Nama Tim : no. fasilkom no worries
Nama Kasus : {Nama Kasus}
Nama Ketua – Gafna Al Faatiha Prabowo
Nama Anggota 1 – Syafran Abdillah Erdin
Nama Anggota 2 – rizkywildansani

KRIPTOGRAFI

1. Nama Challenge

kujou

Diberikan sebuah file python seperti ini:

```python
import random
import signal

FLAG = open('flag.txt').read()

def temper(y):
    y = y ^ (y >> 11)
    y = y ^ ((y << 7) & (0x9d2c5680))
    y = y ^ ((y << 15) & (0xefc60000))
    y = y ^ (y >> 18)
    return y


def main():
    # Let's do more random shenanigans, with spices
    leak_mask = int(input("Enter leak mask: "), 16)
    if leak_mask.bit_count() > 8 or leak_mask <= 0:
        print("Skill issue")
        exit(0)

    for _ in range(50):
        for _ in range(random.getrandbits(32) & 15):
            random.getrandbits(32)
        secret = random.getrandbits(32)
        out = temper(secret)
        print(f'Your leak: {out & leak_mask}')

        res = int(input("Just gimme one bit: "))
```

```
        if res                                          != secret & 1:
            print("Skill issue")
            exit(0)

        random.seed(random.getrandbits(32))

    print("Well played good sir")
    print(FLAG)


if __name__ == '__main__':
    try:
        signal.alarm(88)
        main()
    except Exception as e:
        print(e)
        exit(1)
```

Program ini adalah sebuah challenge berbasis pseudorandom number generator (PRNG) dan bit-level reverse engineering. Tujuannya memvalidasi apakah user bisa menebak bit LSB (Least Significant Bit) dari angka rahasia sebanyak 50 kali berdasarkan leak yang sudah di-temper.

Mekanisme Kerja:

1. **Input:** User memasukkan `leak_mask` dalam format heksadesimal. Mask ini digunakan untuk menyaring bit dari output PRNG.
   - Harus memiliki <= 8 bit aktif (`bit_count() <= 8`) dan > 0.
2. **Loop 50 Kali:**
   - PRNG (`random.getrandbits(32)`) digunakan untuk membuang sejumlah angka acak.
   - Satu angka acak (`secret`) diambil.
   - `secret` di-*temper* (menggunakan transformasi ala Mersenne Twister).
   - Leak ditampilkan: `out & leak_mask`.
   - User menebak bit terakhir dari `secret`.
   - Jika salah → keluar.
3. **Seed PRNG diacak ulang setiap iterasi** dengan hasil PRNG sebelumnya → membuat prediksi lebih sulit.

