, 1}},
        {:move, {:integer, 2}, {:x, 2}},
        {:move, nil, {:x, 3}},
```

```
        {:move, {:integer, 2}, {:x, 1}},
        {:line, 3},
        {:call_ext, 4, {:extfunc, Enum, :chunk_every, 4}},
        {:test_heap, {:alloc, [words: 0, floats: 0, funs: 1]}, 1},
        {:make_fun3, {Wibufication, :"-convert/1-fun-0-", 1}, 0, 14157147,
        {:x, 1}, {:list, []}},
        {:line, 4},
        {:call_ext_last, 2, {:extfunc, Enum, :map_join, 2}, 0}
        ]},
  {:function, :main, 0, 13,
        [
        {:line, 5},
        {:label, 12},
        {:func_info, {:atom, Wibufication}, {:atom, :main}, 0},
        {:label, 13},
        {:allocate, 0, 0},
        {:line, 6},
        {:call_ext, 0, {:extfunc, System, :argv, 0}},
        {:move, {:literal, " "}, {:x, 1}},
        {:line, 7},
        {:call_ext, 2, {:extfunc, Enum, :join, 2}},
        {:line, 8},
        {:call, 1, {Wibufication, :convert, 1}},
        {:line, 9},
        {:call_ext_last, 1, {:extfunc, IO, :puts, 1}, 0}
        ]},
  {:function, :process_chunk, 1, 15,
        [
        {:line, 10},
        {:label, 14},
        {:func_info, {:atom, Wibufication}, {:atom, :process_chunk}, 1},
        {:label, 15},
        {:test, :is_nonempty_list, {:f, 14}, [x: 0]},
        {:get_list, {:x, 0}, {:x, 1}, {:x, 2}},
        {:test, :is_nonempty_list, {:f, 16}, [x: 2]},
        {:get_list, {:x, 2}, {:x, 3}, {:x, 2}},
        {:test, :is_nil, {:f, 14}, [x: 2]},
        {:allocate, 2, 4},
        {:init_yregs, {:list, [y: 0]}},
        {:move, {:x, 3}, {:y, 1}},
        {:move, {:x, 1}, {:x, 0}},
        {:line, 11},
        {:call_ext, 1, {:extfunc, :binary, :first, 1}},
        {:line, 12},
        {:gc_bif, :*, {:f, 0}, 1, [x: 0, integer: 128], {:y, 0}},
        {:move, {:y, 1}, {:x, 0}},
        {:move, {:y, 0}, {:y, 1}},
        {:trim, 1, 1},
        {:line, 13},
        {:call_ext, 1, {:extfunc, :binary, :first, 1}},
```

```
        {:gc_bif, :+, {:f, 0}, 1, [{:tr, {:y, 0}, {:t_number, :any}}, {:x, 0}],
        {:x, 0}},
        {:line, 14},
        {:gc_bif, :+, {:f, 0}, 1,
        [{:tr, {:x, 0}, {:t_number, ...}}, {:integer, 19968}], {:x, 0}},
        {:call_last, 1, {Wibufication, :"-process_chunk/1-fun-0-", 1}, 1},
        {:label, 16},
        {:test, :is_nil, {:f, 14}, [x: 2]},
        {:test_heap, 2, 2},
        {:put_list, {:x, 1}, {:literal, [...]}, {:x, ...}},
        {:call_only, 1, {Wibufication, ...}}
        ]},
{:function, :module_info, 0, 18,
        [
        {:line, 0},
        {:label, 17},
        {:func_info, {:atom, Wibufication}, {:atom, :module_info}, 0},
        {:label, 18},
        {:move, {:atom, Wibufication}, {:x, 0}},
        {:call_ext_only, 1, {:extfunc, :erlang, :get_module_info, 1}}
        ]},
{:function, :module_info, 1, 20,
        [
        {:line, 0},
        {:label, 19},
        {:func_info, {:atom, Wibufication}, {:atom, :module_info}, 1},
        {:label, 20},
        {:move, {:x, 0}, {:x, 1}},
        {:move, {:atom, Wibufication}, {:x, 0}},
        {:call_ext_only, 2, {:extfunc, :erlang, :get_module_info, 2}}
        ]},
{:function, :"-process_chunk/1-fun-0-", 1, 22,
        [
        {:line, 15},
        {:label, 21},
        {:func_info, {:atom, Wibufication}, {:atom, :"-process_chunk/1-fun-0-"},
        1},
        {:label, 22},
        {:bs_create_bin, {:f, 0}, 0, 1, 1, {:x, 0},
        {:list, [{:atom, :utf8}, 1, 0, nil, {:x, 0}, {:atom, :undefined}]}},
        :return
        ]},
{:function, :"-convert/1-fun-0-", 1, 24,
        [
        {:line, 4},
        {:label, 23},
        {:func_info, {:atom, Wibufication}, {:atom, :"-convert/1-fun-0-"}, 1},
        {:label, 24},
        {:call_only, 1, {Wibufication, :process_chunk, 1}}
        ]},
```

```
  {:function, :"-inlined-__info__/1-", 1, 26,
          [
          {:line, 0},
          {:label, 25},
          {:func_info, {:atom, Wibufication}, {:atom, :"-inlined-__info__/1-"}, 1},
          {:label, 26},
          {:jump, {:f, 25}}
          ]}
 ]}
```

Then , I converted the erlang code to **elixir** code.

```
defmodule Wibufication do
  @moduledoc
  def convert(str) do
          str
          |> String.codepoints()
          |> Enum.chunk_every(2, 2, nil)
          |> Enum.map_join(&process_chunk/1)
  end

  def main do
          System.argv()
          |> Enum.join(" ")
          |> convert()
          |> IO.puts()
  end

  # Processes a two-codepoint chunk.
  def process_chunk([a, b]) do
          a_val = :binary.first(a)
          # The BEAM code applies a trim on the second element before extracting its code.
          b_val = :binary.first(String.trim(b))
          codepoint = a_val * 128 + b_val + 19968
          <<codepoint::utf8>>
  end

  # Fallback when there is only one codepoint.
  def process_chunk([a]) do
          a_val = :binary.first(a)
          codepoint = a_val * 128 + 19968
          <<codepoint::utf8>>
  end
end
```

From then on, i modified the code so that i can reverse engineer the process and decode the string that was given at the start:

```
defmodule Decoder do
  @moduledoc

  def decode(encoded) do
        encoded
        |> String.graphemes()
        |> Enum.map(&reverse_char/1)
        |> List.flatten()
        |> Enum.join("")
  end

  defp reverse_char(char) do
        cp = char |> String.to_charlist() |> hd
        value = cp - 19968
        a_val = div(value, 128)
        b_val = rem(value, 128)

        # If b_val is 0, assume this chunk was produced from a single input codepoint.
        if b_val == 0 do
        [<<a_val::utf8>>]
        else
        [<<a_val::utf8>>, <<b_val::utf8>>]
        end
  end

  def main do
        # Accept the encoded string from the command-line arguments.
        encoded =
        System.argv()
        |> Enum.join(" ")

        IO.puts(decode(encoded))
  end
end

Decoder.main()
```

Then, in the command line:

elixir solve.ex "滒珁祻绰统绮苭肯蓡蕧纯蕯襥葟腡襯蝩衟菡蓵淽"

Then the result, i got the flag:

ARKAV{apa_anime/manga/novel_favorit_kamu?}