rev_shadow_ledger — Flag-in-Plaintext, with Theatre

HackTheBox Reversing · binary asks for an 8-digit hex auth key, prints “AUTH FAIL” on a miss, and then prints the flag anyway · the test is psychological, not technical

Executive Summary

A 26 KB stripped x86-64 ELF presenting as a “verification node” that asks for an 8-digit hex authentication key (`shadow_ledger <key>`). The key check is real — a 32-bit comparison against a fixed constant — but solving the key check buys you nothing. **The flag is stored as plaintext in `.rodata` and is printed on every code path, including the failure path.** Total time-to-flag with a competent toolkit: less than 30 seconds.

Flag: `HTB{c0unt_th3_sh4d0ws_0r_d13_try1ng}`

This writeup is short on purpose. There is no engineering depth to extract here; padding it would misrepresent the challenge. The value of the document is in the *lesson* the challenge teaches, not in any technique.

The thirty-second solve

```
$ strings -n 6 shadow_ledger | grep -iE 'HTB|FLAG|CTF\{'
HTB{c0unt_th3_sh4d0ws_0r_d13_try1ng}
```

Done. Everything below this point is for completeness — what the binary actually does, and why the flag leaks on every path.

Reconnaissance

```
$ file shadow_ledger
shadow_ledger: ELF 64-bit LSB pie executable, x86-64, version 1
  (SYSV), dynamically linked, stripped

$ ./shadow_ledger
[shadow_ledger] usage: shadow_ledger <8-hex-key>

$ ./shadow_ledger 00000000
[shadow_ledger] AUTH FAIL  invalid key
[shadow_ledger] flag: HTB{c0unt_th3_sh4d0ws_0r_d13_try1ng}
[shadow_ledger] exit -1
```

Yes, really. The program prints the flag on the failure path. The `AUTH FAIL` line is for atmosphere; the flag line follows it unconditionally. The challenge name itself — “count the shadows, or die trying” — telegraphs the lesson: every `printf` is part of the show, and the analyst's job is to count them all, not to chase the one with auth flavor.

The key check (for completeness)

At `0x40114C` the `main` function parses `argv[1]` with `strtoul(..., 16)` and compares the result against `0xDEADBEEF`:

```
0x401151:  call   strtoul@plt
0x401156:  cmp    eax, 0xDEADBEEF
0x40115b:  je     auth_ok
0x40115d:  lea    rdi, [rip+xxxx]    ; "[shadow_ledger] AUTH FAIL  invalid key"
0x401164:  call   puts@plt
0x401169:  lea    rdi, [rip+yyyy]    ; "[shadow_ledger] flag: HTB{...}"
0x401170:  call   puts@plt           ; <-- flag printed unconditionally
0x401175:  mov    eax, -1
0x40117a:  ret
auth_ok:
0x40117c:  lea    rdi, [rip+zzzz]    ; "[shadow_ledger] AUTH OK"
0x401183:  call   puts@plt
0x401188:  lea    rdi, [rip+yyyy]    ; same flag print
```

```
0x40118f:  call   puts@plt
0x401194:  xor    eax, eax
0x401199:  ret
```

Both branches print the flag from `.rodata:0x402008`. No XOR, no decryption, no compile-time hashing, no runtime decode. Plaintext bytes followed by a `puts`. `0xDEADBEEF` is the correct key (any 8-digit hex parse that yields that value — `DEADBEEF`, `deadbeef`, `0xDEADBEEF`) takes the "happy" branch. The two branches differ *only* in the banner line above the flag.

Why this earned its rating ("Very Easy")

Two-pronged lesson:

(1) Always grep first. `strings -n 6 <bin> | grep -iE 'HTB|FLAG|CTF\{'` should be reflex on every binary. If the flag is plaintext, you're done before you've opened a disassembler. This sounds trivial, but on a timed CTF the 30 seconds you save by doing this every time pays back across the event.

(2) Read every code path, not just the "happy" one. Plenty of harder challenges use this same pattern: the failure branch prints the flag, only after a misleading `AUTH FAIL` string that anchors the analyst onto "I need to solve the auth." Tunnel vision is the real test. The author isn't testing your assembly fluency; they're testing whether you can resist the natural pull toward the code-shaped puzzle.

Artifacts & takeaways

One-liner. `strings -n 6 <bin> | grep -iE 'HTB|FLAG|CTF\{'`. Burn it into muscle memory. Variants worth keeping ready:

Tool	Why
`strings -a`	Includes all sections, not just `.rodata` — occasionally the flag sits in `.data`.
`strings -e l`	Catches UTF-16LE strings (rare on Linux, common on Windows binaries); some CTFs hide flags this way.
`grep -aE 'HTB\{[!-~]+\}'`	Regex anchored to the flag-shape avoids false hits in banner strings that contain "HTB".
`radare2 -qc 'izz' <bin>`	Pulls strings from *every* section including ones `strings` misses on some platforms.

The psychological half of the puzzle. The harder you stare at the key-check disassembly, the less likely you are to glance at the trivially-visible flag-print branch directly above it. `shadow_ledger` is a calibration exercise for that exact failure mode. Treat it as one.

Toolchain. `revsolve auto` on this binary ranks the printable-strings hypothesis above the auth-check hypothesis specifically because the flag-shaped string is present at high confidence in `.rodata`. Trust the ranker on trivial cases — it exists to short-circuit them.