Berikut solver nya

```python
import socket
import re
import time
import random

# --- Temper and Untemper functions (Validated as correct inverses by local test) ---
def temper(y):
    y &= 0xFFFFFFFF
    y = y ^ (y >> 11)
    y &= 0xFFFFFFFF
    y = y ^ ((y << 7) & (0x9d2c5680))
    y &= 0xFFFFFFFF
    y = y ^ ((y << 15) & (0xefc60000))
    y &= 0xFFFFFFFF
    y = y ^ (y >> 18)
    y &= 0xFFFFFFFF
    return y

def untemper(y_out):
    y = y_out & 0xFFFFFFFF
    y = y ^ (y >> 18)
    y &= 0xFFFFFFFF
    res_iter = y
    val_orig_y_for_s15 = y
    for _ in range(2):
        res_iter = val_orig_y_for_s15 ^ ((res_iter << 15) & 0xefc60000)
        res_iter &= 0xFFFFFFFF
    y = res_iter
    res_iter = y
    val_orig_y_for_s7 = y
    for _ in range(4):
        res_iter = val_orig_y_for_s7 ^ ((res_iter << 7) & 0x9d2c5680)
        res_iter &= 0xFFFFFFFF
    y = res_iter
    y = y ^ (y >> 11) ^ (y >> 22)
    y &= 0xFFFFFFFF
    return y

# --- Global chosen leak mask (to be set by local test) ---
chosen_leak_mask_hex = ""
chosen_leak_mask_int = 0

# --- Local Test Function (Modified to set global mask if valid) ---
def local_test_and_set_mask():
    print("[LOCAL_TEST] Starting temper/untemper validation...")
```

```
    test_cases = [0, 1, 2, 0xFFFFFFFF, 0xAAAAAAAA, 0x55555555]
    for _ in range(10000):
        test_cases.append(random.getrandbits(32))

    passed_count = 0
    failed_count = 0
    for i, secret_val in enumerate(test_cases):
        secret_val &= 0xFFFFFFFF
        tempered_val = temper(secret_val)
        untempered_val = untemper(tempered_val)
        if secret_val == untempered_val:
            passed_count += 1
        else:
            failed_count += 1
            if failed_count <= 5:
                print(f"[LOCAL_TEST_FAIL] Mismatch for secret={hex(secret_val)}")

    print(f"[LOCAL_TEST] Temper/Untemper Function Test: Passed={passed_count}, Failed={failed_count}")

    if failed_count > 0:
        print("[LOCAL_TEST_CRITICAL_FAIL] Untemper function is incorrect. Cannot proceed.")
        return False

    print("[LOCAL_TEST] All temper/untemper inversion tests passed.")

    # Calculate the dependency mask for secret LSB
    dep_mask = 0
    for j in range(32):
        t_test = 1 << j
        s_test = untemper(t_test)
        if (s_test & 1):
            dep_mask |= (1 << j)

    calculated_dep_mask_bit_count = dep_mask.bit_count()
    print(f"[LOCAL_TEST] Calculated dep_mask_for_s0 = {hex(dep_mask)}")
    print(f"[LOCAL_TEST] Bit count of this mask is: {calculated_dep_mask_bit_count}")

    global chosen_leak_mask_hex, chosen_leak_mask_int
    if calculated_dep_mask_bit_count > 0 and calculated_dep_mask_bit_count <= 8:
```

```
        print(f"[LOCAL_TEST] dep_mask_for_s0 ({hex(dep_mask)}) is valid
({calculated_dep_mask_bit_count} bits). This will be used.")
        chosen_leak_mask_hex = hex(dep_mask) # format like "0x..."
        if chosen_leak_mask_hex.startswith("0x"): # Ensure it's just
hex digits for server
            chosen_leak_mask_hex = chosen_leak_mask_hex[2:]
        chosen_leak_mask_int = dep_mask
        return True
    elif calculated_dep_mask_bit_count == 0:
        print(f"[LOCAL_TEST_CRITICAL_FAIL] dep_mask_for_s0 is
{hex(dep_mask)} (0 bits)! s_0 appears constant 0? Problem with untemper
or test logic.")
        return False
    else: # > 8 bits
        print(f"[LOCAL_TEST_CRITICAL_FAIL] dep_mask_for_s0
{hex(dep_mask)} has {calculated_dep_mask_bit_count} bits (>8). Cannot
be used as leak_mask directly by server.")
        return False

# --- Network Configuration ---
HOST = "103.163.139.198"
PORT = 8049
TIMEOUT_SECONDS = 15

# --- Network Helper Functions (assumed stable) ---
def recv_until_prompt(sock, expected_prompt_text):
    buffer = b""
    prompt_bytes = expected_prompt_text.encode('utf-8')
    start_time = time.time()
    while True:
        if time.time() - start_time > TIMEOUT_SECONDS:
            print(f"[ERROR] Timeout waiting for prompt:
'{expected_prompt_text}', Buffer: {buffer!r}")
            raise socket.timeout(f"Custom timeout for prompt:
{expected_prompt_text}")
        try:
            chunk = sock.recv(4096)
            if not chunk:
                print(f"[ERROR] Connection closed (prompt), Buffer:
{buffer!r}")
                raise EOFError("Connection closed while waiting for
prompt")
            buffer += chunk
            if buffer.decode('utf-8',
errors='ignore').endswith(expected_prompt_text):
```

```
                return buffer.decode('utf-8', errors='ignore')
        except socket.timeout:
            print(f"[ERROR] Socket s.settimeout() (prompt) for
'{expected_prompt_text}', Buffer: {buffer!r}")
            raise
        except UnicodeDecodeError as ude:
            print(f"[ERROR] UnicodeDecodeError (prompt): {ude}, Buffer:
{buffer!r}")
            continue

def recv_line(sock):
    buffer = b""
    start_time = time.time()
    while True:
        if time.time() - start_time > TIMEOUT_SECONDS:
            print(f"[ERROR] Timeout waiting for line, Buffer:
{buffer!r}")
            raise socket.timeout("Custom timeout for line")
        try:
            byte = sock.recv(1)
            if not byte:
                print(f"[ERROR] Connection closed (line), Buffer:
{buffer!r}")
                if buffer: return buffer.decode('utf-8',
errors='ignore').strip()
                raise EOFError("Connection closed, no partial line")
            buffer += byte
            if byte == b"\n":
                return buffer.decode('utf-8', errors='ignore').strip()
        except socket.timeout:
            print(f"[ERROR] Socket s.settimeout() (line), Buffer:
{buffer!r}")
            raise
        except UnicodeDecodeError as ude:
            print(f"[ERROR] UnicodeDecodeError (line): {ude}, Buffer:
{buffer!r}")
            continue

def recv_eagerly(sock, original_timeout):
    try:
        sock.settimeout(0.2)
        data_chunks = []
        while True:
            chunk = sock.recv(4096)
            if not chunk: break
```

```
            data_chunks.append(chunk)
    except (socket.timeout, BlockingIOError): pass
    finally:
        sock.settimeout(original_timeout)
    if data_chunks: return b"".join(data_chunks)
    return b""

# --- Main Solve Logic (Updated Prediction) ---
def solve():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.settimeout(TIMEOUT_SECONDS)
    original_socket_timeout = s.gettimeout()

    global chosen_leak_mask_hex, chosen_leak_mask_int # Use the
globally set mask

    try:
        print(f"[INFO] Connecting to {HOST}:{PORT} (timeout
{TIMEOUT_SECONDS}s)...")
        s.connect((HOST, PORT))
        print("[INFO] Connected.")

        initial_server_output = recv_until_prompt(s, "Enter leak mask:
")
        print(f"[SERVER_MSG] {initial_server_output.strip()}")

        print(f"[CLIENT_MSG] Sending leak mask: {chosen_leak_mask_hex}
(int: {hex(chosen_leak_mask_int)})")
        s.sendall(chosen_leak_mask_hex.encode('utf-8') + b"\n")

        for i in range(50):
            current_round_start_time = time.time()
            print(f"\n--- Round {i+1}/50 ---")

            leak_line = recv_line(s)
            print(f"[SERVER_MSG] ({time.time()-
current_round_start_time:.2f}s) Leak line: '{leak_line}'")

            match = re.search(r'Your leak: (\d+)', leak_line)
            if not match:
                print(f"[ERROR] Could not parse leak line:
'{leak_line}'")
                print("[INFO] Dumping remaining data:")
                print(recv_eagerly(s,
original_socket_timeout).decode('utf-8', errors='ignore'))
```

```
                return False

            leaked_value = int(match.group(1)) # This is temper(secret)
& chosen_leak_mask_int
            print(f"[INFO] Parsed leaked_value (decimal):
{leaked_value} (hex: {hex(leaked_value)})")

            # NEW PREDICTION LOGIC: Parity of the leaked_value
            # leaked_value itself contains the bits of temper(secret)
that we care about,
            # as it's already masked by chosen_leak_mask_int (which is
dep_mask_for_s0).
            predicted_bit = leaked_value.bit_count() & 1

            print(f"[INFO] Predicted LSB of secret (s_0):
{predicted_bit} (based on parity of leaked_value)")

            bit_prompt_output = recv_until_prompt(s, "Just gimme one
bit: ")
            print(f"[SERVER_MSG] ({time.time()-
current_round_start_time:.2f}s) Bit prompt:
'{bit_prompt_output.strip()}'")

            s.sendall(str(predicted_bit).encode('utf-8') + b"\n")
            print(f"[CLIENT_MSG] Sent predicted bit: {predicted_bit}")

        print("\n[INFO] Completed 50 rounds successfully.")
        print("[INFO] Attempting to read the flag...")

        flag_msg1 = recv_line(s)
        print(f"[FLAG_PART_OR_MSG] {flag_msg1}")
        flag_msg2 = recv_line(s)
        print(f"[FLAG_PART_OR_MSG] {flag_msg2}")

        print("[INFO] Reading any final data...")
        final_data = recv_eagerly(s, original_socket_timeout)
        if final_data: print(f"[SERVER_MSG_FINAL]
{final_data.decode('utf-8', errors='ignore')}")
        print("[INFO] Interaction finished.")
        return True

    except socket.timeout as ste:
        print(f"[FATAL] Socket operation timed out: {ste}")
    except EOFError as eofe:
        print(f"[FATAL] Connection closed unexpectedly: {eofe}")
```

```
    except
ConnectionRefusedError:
        print(f"[FATAL] Connection to {HOST}:{PORT} refused.")
    except ConnectionResetError:
        print(f"[FATAL] Connection reset by peer.")
    except Exception as e:
        print(f"[FATAL] Unexpected error: {type(e)._name_} - {e}")
        import traceback
        traceback.print_exc()
    finally:
        print("[INFO] Closing socket.")
        s.close()
    return False

if _name_ == '_main_':
    if local_test_and_set_mask(): # This now sets the global mask
variables
        print("\n[INFO] Local tests passed and leak mask configured.
Proceeding with server interaction.\n")
        solve()
    else:
        print("\n[INFO] Local tests failed or yielded an unusable
dependency mask. Cannot solve.")
```

```
[INFO] Completed 50 rounds successfully.
[INFO] Attempting to read the flag...
[FLAG_PART_OR_MSG] Well played good sir
[FLAG_PART_OR_MSG] IFEST13{https://www.youtube.com/watch?v=hX_3P0TBGP4&flag=28c24126cb289846}
```

Flag:

IFEST13{https://www.youtube.com/watch?v=hX_3P0TBGP4&flag=28c24126cb289846}

Note: woilah tuan rumah malu maluin :v